HP Unified Functional Testing

Software Version: 12.53

Tutorial

Document Release Date: June 2016 | Software Release Date: June 2016

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

Google™ and Google Maps™ are trademarks of Google Inc

Intel® and Pentium® are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft®, Windows®, Windows® XP, and Windows Vista ® are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com.

This site requires that you register for an HPE Passport and sign in. To register for an HPE Passport ID, go to https://softwaresupport.hpe.com and click **Register**.

Support

Visit the HPE Software Support Online web site at: https://softwaresupport.hpe.com

This web site provides contact information and details about the products, services, and support that HPE Software offers.

HPE Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HPE support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HPE Passport user and sign in. Many also require a support contract. To register for an HPE Passport ID, go to: https://softwaresupport.hpe.com and click **Register**.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions & Integrations and Best Practices

Visit **HPE Software Solutions Now** at https://softwaresupport.hpe.com/group/softwaresupport/search-result/-/facetsearch/document/KM01702710 to explore how the products in the HPE Software catalog work together, exchange information, and solve business needs.

Visit the **Cross Portfolio Best Practices Library** at https://hpln.hpe.com/group/best-practices-hpsw to access a wide variety of best practice documents and materials.

Contents

Welcome to the UFT Tutorial

The UFT Tutorial is a self-paced guide that teaches you the basics of testing your application with UFT. It will familiarize you with the process of creating and running automated GUI and API tests and analyzing the run results.

After completing the tutorial, you can apply the skills you have learned to testing your own application.

> **Note:** To learn more about creating and running GUI tests of your Web application, see the UFT Tutorial for GUI Testing of Web applications, available from the `<UFT installation folder>\help` folder.

Tutorial Audience and Scope

This tutorial is intended for users who are new to UFT. No prior knowledge of UFT, QuickTest, or Service Test is required. A general understanding of testing concepts and functional testing processes may be helpful, but is not mandatory. UFT enables you to create GUI tests, API tests, business process tests, and composite tests containing GUI and API tests or calls to tests.

> **Note:** This tutorial refers to file system paths that are relevant for Windows 7 operating systems. The paths in other operating systems may be slightly different.

Using UFT with BPT

In addition to tests, UFT enables you to create keyword-driven, scripted, or API business components for use in business process tests, if you are connected to an ALM server that supports BPT. The procedures described in this tutorial are designed for creating GUI and API tests, but you can also apply the majority of these procedures to creating keyword-driven components, scripted components, or API components. For more details on business components and BPT, see the *HP Unified Functional Testing User Guide* and the *HP Business Process Testing User Guide*.

> **Note:** Unless otherwise specified, references to **Application Lifecycle Management** or **ALM** apply to all currently supported versions of ALM and Quality Center. Note that some features and options may not be supported in the specific edition of ALM or Quality Center that you are using.

Part 1: Introducing Unified Functional Testing

In this lesson, you will get an overview of automated testing and UFT. You will also get started on preparing tests.

> **Note:** It is recommended to work with an soft copy of this tutorial because there are sections in which you will be asked to copy and paste test steps into UFT. However, keep in mind that in selected parts of this tutorial, UFT will be recording your operations or learning objects that you point to with the mouse. During such sessions, you should refrain from switching focus to the PDF or Help window.

This section includes the following:

Benefits of Automated Testing

If you have ever tested applications or Web sites manually, you are aware of the drawbacks of manual testing of these applications.

Manual testing is time-consuming and tedious, and requires a considerable investment in human resources. Worst of all, time constraints often make it impossible to manually test every feature thoroughly before an application is released. This leaves you wondering if serious bugs have gone undetected.

Automated testing of the GUI and non-GUI (service) layers of your application by UFT addresses the problems with manual testing by speeding up the testing process. You can create tests that check all aspects of your application or Web site, and then run these tests each time your application or Web site changes. As a result, you create the tests once, and run them during each subsequent application change, without the need to update the test for each application update.

As UFT runs these tests, it emulates the human user actions by moving the cursor in an application or Web site, clicking objects in the application's GUI, entering keyboard input, or running the application's API processes. However, unlike manual testing, UFT does this faster than any human user.

The benefits of automated testing are numerous:

Benefits of Automated Testing	
Fast	Automated tests are significantly faster than manual tests performed by human users.
Reliable	Automated tests perform precisely the same operations each time they are run, eliminating human error inherent in manual tests.
Repeatable	You can test how your application or Web site reacts after repeated performance of the same operations.
Programmable	You can program sophisticated tests that test a wide variety of complicated but important scenarios and find problems or defects not easily identifiable by regular manual testing.
Comprehensive	You can build a set of tests that cover all the features in your application or Web site.
Reusable	You can reuse automated tests on different versions of an application or Web site, even if the user interface or internal application APIs change.

UFT's Testing Process

When creating a UFT test, there are a number of steps:

1. **Analyzing your application**

   The first step in planning your test is to analyze your application to determine your testing needs:

What are your application's development environments?	To work with your application's user interface objects, you will need to load the appropriate UFT GUI Testing add-ins. For example, your application may be built in a Web, .NET, or Java environment. Therefore, you would need the Web, .NET, or Java Add-ins in UFT to work with these environments.
What business processes and functionality do you want to test?	To do this, you think about the activities a user would perform in your application and the internal actions your application needs to perform to do these business processes. You create GUI test steps to mimic the user's actions in the user interface. You create API test steps to perform the processes your application runs in the background.
Does your application use standard application activities or custom-designed services?	Depending on what functions your application's API runs, you use the out-of-the-box activities provided with a UFT API test or import/create custom activities.
How can you break your test into small testable units?	You should break the processes and functionality you want to test into smaller tasks, so that you can create UFT actions in your GUI tests. These smaller and more modular actions make your tests easier to read and follow, and help ease maintenance in the long run.

   Even at this stage, you can begin creating test skeletons and adding actions to GUI tests.

2. **Preparing the testing infrastructure**

Based on your testing needs, you must determine what resources are required and create these resources accordingly.

- For a GUI test, these resources include **shared object repositories**, which contain test objects that represent objects in your application, **function libraries**, which contain custom functions to use in a test, **recovery scenarios** that instruct UFT how to respond when the application has problems, **environmental variable files** which contain definitions for common environment variables, or **external data tables** to use to parameterize test steps.
- For an API test, these resources include **WSDL** or **WADL** files describing the application service's methods, **REST Services** that you create to serve as a prototype or your application's REST process, **external data sources**, **virtualization projects** used with service calls, **.NET assembilies** referenced by a test step, or **Java classes** used in a test step. These resources must be imported or created in UFT.

You also need to configure UFT settings so that UFT will perform any additional tasks you may need, such as displaying a results report each time you run a test, enabling or disabling debugging for the test run, and the like.

3. **Building your tests and adding steps to each test**

   After the testing infrastructure and resources are ready, you can begin building tests:

   - For GUI tests, you can create one or more empty tests and add actions to them to create the testing skeletons. You then associate your object repositories with the relevant actions, so that you can insert the steps, either by the keyword-drive methodology or by creating scripts.
   - For API tests, you can create one or more empty tests, add test steps to these tests by dragging activities to the test canvas, and define the input, output, and checkpoint properties for these steps. You can also group steps that run multiple times together in an action that can be run as an individual test step.
   - You can also add all your tests to a single solution. A solution enables you to store, manage, and edit any related tests together, without having to close one test before opening another.
   - You may also want to configure test preferences and settings (for GUI tests) or test-specific properties (for API tests).

4. **Enhancing your tests**

You can enhance your tests in a number of ways:

For GUI tests...	• Insert **checkpoints** as test steps to check whether your application is functioning correctly. For example, these checkpoints can check for the specific value of a page, individual test object, or text string. • You check how your application performs the same operations with different values by parameterizing test step values with mutiple sets of data. You do this by replacing the test step's fixed values with **parameters**. • You can add **programming and conditional or loop statements** and other programming logic to your test using VBscript.
For API tests...	• You can validate test step and individual properties of test steps by selecting **checkpoint properties** and entering expected values for the step properties. • You can check how your application performs the same processes with different values by parameterizing test step properties with multiple sets of data. You do this by replacing fixed values with **parameters**. • You can add functionality to your test steps with **custom code activities**, **event handlers**, or **custom activities** created using UFT's Activity Wizard.

5. **Debugging, running, and analyzing your test**

   You can debug your test using UFT's debugging functionality to ensure that it operates smoothly and without interruption. After the test is working correctly, you run it to check the behavior of your application. While running, UFT perfoms each step on the user interface of your application (while running a GUI test) or runs the application's API processes (while running an API test).

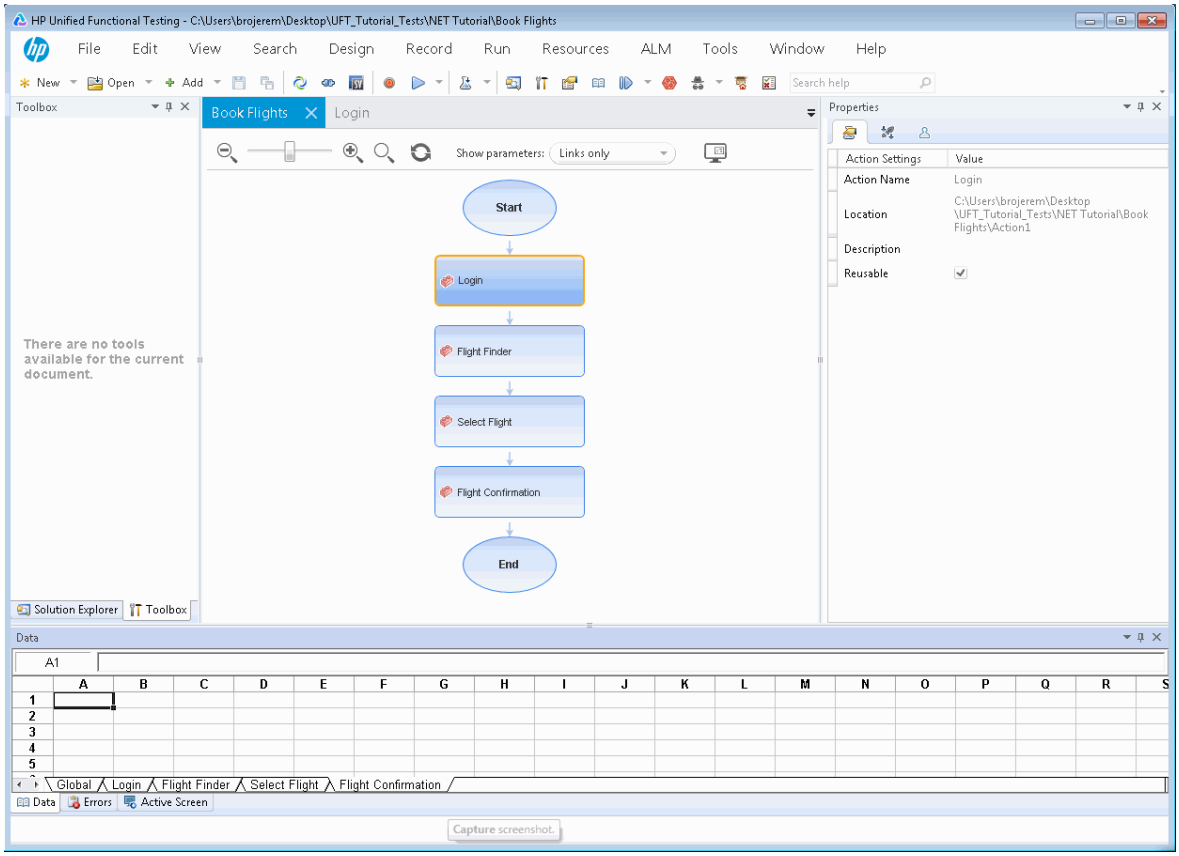
6. **Reporting defects**

   If you have access to an ALM server, you can report defects you discover to your ALM project. If not, you can manually report defects to your own defect database.

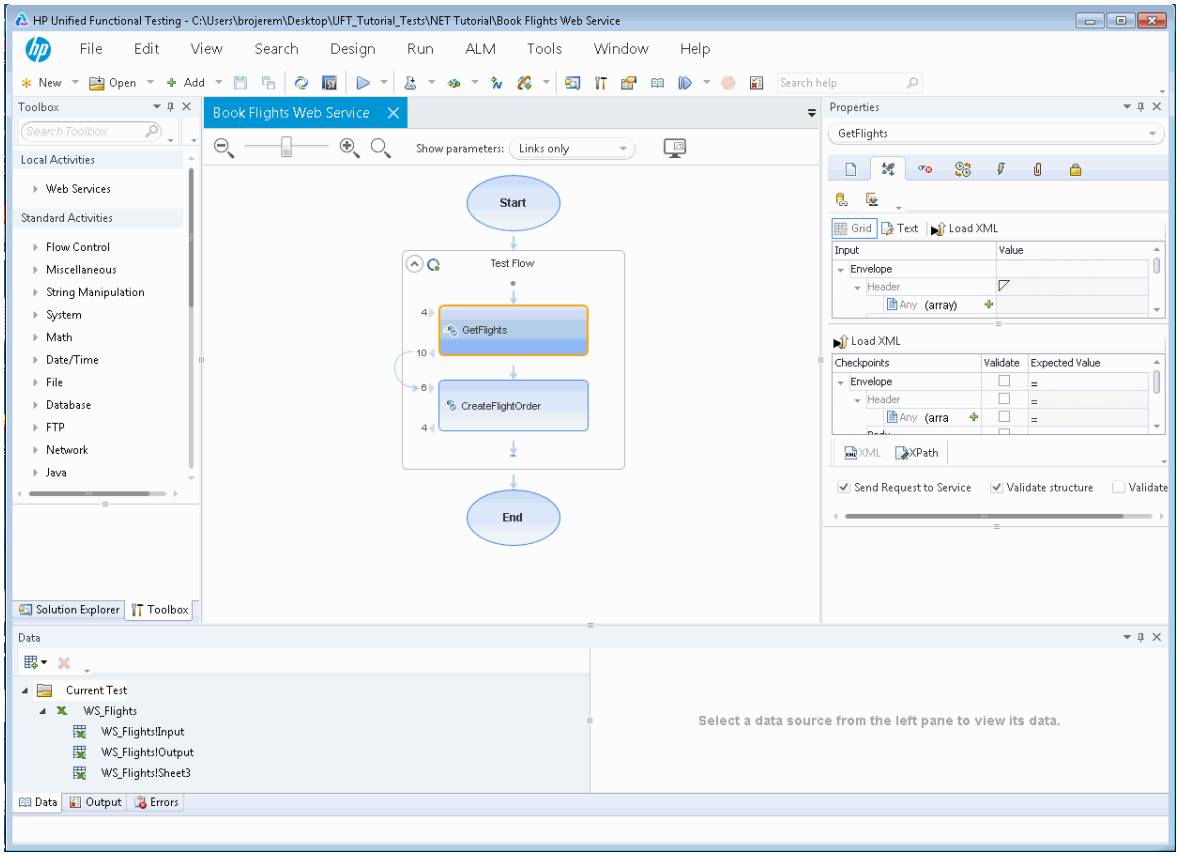
UFT Main Window

Before you begin working with UFT and creating your tests, you should familiarize yourself with the main UFT window.

The image below shows the UFT window after you create a GUI test, with the test flow shown in the canvas, and with the toolbar, Solution Explorer, Data Pane, and Properties pane displayed.



The image below shows the UFT window after you create an API test, with the test flow shown in the canvas, and with the toolbar, Toolbox pane, Data Pane, and Properties panes displayed.



The main window displays a number of elements:

Testing Documents

UFT displays open documents in the document pane (center of the UFT window). You can use the document tabs located just below the toolbar to navigate to open documents and bring them into focus.

The document pane can display the following types of files:

Tests	Both GUI and API tests are displayed on their own tabs in the canvas. A GUI test display shows the test flow, with separate items for each action contained in the test. An API test displays the test flow of the various steps included in your tests. BPT tests are displayed in a grid, listing the individual components, groups, or flows contained in the test.

GUI Test Actions	Each GUI test contains individual actions or calls to other actions. You can view each action in one of the following views: • **Keyword View:** Each step (and the object hierarchy) is displayed in a modular, icon-based table. • **Editor:** Each step is displayed as a VBScript line, displayed in a text/code editor. In object-based steps, the VBScript line defines the object hierarchy.
Business Components	Business components enable you to design a single, modular test "unit" for each business process in your application. These components can then be added to a business process test to run together as an application scenario. You can display one of the following types of business components: • **Keyword GUI components:** These components are displayed in the Keyword View only. • **Scripted GUI components:** These components can be displayed either in the Keyword View or Editor. (They open by default in the Editor) • **API components:** These components are displayed with the test canvas (in the same manner as an API test).
Function Library	Function libraries enable you to create, edit, and modify functions to use in your tests. These functions can be written in a single function library, which can then be used in multiple tests by associating the function library with each test. The function library is displayed in the Editor.
Application Areas	Each GUI business component also contains an application area. The application area serves as the container for the component's object repositories, function libraries, and configuration settings. Each application area can also be associated with multiple business components. The application area user interface is displayed as a series of sidebar tabs.

User code files	In an API test, you can add special event handler code or custom code files. This code enables you to supplement and extend the out-of-the-box functionality of your API test steps. The event handler code is contained in the **TestUserCode.cs** file already included with a default API test. These files are displayed in the Editor.
Start Page	This page welcomes you to UFT and provides links to recent files, description of new features, product forums, and other support links. You can use the shortcut buttons to create documents or open existing ones.
Internal Browser Pages	You can also view internet pages for forums and other product-related materials, such as those accessible from the Start Page or **Help** menu.

Toolbars and Menus

In addition to the document pane, the UFT window contains the following elements:

- **Title bar.** Displays the path of the current test or solution.
- **Menu bar.**
- **UFT toolbar.**

Panes

The UFT window contains a number of panes designed to assist the creation and design of your testing documents.

Some of these panes and toolbar options are described in detail in subsequent lessons. For details on the other panes and toolbar options, see the *HP Unified Functional Testing User Guide*.

Name	Toolbar Button	Description	Default Location

Solution Explorer		Displays all the tests, components, and application areas currently open or included in your solution, as well as all the resources associated with your current tests and components. Using the Solution Explorer, you can manage these resources.	A tab on the left side of the UFT window.
Toolbox		• For GUI tests and components: Displays all the keywords available to your test, and enables you to drag and drop objects, or calls to functions, from the Toolbox pane to your test. • For API tests and components: Displays all the activities available to use in your test, and enables you to drag and drop these activities on the canvas.	A tab on the left side of the UFT window.
Document pane	N/A	Displays all open documents. Each document has a tab that you can click to bring the document into focus.	An unlabeled pane in the center of the UFT window. Each document tab is labeled with the document name. To display: Open a testing document.

Properties		For GUI tests and components: Displays all properties for the currently selected test, action, component, or application area For API tests and components: Displays all properties for the selected test step/test flow or the selected data source (in the Data pane).	A pane on the right side of the UFT window. To display: • Select **View > Properties** • Click the **Properties** button in the toolbar. • Double-click an API test step in the canvas. • Right- click an API test step in the canvas and select **Properties**.
Data		Assists you in parameterizing your test.	A tab at the bottom of the UFT window.
Output	N/A	Displays information during the run session.	A tab at the bottom of the UFT window. To display, select **View > Output**.
Errors	N/A	Displays a list of problems with your tests or components: missing references from a test (such as missing object repositories or recovery scenarios from a GUI test, or missing references to external files or missing property values for an API test).	A tab at the bottom of the UFT window. To display, select **View > Errors**.

Active Screen	N/A	Provides a snapshot of your application as it appeared when you performed a certain step during a recording session. This pane is not used for API tests.	A tab at the bottom of the UFT window. To display, select **View > Active Screen**.
Debug panes		Assists you in debugging your tests. There are multiple debug panes: • **Breakpoints** • **Call Stack** • **Local Variables** • **Console** • **Watch** • **Threads** (for API tests only) • **Loaded Modules** (for API tests only)	Tabs at the bottom of the UFT window.
Tasks	N/A	Displays and enables you to manage the tasks defined for the current test. This pane also displays the TODO comment steps of the test's actions, function libraries, or user code files.	A tab at the bottom of the UFT window. To display, select **View > Tasks**.
Search Results	N/A	Displays all occurences of the search criteria you define in the Find dialog box or using other Search menu items.	A tab at the bottom of the UFT window. To display: • Select **View > Serch Results**. • Perform a search.

Bookmarks	N/A	Displays the list and location of bookmarks contained in your testing documents.	A tab at the bottom of the UFT window. To display, select **View > Bookmarks**.
Run Step Results	N/A	Displays the run results of a test run for an individual API test step. This pane is not used for GUI tests.	A tab at the bottom of the UFT window. To display: • Select **View > Run Results**. • Run a step by right-clicking an API test step and selecting **Run Step**.

Part 2: Analyze your application and creating tests

"Introducing Unified Functional Testing" on page 8 gave you an overview of automated testing and UFT.

In this lesson, you will analyze an application to see what needs to be tested.

This section includes the following:

Get to know the application

Before you begin creating tests of your applications, you should determine exactly what you want to test. To do this, analyze your application in terms of its application processes - the distinct activities that the application performs in order to complete a specific task.

For the purposes of this tutorial, you are testing a flight booking application. This application emulates a flight information and reservation service.

The application consists of two separate components:

- **The Book Flights layer.** The user interface for the application.
- **The Flights API layer.** This application provides the service (API) side of the flight booking application.

> **Note:** For specific details on the service's methods and operations, click the **Open Help Page** button in the Flights API window.

Using the Book Flights layer, you will create a GUI test of the application. Using the Flights API layer, you will create an API test of the application.

As you plan a test of the flight booking application, consider the following:

What business processes need to be tested?	Consider the following: • What processes is your application supposed to perform? • Based on the processes you decide, what actions does a user take to fulfill these processes?
How is the application organized?	• Are there separate sections/pages/modules of the application for each user activity? • What are the activities a user can perform? • Where are these sections/pages/modules in the application? • What are the expected results of these user activities? • What behind-the-scenes processes support these user activities?

What user interface elements need to be tested in each of the application's sections/pages/modules?	• What user interface objects need to be tested in each area? • What user actions does the test need to simulate?
What activities might be used in multiple scenarios?	What specific user actions or application processes are done repeatedly? For example, logging onto an application could be something a user performs repeatedly, or connecting to the user credential database is an activity the application performs repeatedly.
What development environments need to be supported for testing purposes?	UFT provides add-ins to support numerous testing environments. In order to load the proper ones for your application, you must consider the technologies used in the development of the application in order to ensure UFT supports your technologies. In addition, you must also load these add-ins when opening UFT and creating tests. This ensures that UFT will properly recognize the objects in your application when creating and running tests.

Explore the flight reservation application

Before you begin creating tests, you need to explore the sample application and see what user actions it has and what processes support the application flow.

As you navigate and use the application, consider the questions posed in the previous lesson. Use these questions to guide your thinking about how you could create a test or tests from the application.

1. **Open the flight reservation application.**

   Open both the Book Flights (GUI) layer and the Flights API (service) layer:

   - The Book Flights layer is available at **Start > All Programs > HP Software > HP Unified Functional Testing > Sample Applications > Flight GUI.**
   - The Flights API layer is available at **Start > All Programs > HP Software > HP Unified Functional Testing > Sample Applications > Flight API**.

   For details on accessing UFT and UFT tools and files in Windows 8.X or higher and Windows Server 2012, see "Accessing UFT in Windows 8.X or Higher Operating Systems" on page 246.

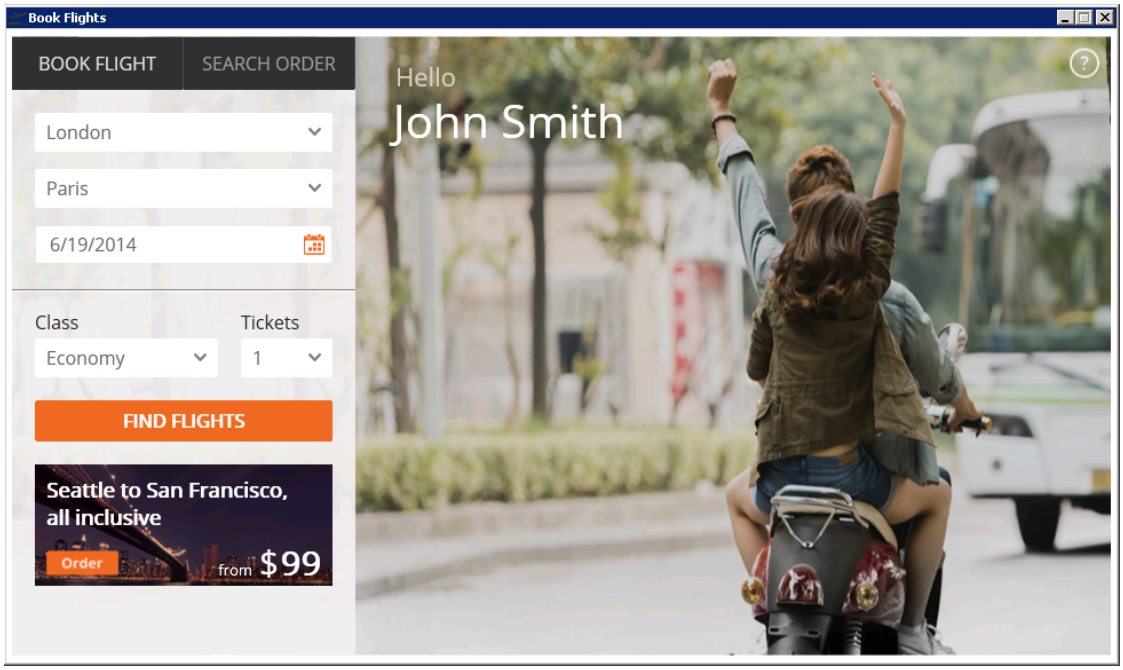
2. **Log in to the Book Flights application.**

In the Book Flights application start page, enter `John` for the user name and `hp` for the password.

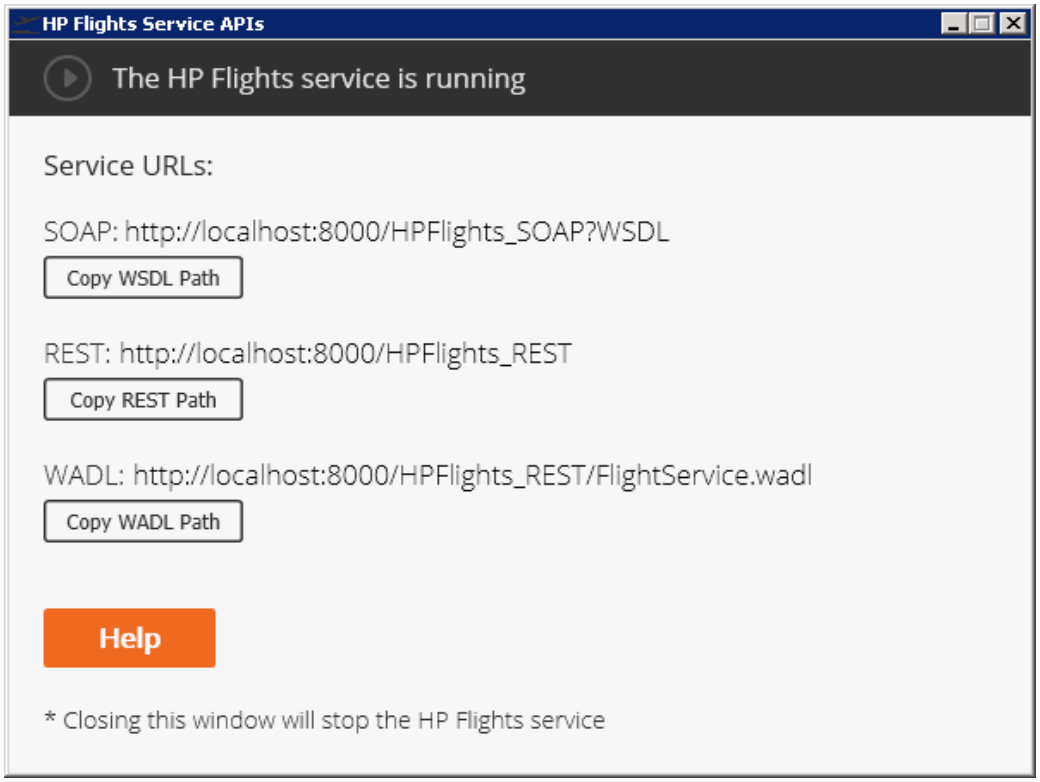
> **Note:** The Flights API layer requires no login information.

After logging in successfully, the application layer display should look like this:

Book Flights (GUI layer)



Flights API layer



3. **Explore the application layers.**

For Book Flights layer...	Enter the requested information or selections on each page to follow the reservation process. As you navigate through the application, consider what user actions you might want to test, and which objects you would need to create to set up your test.
For Flights API layer...	Click the **Help** button to see a description of the methods included in the application. As you explore the list of methods included in the application, note the property details provided for each method. You can use this data later when designing a test to provide property values for these methods.

4. **Exit your application browsing session.**

- **For the Book Flights layer:** After the flight reservation order details are displayed in the Order Details window, click **New Search** to return the application to the start.
- **For Flights API layer:** Minimize the Flights API window. Do not close this window, as this will stop the service.

You are now ready to use these applications to create tests for each layer of the application. Continue with "Create a solution" below to create a solution for the tests you will use in the course of this tutorial.

Create a solution

In UFT, you create a **solution** to serve as a container for your tests. In the solution, you add any tests, business components, application areas, or function libraries to the solution. The same document can be added to multiple solutions.

Solutions enable you to group tests together in a convenient way. For example, you can create solutions containing all the tests of a particular application, or create solutions containing only the GUI or API tests of your application's parts.

In this exercise, you will create a new solution to hold the tests you will create in the course of this tutorial.

1. **Start UFT.**

   Do one of the following:

If UFT is not currently open	a. Open UFT. ◦ Double-click the UFT icon on your desktop or on the **Start** Screen (in Windows 8.x or higher or Windows Server 2012). ◦ In Windows 7 or Windows Server 2008 R2, select **Start > All Programs > HP Software > HP Unified Functional Testing**. b. In the Add-in Manager, confirm that **only** the **WPF** Add-in is selected. Clear all other add-ins. c. Click **OK** to close the Add-in Manager and open UFT. The UFT splash screen is displayed while UFT loads your selected add-ins.

If UFT is currently open	a. Select **Help > About HP Unified Functional Testing** to check which add-ins are loaded. The loaded add-ins are displayed with a checkbox next to their name in the About dialog box. b. If the, WPF Add-in is not loaded, you must exit and restart UFT. When the Add-in Manager opens again, select WPF. If the Add-in Manager does not open when starting UFT, after UFT opens select **Tools > Options**. Then, in the **Startup Options** pane (**Tools > Options > General** tab > **Startup Options** node), select the **Display Add-in Manager on startup** option.

2. **Create a new solution.**

   a. In the toolbar, click the **New** down arrow 

and select **New Solution**.

   b. Enter the details for the solution:
      - **Name:** Flight Reservation Application
      - **Location:** By default, all solutions and tests are saved at **C:\%HOMEPATH%\My Documents\Unified Functional Testing**. For the purposes of this tutorial, you do not need to modify this path.

   c. Click **Create**.

      In the Solution Explorer pane, you can now see that the solution name is displayed at the top of the pane. Tests you add to this solution will be displayed as sub-nodes of this solution.

      Note: If the Solution Explorer is hidden, click the Solution Explorer button  in the toolboar or select **View > Solution Explorer** to display it.

You can now begin creating GUI tests, as described in "Create and running automated GUI tests" on page 28 or API tests, as described in "Create and run automated API tests" on page 150.

Part 3: Create and running automated GUI tests

After analyzing your application and planning your testing goals, you create the tests of the application. A major part of this effort is creating tests of the user interface (GUI) of your application. Doing so ensures that the controls and objects in your application work as designed.

Creating a GUI test involves a number of separate processes:

- Creating object repositories containing test objects for the objects in your application
- Creating supplementary functions to use in your tests inside function libraries
- Adding steps to the test representing user actions in the application
- Creating checkpoints to validate specific objects in the application
- Parameterizing the test object values to see how the application reacts to different input values
- Running the test and analyzing the results of the test run

The following lessons will introduce and teach these processes in detail.

This section includes the following:

Lesson 1: Create a GUI test and actions

Before creating steps to test your application's GUI, you must first create a test and create the actions that provide the test structure.

Each UFT GUI test consists of calls to actions. Actions are units (within the test) that divide your test into logical sections. By dividing your tests into multiple actions, you can design more modular, understandable, and efficient tests.

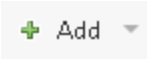
Your test can contain a number of different types of actions:

Internal and External Actions	• An **internal action** is an action that is stored in the local test (also know as a source test). • An **external action** is a referenced call to an action that is stored in a different test. An external action called by a test is shown as a separate node under the test node in the Solution Explorer. For example, if you have an action that you want to use in multiple tests, you would store the action as an internal action in one test and insert calls to that action from other tests. In the other tests which call the action, the action is available as an external action.
Reusable actions	When you insert a call to a new action, it is **reusable** by default, enabling you to call the action from any test. When you use reusable actions, you only need to update the existing action stored with the original test. When you modify that original action, all tests containing calls to the action are updated. These reusable actions are read-only in the tests that call the reusable action. Reusable actions can be useful if: • You have a process that you may need to include in several tests, such as logging into your application. • You may have a process that you need to insert several times in the same test, such as entering user credential dialog boxes that open whenever a user tries to access a secure part of your application. If you want to prevent an action from being used in other tests, you can make the action **non-reusable**.
Copied actions	You can also insert a copy of an action in a test if you want to modify the action steps. When you copy the action, it becomes an **internal** action of the test into which it is copied. These copies are not linked to the source test, so any changes in the original action are not updated in the copy.

Tip: If you plan to use the same or similar actions in many tests, you might consider creating a repository test to store your reusable actions. Then, you can insert calls to these existing actions from other tests.

You add actions to a test in the following ways:

- **Calls to new actions.** This inserts a new, empty action in your test. The action is an internal action of the test containing it.
- **Calls to a copy of an action.** This inserts a copy of an existing action in your test.
- **Calls to an existing action.** This inserts a call to an existing action (an action external to the current test).

1. **Create a new test and add it to the solution.**
   a. In the toolbar, click the **Add** button down arrow  and select **Add New Test**.
   b. In the Add New Test to Solution dialog box, select **GUI Test**.
   c. Enter the following details:
      - **Name: Book Flights**
      - **Location:** By default, UFT saves documents at **C:\%HOMEPATH%\My Documents\Unified Functional Testing**. For this lesson, you do not need to modify this path.
   d. Click **Add**.

      A blank test opens in the canvas, with one tab for the test flow (name **Book Flights**), and another tab for the action (name **Action 1**).

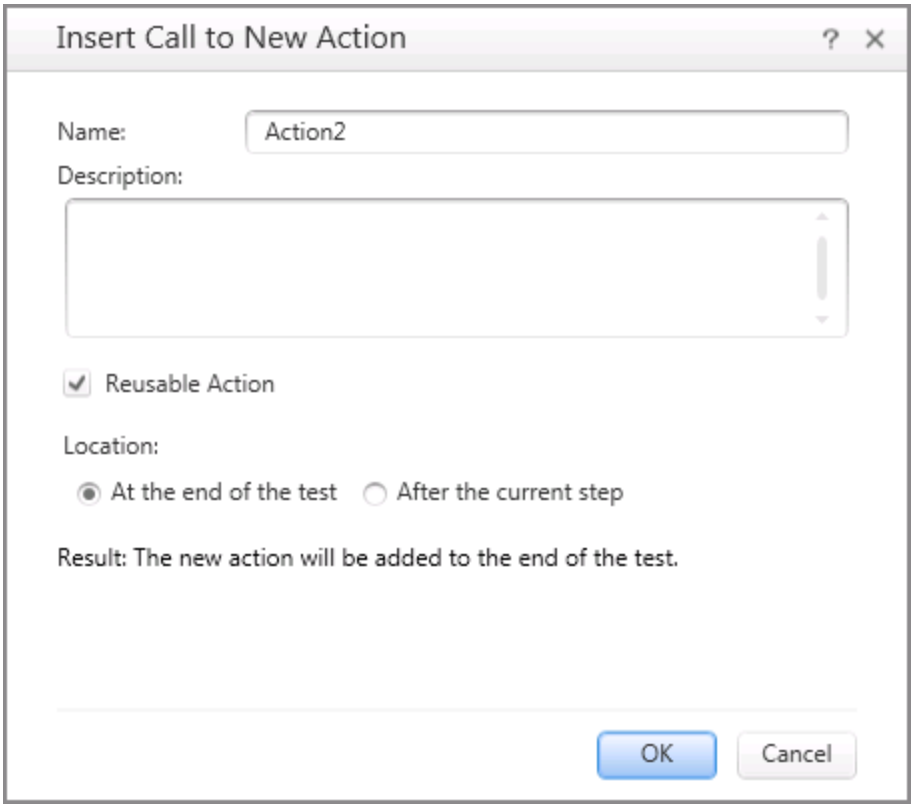
      This test is also displayed as a subnode of the Flight Reservation Application solution node in the Solution Explorer pane.

2. **Rename Action 1 so that it has a logical name.**
   a. In the canvas, right-click the **Action1** box and select **Action Properties**.
   b. In the General tab of the Action Properties dialog box, rename **Action1** to **Login** and click **OK**.
   c. In the warning that opens, click **Yes**.

      The action block in the canvas should now be displayed with the name **Login**.

3. **Create a new action named Flight Finder.**
   a. Right-click anywhere in the canvas and select **Call to New Action**.

The Insert Call to New Action dialog box opens:



b. In the **Name** box, enter **Flight Finder**.
c. Ensure that the **Reusable action** and **At the end of the test** options are selected.
d. Click **OK**.
   An additional block is added in the canvas with the name **Flight Finder**.

4. **Add an additional action to your test.**
   a. In the toolbar, click the **Insert Call to New Action** button .
   b. In the Insert Call to New Action dialog box, in the **Name** box, enter **Select Flight**.
   c. Ensure that the **Reusable action** and **At the end of the test** options are selected.
   d. Click **OK** to add the action to the test.
      Another block is added in the canvas with the name **Select Flight**.

5. **Create a final action.**
   Using either of the methods used above, add another action to your test named **Flight Confirmation**.
   Your test now contains all the actions necessary to test your application.

6. **Save your test.**

In the toolbar, click the **Save** button  .

You may have noticed that before you clicked **Save**, an asterisk (*) was displayed in the Book Flights tab in the document pane and the Book Flights node in the Solution Explorer. These asterisks are displayed to indicate that a document has unsaved content. When you save a test, all changes in action tabs are also saved.

Now that you have your test structure, you can begin creating object repositories for the test. Continue to "Lesson 2: Creating Object Repositories" on the next page to continue.

Lesson 2: Creating Object Repositories

The basis of a GUI test is the collection of test objects used to test your application's user interface. These test objects are learned by UFT and then stored in object repositories that are associated with your test.

Now that you have created a test and its test structure (by creating actions), you need to create the test objects to use in your tests. This lesson introduces the basic concepts of test objects, run-time objects, and object repositories which are used in your tests.

This lesson includes the following:

UFT Test object recognition

When creating and running GUI tests, UFT uses **test objects** to recognize objects in your application and then create test steps based on your application's objects. These test objects are based on UFT's **test object model**.

The test object model is a large set of object types or classes that UFT uses to represent the objects in your application. Each test object class has a list of identification properties that UFT can learn about the object, a sub-set of these properties that can uniquely identify objects of that class, and a set of relevant operations that UFT can perform on the object.

When designing and running a test, there are two different types of objects:

Test objects	Test objects are stored representations that UFT creates to represent the actual objects in your application. UFT creates test objects by learning a select set of the properties and values of the objects in your application. UFT then stores the information on the object that will help it identify and check the object during the run session, and uses the data to recognize the application object during the run session. Each test object is part of a larger test object hierarchy. For example a **Link** object can be part of a **Page** object inside a (Web) **Browser** object.  Top-level objects, such as **Browser** objects, are known as container objects, as they can hold lower-level objects, such as **Page** or **Frame** objects.
Run-time objects	Run-time objects are the actual objects in your application on which UFT performs the actions (methods) during the run session. UFT learns the properties of run-time objects and translates them into test objects.

When UFT learns an object in your application, it adds a corresponding test object to an **object repository**. This object repository serves as a storehouse for the test objects. When UFT runs a test, it looks in the test's object repositories for the objects contained in the test steps.

When you add an object to an object repository, UFT:

- Identifies the UFT test object class that represents the learned object in your application and creates an appropriate test object.
- Reads the current value of the object's properties in your application and stores

the list of identification properties and values with the test object.

- Chooses a unique name for the test object.

There are two different types of object repositories:

Shared object repositories	A shared object repositories is an object repository that exists independently of an individual test. The test objects in a shared object repository can be used in multiple tests/actions. This makes this type of object repository the preferred repository type for storing and maintaining your test objects as any updates you make to a test object are then applied to all tests using that shared object repository.
Local object repositories	Local object repositories contain the test objects used in the context of a specific action. These type of object repositories cannot be used with other actions. By default all actions have a local object repository.

When you create an object repository, it is recommended to include only the test objects you need for your testing purposes. This keeps the object repository relatively small and helps to simplify maintenance and object selection. Also, make sure that you provide logical names so that others can easily select the correct objects when creating or modifying tests.

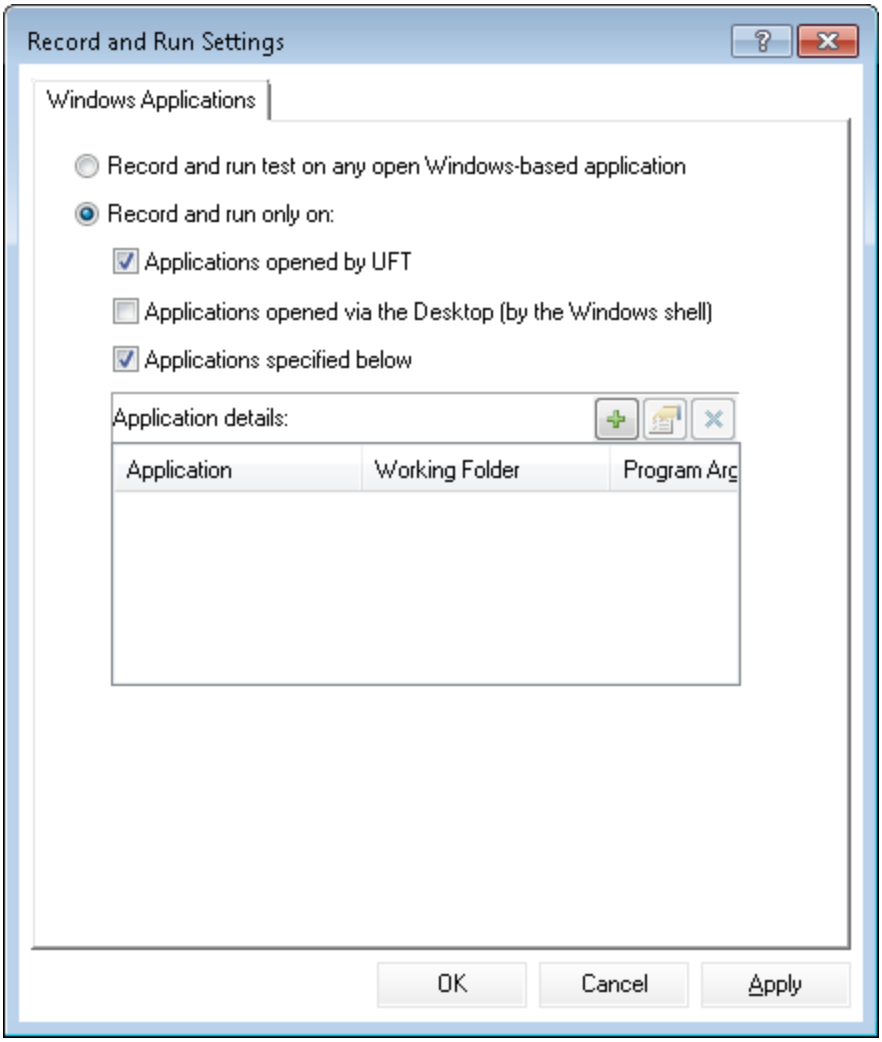
Object repositories can also include checkpoint and output objects. Checkpoint object types are covered in "Lesson 5: Parameterize steps and objects" on page 80.

Exercise 2a: Add objects from the application

In "UFT Test object recognition" on the previous page, you learned about UFT's test object model and how UFT learns and stores objects. In this exercise, you will use UFT's object recognition capabilities to learn objects and create an object repository.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not open, open UFT, as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent tests/components** area, click **Book Flights**.
      The Book Flights test opens, displaying the Book Flights test (and its actions) you created in "Lesson 1: Create a GUI test and actions" on page 29.
2. **Set the learn settings for UFT.**

a. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.
b. In the Record and Run Settings dialog box, select the **Windows Applications** tab.
c. In the Windows Applications tab, select the **Record and run only on** option.
d. Under the **Record and run only on** option, select the checkboxes for the **Applications opened by UFT** and **Applications specified below** options:



e. In the Application details area, click the **Add** button .
f. In the Application details dialog box, enter the paths to the application and the working folder for the application:
   - **Application: <UFT installation folder>\samples\Flights Application\FlightsGUI.exe**
   - **Working folder: <UFT installation folder>\samples\Flights Application**
g. Select the **Launch application** option and click **OK**.

h. In the main Record and Run Settings dialog box, click **OK**. Later, when you record steps on the application or run a test for it, UFT will be able to work with the application.

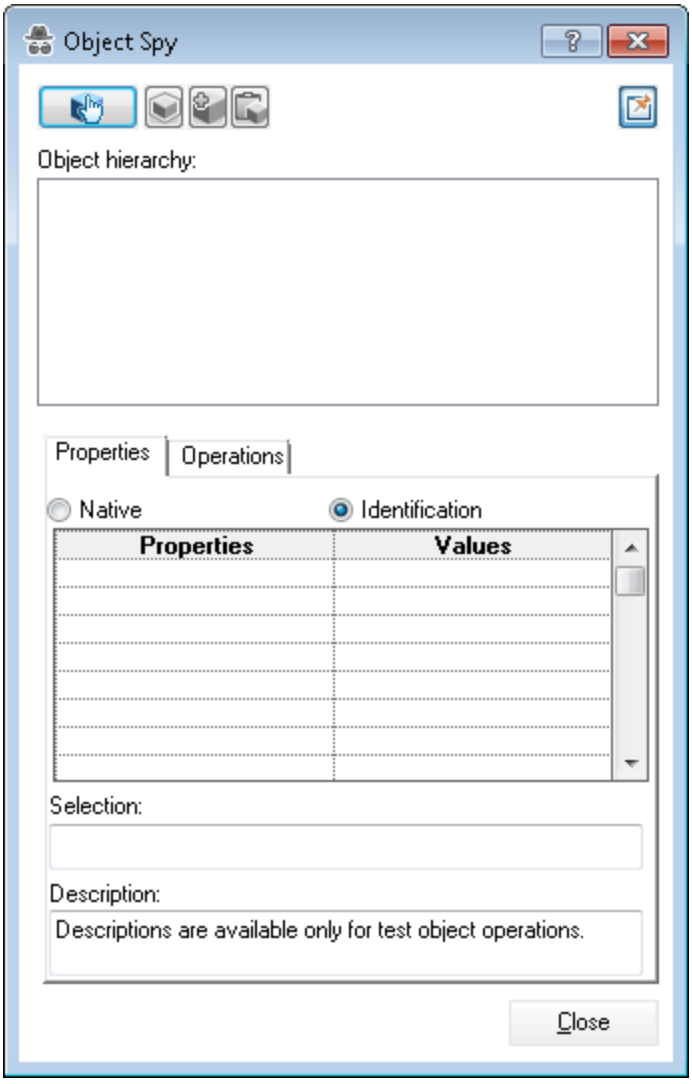
3. **Open the flight reservation application.**

   If it is not already open, open the flight reservation application.

   > **Note:** If the flight reservation application is open, close it and reopen the application to enable UFT to recognize the objects as WPF-based objects.

4. **View the available properties and operations for some of the objects in the flight reservation application.**

   a. In the toolbar, click the **Object Spy** button  . The Object Spy dialog box opens:



b. Drag the Object Spy dialog box to the side of the application. This enables you to see the objects in your application that you want to spy on more clearly.

c. Verify that the **Keep Object Spy on top while spying** toggle button is pressed.

d. Click the pointing hand button .

When you press the pointing hand, UFT is hidden and the Object Spy dialog box is displayed over the flight reservation application.

Tip: If you need to switch back and forth between the flight reservation application, UFT, or any other open window, press **CTRL** to change the pointing hand back to a regular Windows pointer. Hold down the

> **CTRL** button as long as you need the pointer, and then release it when you are ready to use the pointing hand.

e. Hover over the various objects on the page and watch to see what happens in the Object Spy dialog box.

> **Note:** If UFT does not recognize your objects in the correct location, check to see that you are viewing your application or page at 100%, and are not zooming in or out of the current view.
>
> For example, if you view a page at 90% or 120, you may be required to click or select an area to the left or right of the actual object in order to recognize it.

f. Click in the **Username** edit box. The test object hierarchy of that object is displayed. In the hierarchy box, notice that the name of the object is WpfEdit, which is its object class type.

g. In the application **Login** window, click inside the **Username** edit box. This makes the object active in the application.

h. In UFT, in the Object Spy dialog box, click the pointing hand button again  . In the Object hierarchy box, note the Object Spy displays **agentName**.

i. Close the Object Spy dialog box.

5. **Add the necessary objects for your test to the object repository.**

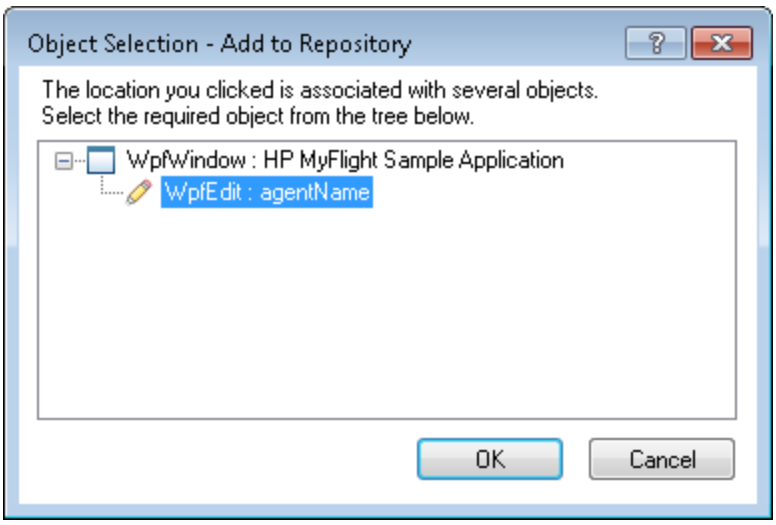
   In this step, you take the "spy" process a step further and instruct UFT to learn only the objects that are needed for your test and add them to the object repository.

   a. In the application **Login** window, click inside the **Username** edit box.

   b. In UFT, select **Resources > Object Repository Manager**. The Object Repository Manager window opens.

   c. In the Object Repository Manager window, select **Object > Add Objects**. Both UFT and the Object Repository Manager are hidden.

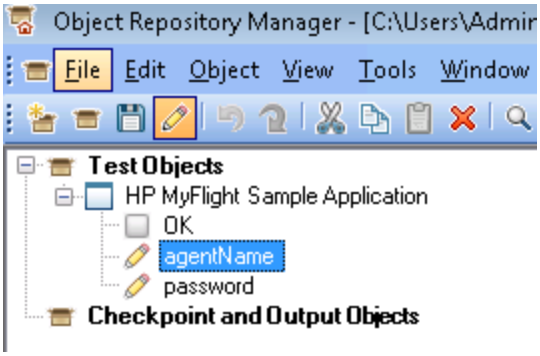
d. In the **Login** page, click again in the **Username** edit box. The Object Selection - Add to Repository dialog box opens:



e. In the Object Selection dialog box, select the **agentName** object and click **OK**. The **agentName** object is added to the Object Repository along with its parent object, the **Login** window object.

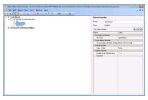
f. Repeat the process above to add the objects for the **Password** edit box and **OK** button.

After you add all the objects to the object repository, your repository should look like this:



6. **See what UFT learned about one of the objects.**

In the **Test Objects** tree, select the **agentName** object, and notice the object properties displayed in the right pane of the object repository. These are the descriptive properties that UFT uses to identify the object during a run session:



7. **Save the object repository.**

a. In the Object Repository Manager window, click **Save**  .
b. Browse to the folder where your solution and tests are saved, in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**. In that folder, create a new folder named **Tutorial_Object Repositories**, and open it.
c. In the Tutorial_Object Repositories folder, save the object repository with the name **Login** and click **OK**.

> **Note:** This step only saves the object repository. It is not connected with a test when you save it, even if you have a test open the UFT main window.

8. **Associate the object repository with the Login action.**

   Associating an object repository with an action enables you to then use any object from that repository in any step in the associated action.

   > **Note:** The same object repository can be associated with multiple tests and actions.

   a. Open the main UFT window.
   b. In the main UFT window, open the Solution Explorer.
   c. In the Solution Explorer, under the **Book Flights** node, expand the **Login** node.
   d. Right-click the **Login** action and select **Associate Repository with Action**. The Open Shared Object Repository dialog box opens.
   e. In the Open Shared Object Repository dialog box, navigate to the folder where you saved the object repository in the previous step and select the **Login.tsr** file. Click **Open**.
   f. In the dialog box that opens, click **Yes**.

      The object repository is now associated with the **Login** action, and is displayed in the Solutoin Explorer as a child element of the action.

9. **Save your test.**

   Select **File > Save** to save your test.

Now that you have taught UFT to recognize your appilcation's objects, created an object repository containing those objects, and associated the object repository with a test, you can create additional object repositories for the other areas of the application. Continue to "Exercise 2b: Create object repositories using Navigate and Learn" on the next page to learn more.

Exercise 2b: Create object repositories using Navigate and Learn

In "Exercise 2a: Add objects from the application" on page 35, you learned how to use UFT to add objects in your application, create a shared object repository for the Login page of the flight reservation application, and associated this object repository with a test.

In this lesson, you will create a shared object repository for each of the remaining pages in the site. You will use the Navigate and Learn process, which enables you to learn all the objects in a page or section of application at once.

> **Tip:** It is recommended to always create a separate shared object repository for each section/page of your application or Web site. This makes it easier to find the correct object when adding or modifying test steps or when performing maintenance tasks.

For the purposes of this exercise, you will instruct UFT to learn all of the objects on each page. At this point, you do not need to associate the object repositories with specific actions.

1. **Log in to the flight reservation application's Flight Finder page.**
   a. If it is not already open, open the flight reservation application.
   b. In the Login page, enter the login credentials:
      - **Username:** john
      - **Password:** hp
   c. Click **OK**. The Book Flight page opens.

2. **Create a new shared object repository.**
   a. If the Object Repository Manager is closed, open it by selecting **Resources > Object Repository Manager** in UFT.
   b. In the Object Repository Manager window, click **New**  . A blank shared object repository opens.

3. **Start the Navigate and Learn process by setting up the Define Object Filter.**
   In addition to adding objects individually as you did in the previous exercise, you can learn and add all the objects in your application in one process using the Navigate and Learn mechanism.

a. In the Object Repository Manager window, select **Object > Navigate and Learn**. Both UFT and the Object Repository Manager are hidden.

b. In the Navigate and Learn toolbar, click the **Define Object Filter** button  . The Define Object Filter dialog box opens.

c. In the Define Object Filter dialog box, select **All object types**, and click **OK**.

4. **Learn all of the objects from the Flight Finder page.**

   In this step, you instruct UFT to learn all of the objects in the Book Flight page that match your filter, and to add them to a shared object repository.

   a. In the flight reservation application's Book Flight page, click the application title bar to bring it into focus as the page you want UFT to learn.

   b. In the Navigate and Learn toolbar, click **Learn**. The application flickers and the Adding Objects message box is displayed as UFT begins adding representations of the objects on the page to a new object repository.

   > **Note:** Adding these objects takes a few seconds. Do not interact with the application while the Navigate and Learn process runs.

   c. Close the Navigate and Learn toolbar. UFT and the Object Repository Manager window are visible again.

5. **Save the shared object repository.**

   a. In the Object Repository Manager window, click **Save**  . The Save Shared Object Repository dialog box opens.

   b. Browse to the **Tutorial_Object Repositories** folder created the previous exercise.

   c. Name this object repository `Flight Finder` and click **OK**.

6. **Create object repositories for the remaining application pages.**

a. Using the process described in the previous step, create new shared object repositories for each of the following pages in the application:
   - **Select Flight** window
   - **Flight Details** window

   > **Note:** Make sure before learning the objects in the Flight Details page that you enter a string in the **Passenger Name** box.
   >
   > This activates the **Order** button and enables UFT to learn it properly. You will need this button in other exercises.

b. Name the object repositories `Select Flight` and `Flight Confirmation`, respectively.

   > **Note:** The Flight Confirmation repository is for the Flight Details window of the application (even though the names differ).

7. **Associate the Flight Finder object repository with the Flight Finder action.**
   a. Switch to the UFT window. If the Solution Explorer is not already open, open it by clicking on the **Solution Explorer** tab in the bottom left corner of the UFT window.
   b. In the Solution Explorer, in the Book Flights node, right-click the **Flight Finder** node and select **Associate Repository with Action**. The Open Shared Object Repository dialog box opens.
   c. Browse to the **Tutorial_ObjectRepositories** folder.
   d. In the Tutorial_ObjectRepositories folder, select the **Flight Finder.tsr** file and click **Open**.
   e. In the dialog box that opens, click **Yes**.

      The object repository is now associated with the **Flight Finder** action, and is displayed in the Solution Explorer as a child of that action.

8. **Associate the remaining object repositories with the relevant actions.**

   Associate the object repositories with the actions as follows:

Action	Object Repository
Select Flight	**Select Flight.tsr**
Flight Confirmation	**Flight Confirmation.tsr**

Later, when you add steps to each action, all of the required test objects will be available for use.

9. **Save your test.**

   In the toolbar, click **Save**  .

   Now that you have created object repositories and associated them with your tests, you can create test steps using these objects. Continue to "Lesson 3: Add steps to a test" on the next page to begin creating test steps.

Lesson 3: Add steps to a test

In "Lesson 1: Create a GUI test and actions", you created a test and actions to provide a structure for the test of the flight reservation application. In "Lesson 2: Creating Object Repositories", you created object repositories with the test objects for the application.

In this lesson, you will learn the final mandatory step for creating tests of your application. You will learn how to add test steps to your GUI tests that enable you to run an accurate test of user actions in the user interface.

This lesson includes the following:

Adding Test Steps in a GUI Test

To create test steps in a GUI test, you have to use the objects in test steps and instruct UFT what actions to perform on the test objects. This enables UFT to replay the actions on your application by translating the test object methods (actions) into actions on the run-time objects in your application.

To assist with this, UFT has a number of different ways to add test steps:

Keyword View	You select your test objects in the step grid, and add the necessary actions (methods) for these test objects. The Keyword View automatically sorts the object hierarchy as needed. After you select the appropriate objects and methods, your test steps are displayed in a grid that shows the object name, object method, any parameters added, and a documentation summary of the step.
Editor	In the Editor, you type in the objects (including the necessary hierarchy for the objects as needed), along with the object method and parameters. If you are experienced with writing code for your application, this can be an easier way to create test steps.
Recording	When you record in your application, UFT translates your actions into test steps, displaying the object name and the action (method) performed on the object. This enables you to perform the test as a user would and in turn have UFT automatically create the test instead of manually editing it inside UFT.
Toolbox pane	When you select a GUI action tab in the document pane, UFT automatically displays the associated objects and functions for that action in the Toolbox pane. Then, you drag these test objects (or functions) into the Keyword View or Editor , and UFT automatically creates a step with the object. (You still need to provide the method for the object however, after dragging it from the Toolbox pane.)
Step Generator	Using the Step Generator dialog box, you can select and provide all the details for the test step in a single dialog box. The Step Generator enables you to choose any test object currently associated with the selected action, the method for that action, and the necessary parameters. After you select this information, UFT inserts a step in the selected place with all the details.

In the exercises that follow, you will create test steps using each of these methods.

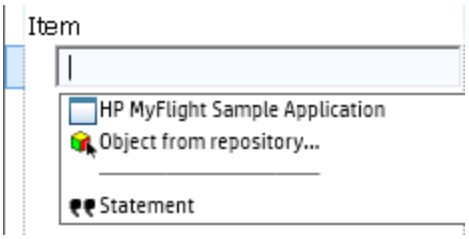
Exercise 3a: Add steps to the Login action in the Keyword View

In this exercise, you will use the Keyword View to insert steps into the Login action.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open it as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the Recent solutions area, click **Flight Reservation Application**.
      The Flight Reservation Application solution opens, displaying the Book Flights test you created in "Lesson 1: Create a GUI test and actions" on page 29.

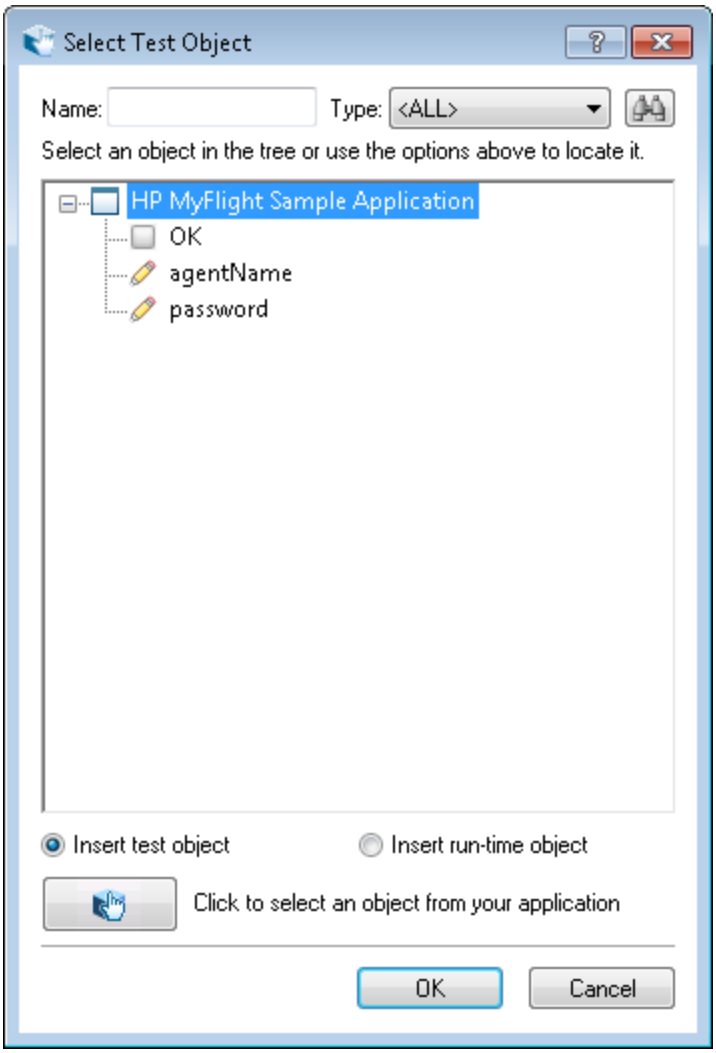
2. **Open the Login action and display the Keyword View.**
   a. If the test flow canvas is not currently open, click the **Book Flights** tab in the document pane to display it.
   b. In the canvas, double-click the **Login** action.
      The **Login** action opens in a separate tab in the document pane.
   c. If the Editor is displayed, in the toolbar, click the **Keyword View** button  to display the Keyword View.

3. **Add the first step to log in to the flight reservation application.**
   a. In the Keyword View, in the **Item** column, click the **NEW STEP** button. An empty edit field in the grid in the Item column.
   b. In the Item column, click in the empty edit field. The Item list dropdown list opens, displaying the top-level parent object in the associated object repository, and instructing you to select an item.



      In this step, the parent object is the **Login** window. *Do not* select this object now because you do not want to perform an operation on the main window. You only need to insert steps on the objects on which you want to perform operations.
   c. Select **Object from repository** to open the Select Test Object dialog box.

d. In the Select Test Object dialog box, expand the test object tree:



e. In the test object tree, select **agentName** and click **OK**.

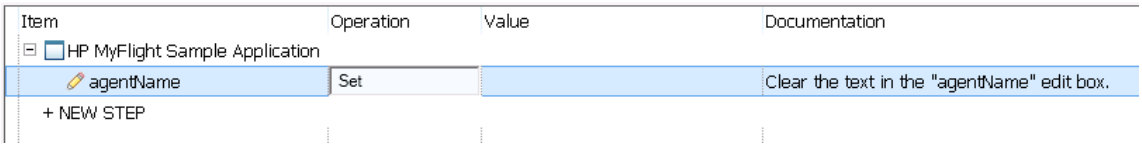
The Select Test Object dialog box closes and one step is added to the action.

Note that three rows are added to the Keyword View. UFT adds a row for each of the parent test objects, even though it does not perform an operation on these objects. The rows are part of the path to the object upon which the step is performed

During a run session, UFT uses the parent objects to identify the actual object on which needs to perform an operation.

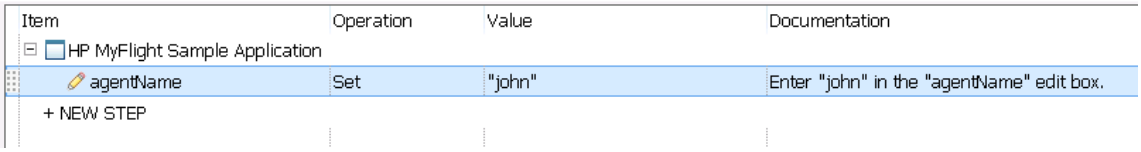
In this step, represented by the last of the three new rows:

- The selected **agentName** WpfEdit object is added to the **Item** cell.
- The default method, **Set** is added to the **Operation** cell.
- Text is added to the **Documentation** cell indicating that this step clears the text in the edit box. This is because the step is missing a required value in the **Value** cell, and needs to be updated with the user name.



f. Click in the **Value** cell and enter `john`. After entering the string, press **ENTER**.

Inserting this value completes the step. When you click another area in the Keyword View, the documentation for this step is updated in the **Documentation** cell:



Tip: Quotes are automatically added around the value you entered in the **Value** column, indicating this is a `String` value. If the method supported an `Index` value, and you entered an `Index` value, no quotes would be added.

g. Select **View > Editor** to display the Editor, which displays the syntax of the step in VBScript:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("agentName").Set "john"
```

Notice that this step is being performed on a **WpfEdit** (edit box) test object, named **agentName**, and that:

- The test object hierarchy for the WpfEdit (edit box) test object is displayed prior to the test object. In this step the hiearchy includes a WpfWindow object.
- The method to be performed on the object is displayed immediately after the test object. In this step, the method is **Set**.
- The text to enter in the **agentName** edit box is displayed immediately after the **Set** method. The text is displayed in quotes, which indicate that this value is a string. In this step, the text to enter is `john`.
- Full stops (periods) separate each part of the step.

h. Click the **Keyword View** button  to return to the Keyword View.

4. **Add the next step.**

a. In the **Item** column, directly below the **agentName** row, click the **NEW STEP** button. An additional row with an edit field is added.
b. Click in the blank edit field. The Item list opens, listing the sibling objects of the previous step's test object.
c. Select **password** from the Item list. This time, only one new row is added because the object shares the same parent objects as the previous step.

   In this step:

   - The **password** WpfEdit test object is added to the **Item** cell.
   - The default method, **Set**, is added to the **Operation** cell. You need to change this method because the password needs to be encoded.
   - Text is added to the **Documentation** cell indicating that this step clears the text in the edit box. This is because the step is still missing a required value in the **Value** cell, and needs to be updated with the password.

d. Click in the **Operation** cell to display the down arrow, and then click the down arrow to display the list of available methods for the selected test object.
e. In the list of methods, select **SetSecure**. This method enables the use of encrypted text.

5. **Generate an encoded password using the HP Password Encoder application.**
   a. Select **Start > All Programs> HP Software > HP > HP Unified Functional Testing > Tools > Password Encoder** or **<UFT installation folder>\bin\CryptonApp.exe**.
   b. In the Password Encoder, in the **Password** box, enter `hp`.
   c. Click **Generate**. The Password Encoder encrypts the password and displays it in the **Encoded String** box.
   d. Click **Copy**.
   e. In UFT, in the Login action, paste the encoded value in the **Value** cell for the password step and press the **ENTER** key.

      In the **Documentation** column of this step, the updated documentation is displayed.
   f. Close the Password Encoder dialog box.

      If you ran your action at this point, UFT would automatically open the flight reservation application and inser the values you specified in the **Username** and **Password** boxes.

6. **Insert the last step in the Login action.**
   a. In the **Item** column below the last step, click **NEW STEP** to insert the next step. A blank edit field opens.
   b. Click in the edit field. The Item list opens, listing the sibling objects of the previous step's test object.

c. Select **OK** from the **Item** list.

This step instructs UFT to click **OK** after entering the Username and password for the application.

7. **Save your test.**

Select **File > Save**.

To learn more about the Keyword View and Editor, continue to "Analyzing the Login action in the Keyword View and the Editor" below.

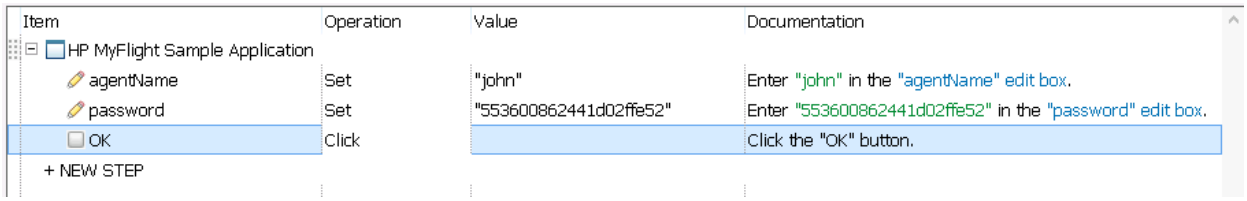
To continue adding steps to your test, go to "Exercise 3b: Add steps to the FlightFinder action by recording" on page 54.

Analyzing the Login action in the Keyword View and the Editor

Now that you have created some test steps, let's look how these steps are displayed, both in the Keyword View and Editor:

Keyword View

After you have added your steps, the Keyword View should look similar to this:



As you can see, the steps in your test are arranged in an icon-based grid which shows the test step sequence as well as the object hierarchy. Each line in the Keyword View represents a different piece of information:

Line	Description
HP MyFlight Sample Application	The **HP MyFlight Sample Application** window object is the parent object for the test objects contained in this action. All test objects contained in the following steps are displayed as sub-nodes to this object.
agentName Set "john"	**agentName** is the name of the edit box on which UFT performs the action. The **Set** method is the action performed on the **agentName** object.

	UFT enters `john` as the text in the edit box.
password SetSecure	**password** is the name of the edit box on which UFT performs the action. The **SetSecure** method is the action performed on the passwordWatermakr object. UFT enters the digit string as the text in the password edit box.
OK Click	**OK** is the name of the button UFT clicks after the information is entered in the agentName and password edit boxes. The **Click** method is the action performed on the button.

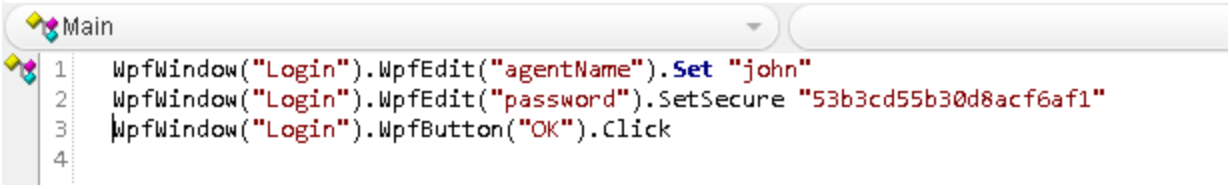
For each step in the Keyword View, there are a number of different elements:

Keyword View Element	Description
Item	The item for the step (test object, utility object, function call, or statement). This item is displayed in a hierarchical, icon-based tree.
Operation	The operation to perform on the Item, such as **Click**, **Set**, or **Select**.
Value	The argument values for the selected operation, if required. For example, the text to enter in an edit box, or the mouse button to use when clicking an image.
Documentation	The automatically provided statement of what the step does, in an easy-to-understand sentence. For example, `Click the "OK" button`.
Assignment	The assignment of a value to or from a variable so you can use the value later in the test. This column is not visible by default.
Comment	Any textual information you want to add regarding the step. For example, you could ad add a comment `Return to page used in first step of test.` This column is not visible by default.

Tip: You can hide or display individual columns by right-clicking the column heading in the Keyword View, and selecting a column name from the list.

Editor

After you have added your steps, the Editor should look similar to this:



Unlike the Keyword View, each step in the Editor is represented by a script line, with the format:

```
<object hierarchy>.<method> <method parameters>
```

Thus, for each step (and each line in the script), you see a number of things:

Test object hierarchy	For each step in the Editor, you get the **full** object hierarchy, including: • The test object type • The object name (as identified by UFT) for each object In the first line of the example pictured above, you can see both of these elements: • The **WpfWindow** is the test object type • **"Login"** is the object name (as identified by UFT)
Object method	After the object hierarchy, you also see the method (action to be performed on the object. Each method is displayed as bolded text. In the first line of the example pictured above, the object performs the **Set** method.
Method parameters	For many methods, there are required or optional parameters that you must provide. These are listed in the Editor after the method name. In the first line of the example pictured above, the **Set** method enters the parameter **John**.

Exercise 3b: Add steps to the FlightFinder action by recording

In "Exercise 3a: Add steps to the Login action in the Keyword View", you added steps to your test to run on the Login page of the flight reservation application by creating the steps using the Keyword View.

In this exercise, you will record steps for the **Flight Finder** action you created for the Flight Finder page in the application. This action will use test objects contained in the Flight Finder shared object repository.

Tip: Before you begin your recording session, you may want to place the application window and this tutorial window side-by-side on your screen. This allows you to read the tutorial during recording.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure that the WPF add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The Flight Reservation Application solution opens, displaying the Book Flights test you created in "Lesson 1: Create a GUI test and actions" on page 29.

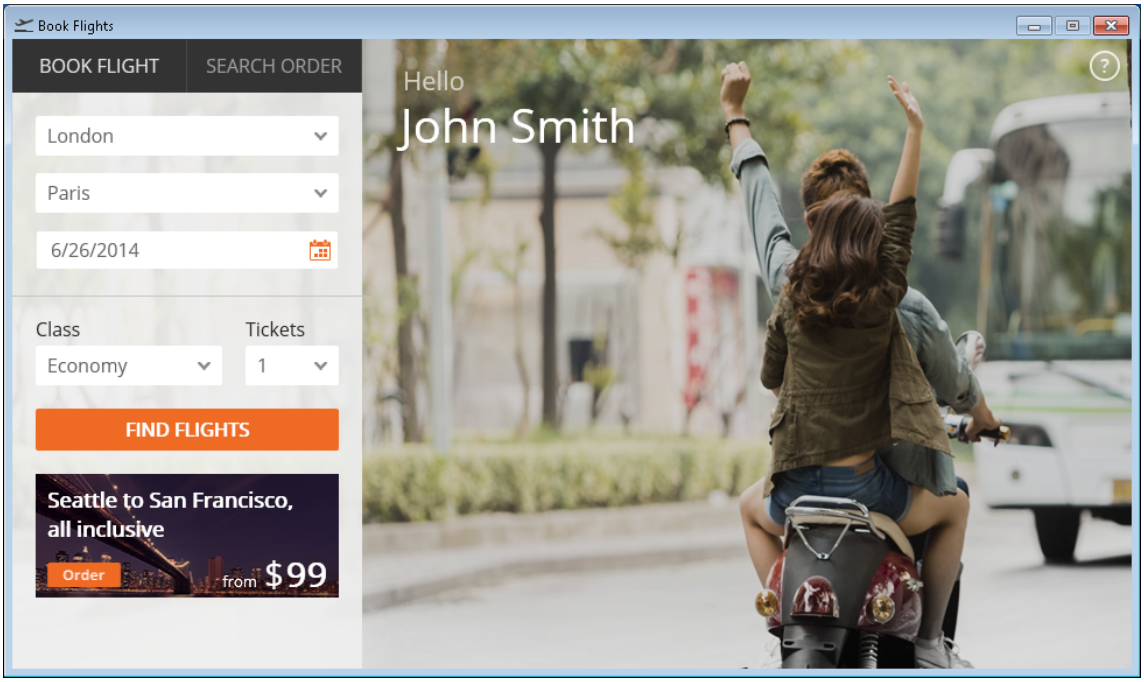
2. **Open the Flight Finder action.**
   a. If the test flow canvas is not currently open, click the **Book Flights** tab in the document pane to display it.
   b. In the canvas, double-click the **Flight Finder** action.

      The Flight Finder action opens in a separate tab in the document pane.

3. **Open the flight reservation application's**
   a. If it is not already open, open the flight reservation application, as described in "Explore the flight reservation application" on page 23.

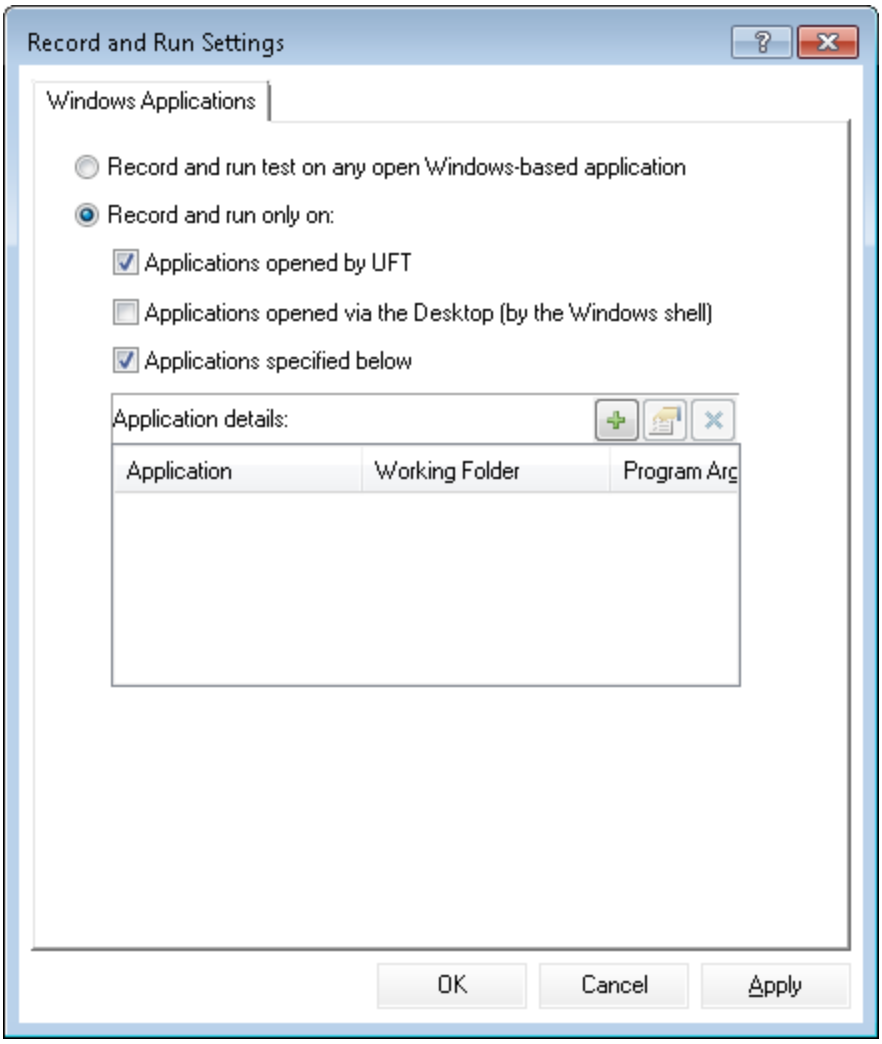
      The **Login** page of the flight reservation application opens.
   b. Log in to the application. Use `john` for the Username and `hp` for the password.

The Flight Finder page of the application opens.



c. In UFT, select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

d. In the **Windows Applications** tab of the Record and Run Settings dialog box, select **Record and run test on any open Windows-based application**. This enables UFT to run on the open application.



e. In UFT, click the **Record** button  . The recording toolbar appears near the top of the screen and UFT begins recording.

The UFT window disappears, and the Flight Finder page of the application is displayed.

f. In the Flight Finder page, change the following selections by selecting from the different items:
   - **From:** Los Angeles
   - **To:** Sydney
   - **Date:** tomorrow's date (type this using the **M/D/YYYY** syntax)
   - **Class:** `Business`
   - **Tickets:** `2`

   > **Note:** If you are performing this tutorial at the end of a month or year, select a different month or year while recording. UFT records an operation only when you make a change in the application, so this ensures that the step is recorded. If you accept a default value (or re-select the default value), UFT does not record an operation

g. After you make the selections, click **FIND FLIGHTS** to continue. The Select Flight page opens.
h. On the Record Toolbar, click the **Stop** button to stop recording.

   You have now reserved an imaginary ticket from Los Angeles to Sydney. UFT recorded your actions in the application from the time you clicked the **Record** button in UFT until the time you clicked **Stop** on the record toolbar.

4. **Save your test.**

   Click **Save**  .

Do not close the test, because you will continue to add steps to other actions. Continue with "Exercise 3c: Add a step to the Select Flight action using the Toolbox Pane" below to add additional steps to other actions.

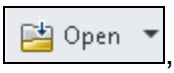
Exercise 3c: Add a step to the Select Flight action using the Toolbox Pane

In "Exercise 3b: Add steps to the FlightFinder action by recording" on page 54, you added steps to the Flight Finder action by using UFT's recording functionality. This created steps based on exactly the actions you performed in the flight reservation application's user interface.

In this exercise, you drag the relevant test objects into your action from the Toolbox pane.

> **Note:** Toolbox pane items are listed according to the action in focus in the document pane. If a the test flow tab or a function library tab is in focus, or if you

do not have a test open at all, the Toolbox pane is empty.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26.
   b. Click the **Open** button down arrow 

, and select **Open Solution**. The Open Solution dialog box opens
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.
      The Flight Reservation Application solution opens, displaying the Book Flights test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Open the Select Flight action.**
   In the Solution Explorer pane, double-click the **Select Flight** action.
   The Select Flight action opens as a separate tab in the document pane.

3. **Display the Toolbox pane.**
   In the UFT toolbar, click the **Toolbox** button  . The Toolbox pane opens on the right side of the UFT window.

   Note: The Toolbox pane displays the action's test objects and functions regardless of whether the action is displayed in the Keyword View or Editor.

4. **Select the flight to book from the grid.**
   a. In the Toolbox Pane, expand the **Test Objects** tree.
   b. In the Item column, click the **NEW STEP** button. A blank edit field opens.
   c. In the Toolbox pane, locate and drag the **flightsDataGrid** object to the edit field in the Keyword View or Editor (depending on which view is open).

      Note: You may notice that this object repository has many more objects than the Login repository. This is because you did not delete extraneous objects when creating this object repository.

      The selected Table test object is added to the step, together with its default method, **SelectCell**.
      - In the Keyword View, the **Documentation** cell is empty, because you have not provided the required parameters for the method. Remember that this step is

displayed on three rows in the Keyword View because the parent test objects are part of the step.

- In the Editor, the step is displayed as follows:

```
WpfWindow("HP MyFlight Sample Application").WpfTable
("flightsDataGrid").SelectCell
```

d. If the Keyword View is not open, select **View > Keyword View** to open it.

e. In the Value cell, click in the edit field. The Value cell becomes selected and able to edit.

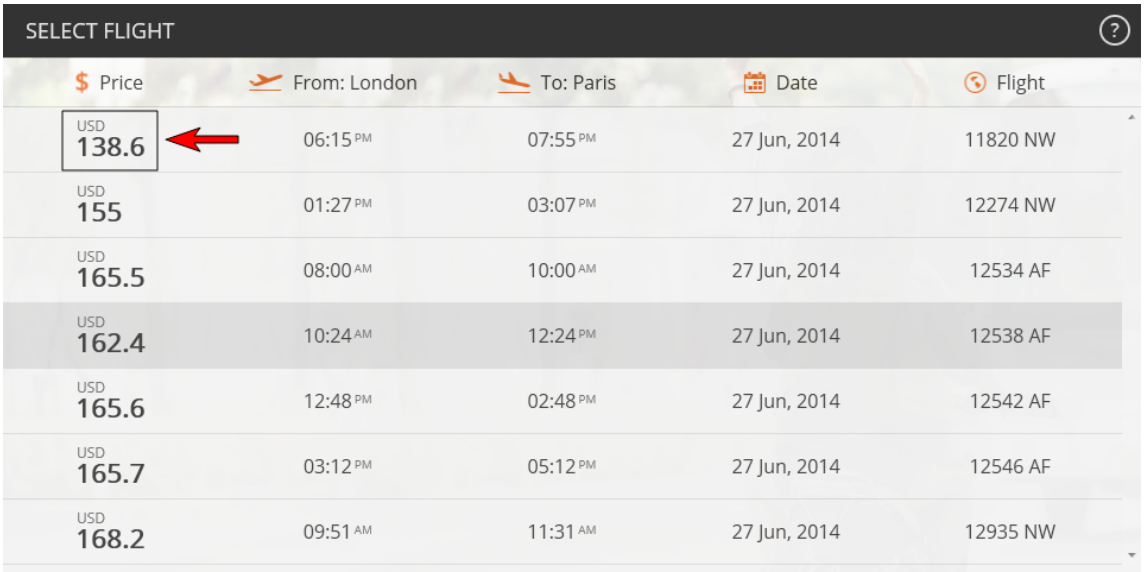
> **Note:** When you click this field, you will see a tooltip that says `row, Column`. This tells you that you are setting the value for the row parameter. When you configure the value for method parameters, you will always see a tooltip that informs you which parameter you are setting.

f. Enter `0` for the row parameter value and then enter a comma (,) character.

> **Note:** After you click the icon, you see the tooltip that says `row, Column`. This tells you that you are setting the value for the **Column** parameter.

g. Enter `0` for the column parameter.

By entering this parameter, you have instructed UFT to select the flight in the first row, as seen in the example below:



After adding the method parameters, the step is updated in the Keyword View and Editor:

- The **Value** column in the Keyword View now shows **"0","0"** for the step values.
- The Editor adds **"0", "0"** after the **SelectCell** method. Your statement in the editor now looks like this:

```
WpfWindow("HP MyFlight Sample Application").WpfTable
("flightsDataGrid").SelectCell "0", "0"
```

- The **Documentation** column now has a statement explaining the step's action.

5. **Add a step to click the Select Flight button.**

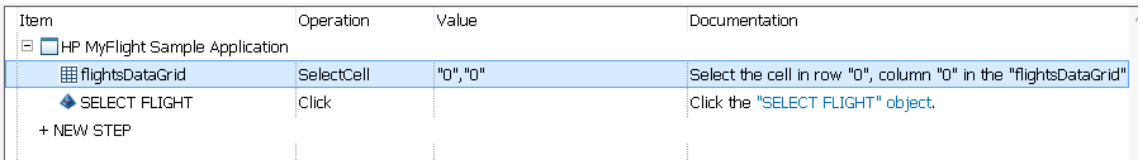
   After you select a cell from the flight list, you must also click the Select Flight button to continue with the flight reservation process.

   a. In the Toolbox pane, find the **SELECT FLIGHT** object.

   b. In the Item column, click the **NEW STEP** button. A blank edit field opens.

   c. From the Toolbox pane, drag the SELECT FLIGHT object to the to the edit field in the step grid in the Keyword View, under the **flightsDataGrid** object.

   A new step is added to your test, containing the SELECT FLIGHT object:



   In this case, the default method for the SELECT FLIGHT object, **Click**, is the method you want to use for the test.

   In the Editor, the action steps now look like this:

```
WpfWindow("HP MyFlight Sample Application").WpfTable
("flightsDataGrid").SelectCell "0", "0"
WpfWindow("HP MyFlight Sample Application").WpfObject("SELECT FLIGHT").Click
```

6. **Save your test.**

   Select **File > Save**.

Do not close the test, becuase you will still need to add steps to other actions. Continue to "Exercise 3d: Add steps to the Book Flight action using the Step Generator" on the next page to add steps by using the Step Generator.

Exercise 3d: Add steps to the Book Flight action using the Step Generator

In "Exercise 3c: Add a step to the Select Flight action using the Toolbox Pane" on page 58, you used the objects displayed in the Toolbox pane to create test steps.

In this lesson, you will use an additional way of creating test steps - the Step Generator. The Step Generator enables you to define an entire step in one dialog box, instead of inserting different parts of a step in the various columns of the Keyword View.

1. **Open the Flight Confirmation action.**

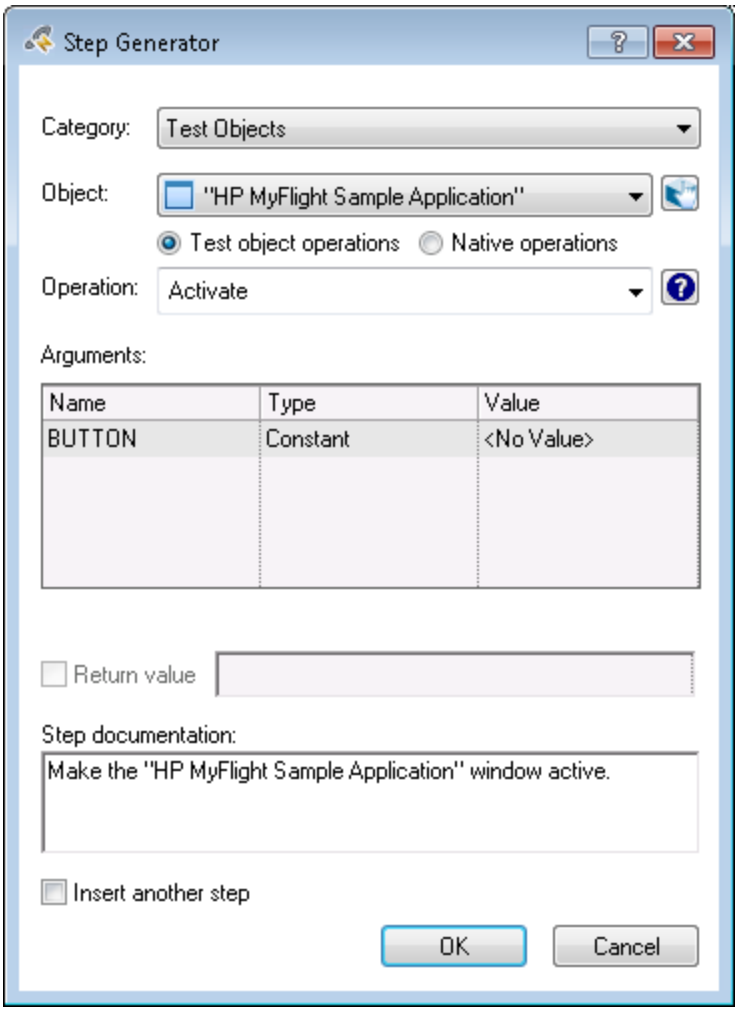
   In the Solution Explorer, under the **Book Flights** node, double-click the **Flight Confirmation** action.

   The Flight Confirmation action opens as a separate tab in the document pane

2. **Add a step using the Step Generator.**

   a. If the Editor is not already displayed, display it by selecting **View > Editor**.

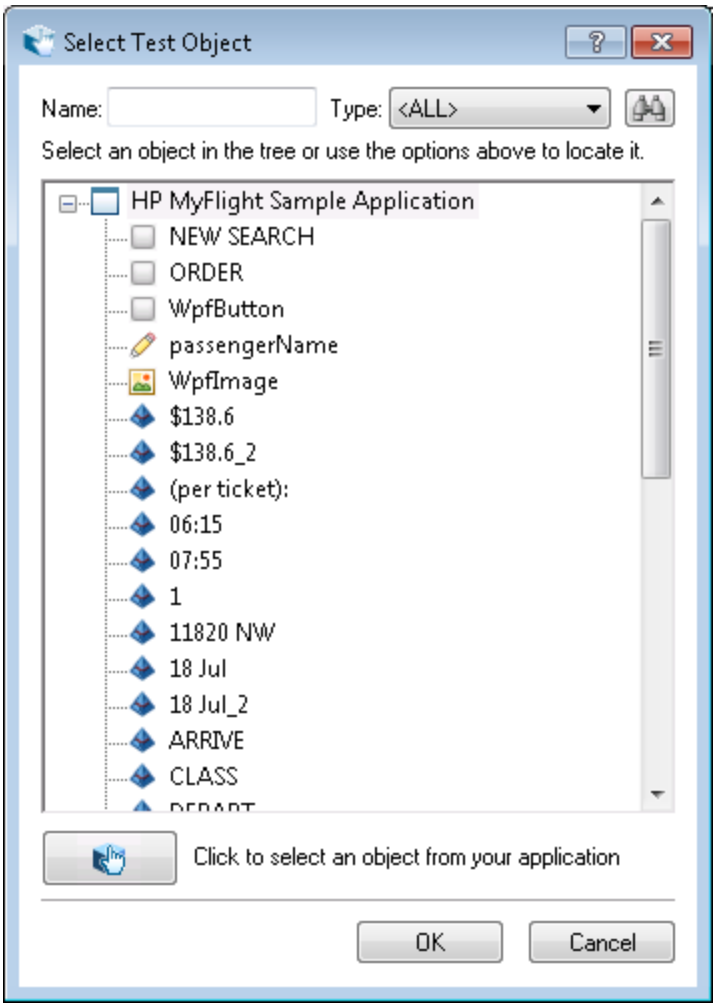
b. In the first line of the Editor, right-click and select **Insert Step > Step Generator**. The Step Generator dialog box opens.



c. In the **Object** drop-down, select **"HP MYFlight Sample Application"**.

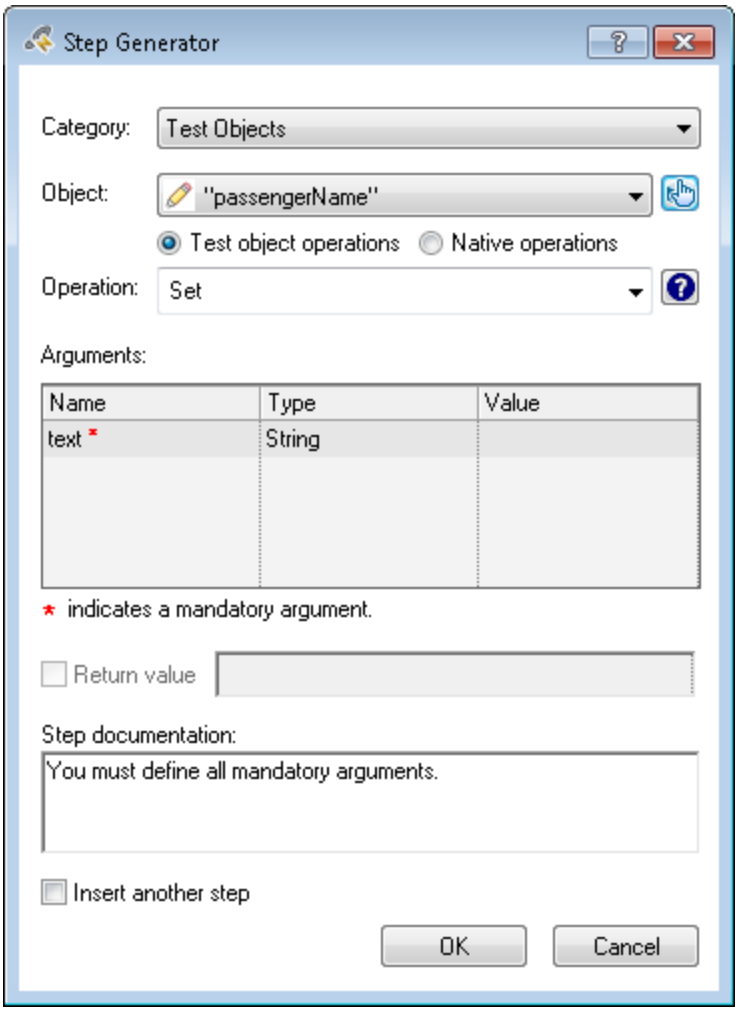
d. Click the **Select Object** button. The Select Test Object dialog box opens.

e. In the Select Test Object dialog box, expand the **HP MyFlight Sample Application** node:



f. Select the **passengerName** edit box, and click **OK**.

The Step Generator displays the default options for the **passengerName** object:



g. Define the arguments and values in the Step Generator just as you would in the Keyword View:
   - In the **Operation** drop-down list, leave the Operation as **Set**. Note that when you select a test object in the Object drop-down list, the default operation is displayed. You can select other operations for this object as needed. However, in this exercise, the default operation is the correct one.
   - In the **Arguments** table, click inside the **Value** column (like you would in the Keyword View), and enter a name of your choosing. Note that if arguments are mandatory, a red asterisk is displayed next to the argument name.
   - The **Step documentation** displays the instruction for this step as it will be displayed in the **Documentation** cell of the Keyword View.
   - Select the **Insert another step** check box to open the Step Generator dialog box again after adding this step.

h. Click **Insert**. The Step Generator remains open, but a step is added to the Editor in the background with the details you entered.

3. **Add a step to wait for the progress bar in the application window to load.**

   In the Flight Details window, you will notice that there is a progress bar object, called **progBar** in the object repository. In order for the application to work correctly when tested, you must add a step to ensure that this loads before clicking the button to complete the order.

   a. In the Step Generator, in the **Objects** drop-down list, select the **progBar** object. The Step Generator updates the dialog box fields with the default properties for the **progBar** object, including its default method, **Value**.

   b. In the **Operation** drop-down menu, select **WaitProperty**. This method instructs UFT to wait during the test run until a certain property achieves a specified state.

   c. In the **Arguments** table, enter the following information:

Item	Value
PropertyName	`value`
PropertyValue	`100`

   d. Select the **Insert another step** check box and click **Insert**.

   Another step is inserted in the Keyword View in the background and the Step Generator box remains open.

4. **Add a step to the test to complete the order using the Step Generator.**

   Now that you have defined the details for the order, you must provide a step that clicks the **ORDER** and **NEW SEARCH** buttons in the Flight Details window to complete the order. You will use the Step Generator again to insert this statement.

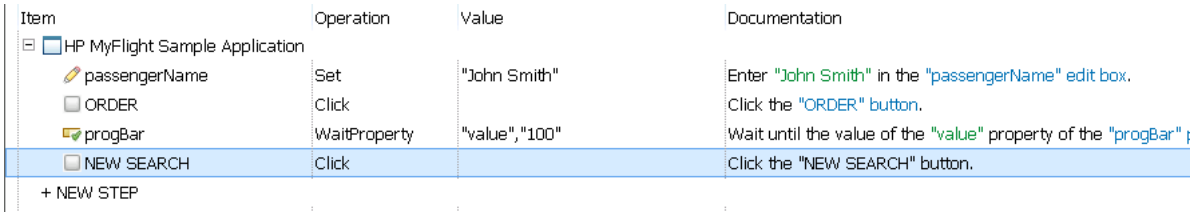
   a. Using the process in the previous steps, enter the following details (left column first):

> **Note:** The step that the clicks the **ORDER** button must be inserted before the step with the progress bar object created in the previous step.

	ORDER button step	NEW SEARCH button step
Object	ORDER (Wpf Button)	NEW SEARCH (WpfButton)
Operation	Click	Click
Arguments	Leave blank	Leave blank
Insert another step checkbox	Select	Clear

b. Click **OK**. The Step Generator closes and the step is added to the Keyword View.

Now that you have added both steps to the test, the Keyword View should look like this:



In the Editor. the steps are displays as follows:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("passengerName").Set "John
Smith"
WpfWindow("HP MyFlight Sample Application").WpfButton("ORDER").Click
WpfWindow("HP MyFlight Sample Application").WpfProgressBar
("progBar").WaitProperty "value", "100"
WpfWindow("HP MyFlight Sample Application").WpfButton("NEW SEARCH").Click
```

5. **Save your test.**

Click **Save**  .

Now that you have created your first test, you can run it. Continue with "Lesson 4: Run and analyze GUI tests" on page 73 to learn more about running tests.

If you would like an advanced lesson, continue to "Advanced Exercise 3e (Optional) - Add steps using the Editor" below to learn how to add steps in the Editor.

Advanced Exercise 3e (Optional) - Add steps using the Editor

In addition to adding steps to your test through the Keyword View, the Toolbox pane, or the Step Generator, you can also add steps directly into the Editor.

However, when adding steps through the Editor, you must have a greater knowledge of both your application and your test objects. In the Keyword View, Toolbox pane, and Step Generator, all the information for your test objects is provided in the dialog boxes by UFT. In the Editor, you must know a number of things:

- The full test object hierarchy for your objects
- The name of the test objects (as recorded in the Object Repository)
- The type of the test objects, i.e. WpfWindow, WpfButton, etc.
- The method to use

You use this information to create lines in the Editor. In this lesson, you will learn where to find this information and how to enter it into the Editor to make test steps.

You will create statements in the Editor for an action which already has test steps.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure that the WPF add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The Flight Reservation Application solution opens, displaying the Book Flights test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Open the Flight Confirmation action.**

   In the Solution Explorer pane, double-click the **Flight Confirmation** action.

   The Flight Confirmation action opens as a separate tab in the document pane. Do not delete the currently existing test steps in the action.

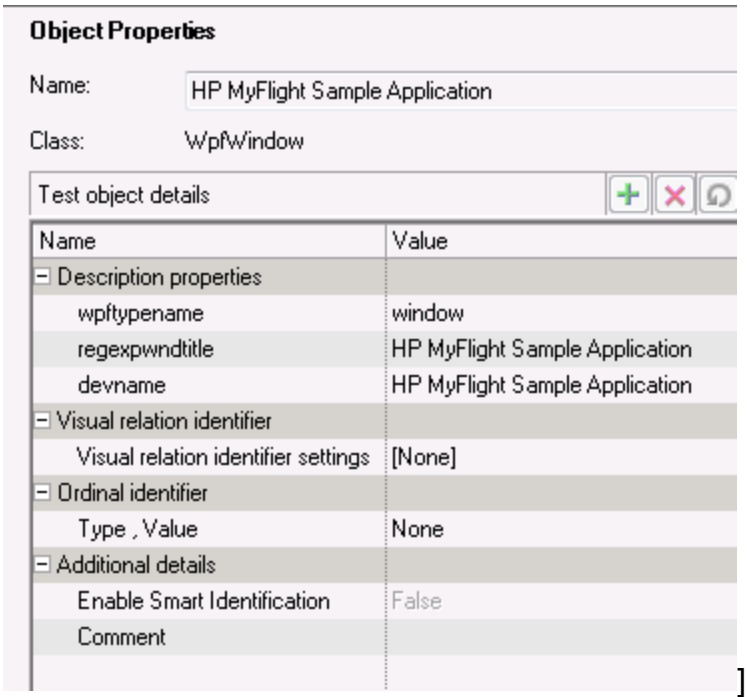
3. **Learn the object details for the parent object.**
   a. If it is not already open, open the Editor by selecting **View > Editor**.

      Note that the already existing steps look like this:

      ```
      WpfWindow("HP MyFlight Sample Application").WpfEdit("passengerName").Set "John
      Smith"
      WpfWindow("HP MyFlight Sample Application").WpfButton("ORDER").Click
      ```

      For a description of each of the elements in these statements, see the **Editor** section in "Analyzing the Login action in the Keyword View and the Editor" on page 52.
   b. In the Solution Explorer, under the **Book Flights** node, expand the **Flight Confirmation** action node.
   c. Double-click the **Flight Confirmation.tsr** object repository. The Object Repository Manager opens with the objects for the Flight Details page of the flight reservation application.
   d. In the Object Repository Manager, select the **HP MyFlight Sample Application** object (top level node). This is the parent object for all objects in the Flight Details page.

The object's details and properties are shown in the **Object Properties** area (right side of the window):



]

e. Record the following details for the **HP MyFlight Sample Application** object:

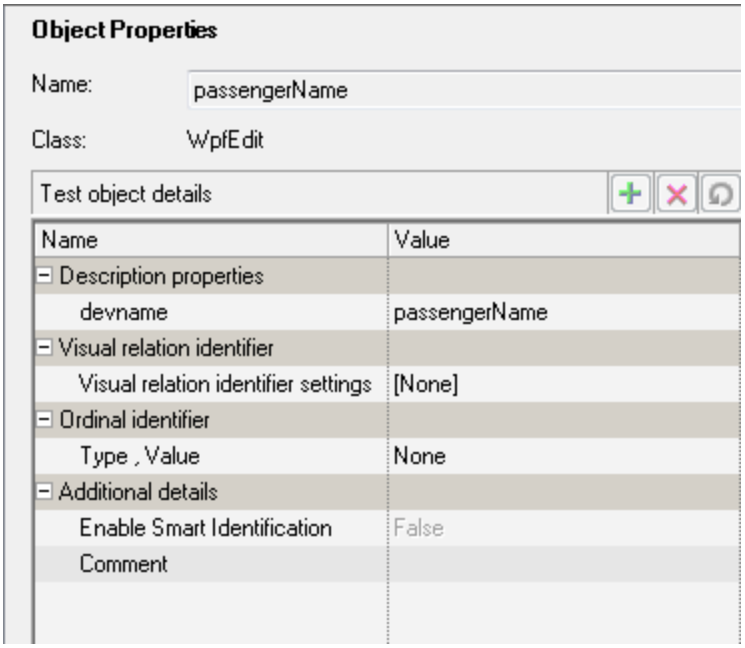
- **Name: HP MyFlight Sample Application**
- **Class: WpfWindow**

You will need this when you create a statement in the Editor. This information is the first part of all statements for steps using test objects on this page.

4. **Learn the object details for the child objects.**

In "Exercise 3d: Add steps to the Book Flight action using the Step Generator" on page 62 (the exercise on which this exercise is based), you created two steps: one step that enters the name for the flight order, and the second step that clicks the **ORDER** button. In order to create these steps in the Editor, you must also learn the object details for objects included in these steps.

a. In the Object Repository Manager, select the **passengerName** object. The test object details are shown in the **Object Properties** section of the Object Repository Manager:



b. Record the following properties for the object:
   - **Name: passengerName**
   - **Class: WpfEdit**

c. Follow the same process for the **ORDER** object.

5. **Create the statements for the steps in the Editor.**

   After you viewed the object properties for the objects involved in this action's steps, you should have the following information:

Object	Name	Class
HP MyFlight Sample Application window (parent object)	HP MyFlight Sample Application	WpfWindow
Passenger Name (edit box)	passengerName	WpfEdit
ORDER button	ORDER	WpfButton

   Using the object details, you must create statements that include the object hierarchy as well as the method (action) to be performed on the object. (Each object has a number of supported methods to use for the test object. For full details on all available objects and their methods, see the *UFT Object Model Reference for GUI Testing* after you finish the tutorial exercises.)

   a. On the first new line, enter the parent object hierarchy for the first step (entering the customer name for the order), using the format

```
<object class>("<object name>").
```

For this step, you should enter the following:

```
WpfWindow("HP MyFlight Sample Application").
```

b. Enter the child object (**passengerName**) for the first step, using the same format.
For this step, you should enter the following:

```
WpfEdit("passengerName").
```

c. After the **WpfEdit("passengerName")** object, enter the **Set** method for the **passengerName** object.
Your step should now look like this:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("passengerName").Set
```

d. After the **Set** method, enter `"John Smith"` as the string to be entered (Set) for the **passengerName** object.
Your step should now look like this:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("passengerName").Set "John
Smith"
```

e. Add the object hierarchy for the second step (clicking the **ORDER** button), using the same process as described above.
After you add the object hierarchy for the second step, the statement should look like this:

```
WpfWindow("HP MyFlight Sample Application").WpfButton("ORDER").
```

f. After the **WpfButton("ORDER")** object, add the **Click** method. Your statement should now look like this:

```
WpfWindow("HP MyFlight Sample Application").WpfButton("ORDER").Click
```

> **Note:** The **Click** method does not require any parameters, so there is no need to add additional information after the method name.

Once you have completed both statements, the Editor should display the following:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("passengerName").Set "John
Smith"
WpfWindow("HP MyFlight Sample Application").WpfButton("ORDER").Click
```

6. **Remove the extra lines.**

After you performed the previous steps, you added two extra lines to the action (for a total of four statements). Remove the last two statements from the action to ensure that the test runs successfully.

7. **Save the test.**

   Select **File > Save**.

Lesson 4: Run and analyze GUI tests

In "Lesson 3: Add steps to a test" on page 46, you created multiple actions and test steps in each action to test the flight reservation application. Now that the test is complete, you can run the test to see how the flight application performs.

In this lesson, you will learn how to run a test and view the run results.

This lesson includes the following:

Exercise 4a: Run a test

In "Lesson 3: Add steps to a test" on page 46, you created a basic test that runs through the flight reservation application to book a flight.

In this exercise, you will learn how to run the test you just finished.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure the WPF Add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The Flight Reservation Application solution opens in the Solution Explorer, and the Book Flights test opens as a separate tab in the document pane.

2. **Configure UFT to save all images to the run results.**

   When you perform a test run, UFT gives you the option to save all images to the test results.
   a. Select **Tools > Options > GUI Testing** tab > **Screen capture** node. The Screen Capture options pane opens.
   b. In the Screen Capture options pane, select the **Save still image captures to results** checkbox, and then select **Always** from the drop-down menu.
   c. Click **OK** to close the Options dialog box.

3. **Configure the Record and Run Settings for your test.**

   In some cases, you may need UFT to open your application for you at the beginning of the test run. In these cases, you can set up the Record and Run Setting to enable this.
   a. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.
   b. In the Windows Application tab of the Record and Run Settings dialog box, select the **Record and run only on:** option.
   c. Under the Record and run only on option, select the **Applications specified below** option.
   d. In the Application details area, click the **Add** button  . The Application Details dialog box opens.

e. In the Application details dialog box, enter the application details:

Application	C:\%HOMEPATH%\Unified Functional Testing\samples\Flights Application\FlightsGUI.exe
Working folder	C:\%HOMEPATH%\Unified Functional Testing\samples\Flights Application\

f. Click **OK** to close the Application Details dialog box.

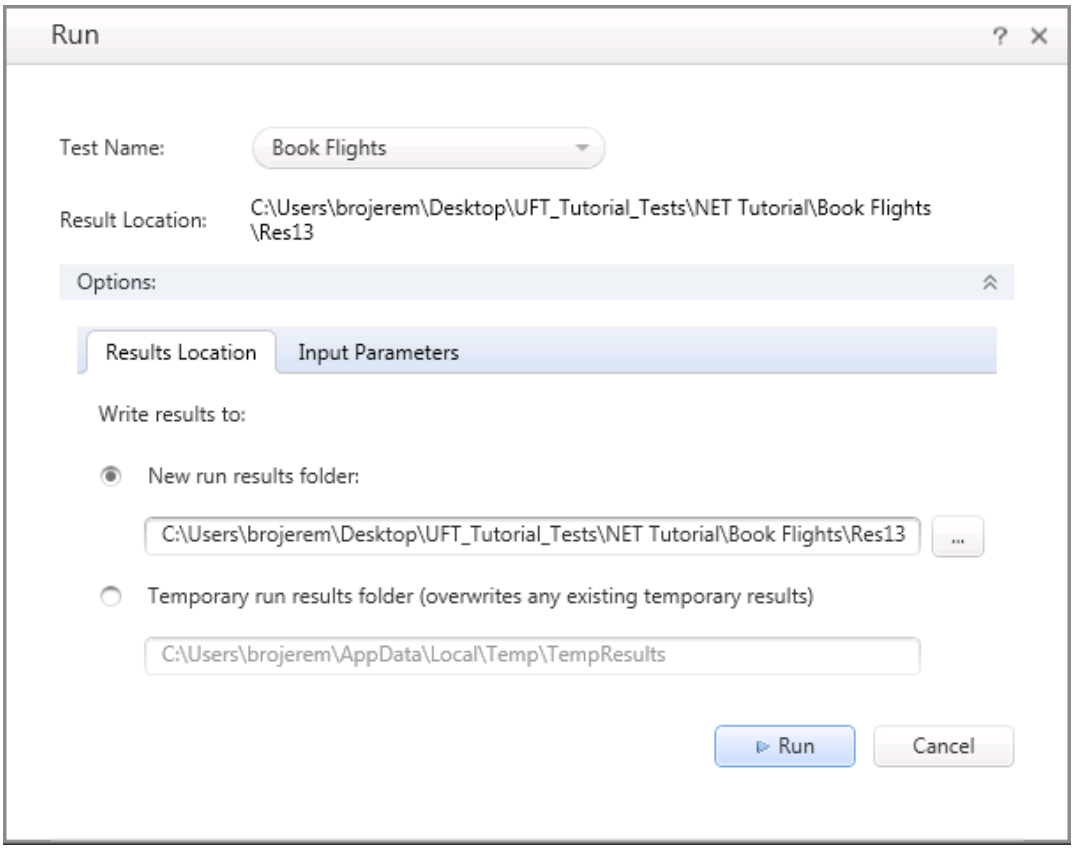
g. In the Record and Run Settings dialog box, click **Apply** and then **OK** to enable the settings and close the dialog box.

4. **Start running your test.**

a. In the Solution Explorer, select the **Book Flights** test node.

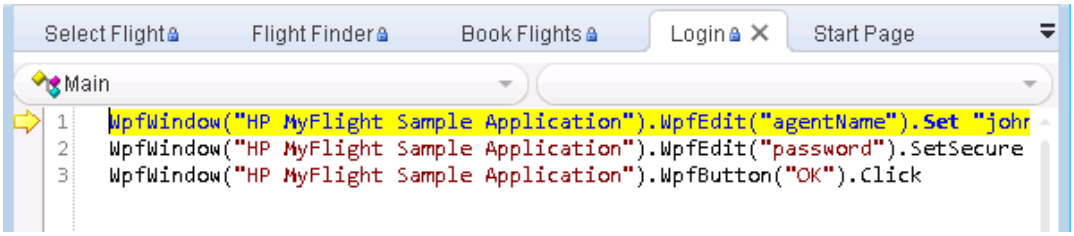
b. In the toolbar, click the **Run** button  . The Run Test dialog box opens.

c. In the Run Test dialog box, click the **Options** bar to expand the Run Test Options area, Ensure that the **New run results folder** option is selected. Accept the default folder name:



d. Click **Run** to close the Run dialog box and start the test run.

Watch carefullly as UFT opens the application and starts running the test. In the application, you can see UFT perform each step you inserted: a yellow arrow in the left margin of the Keyword View or Editor and the highlighted row indicate the step that UFT is running:



If any errors appear, go to the point in the test that is indicated in the error message and verify that the step is configured as described in the relevant exercise.

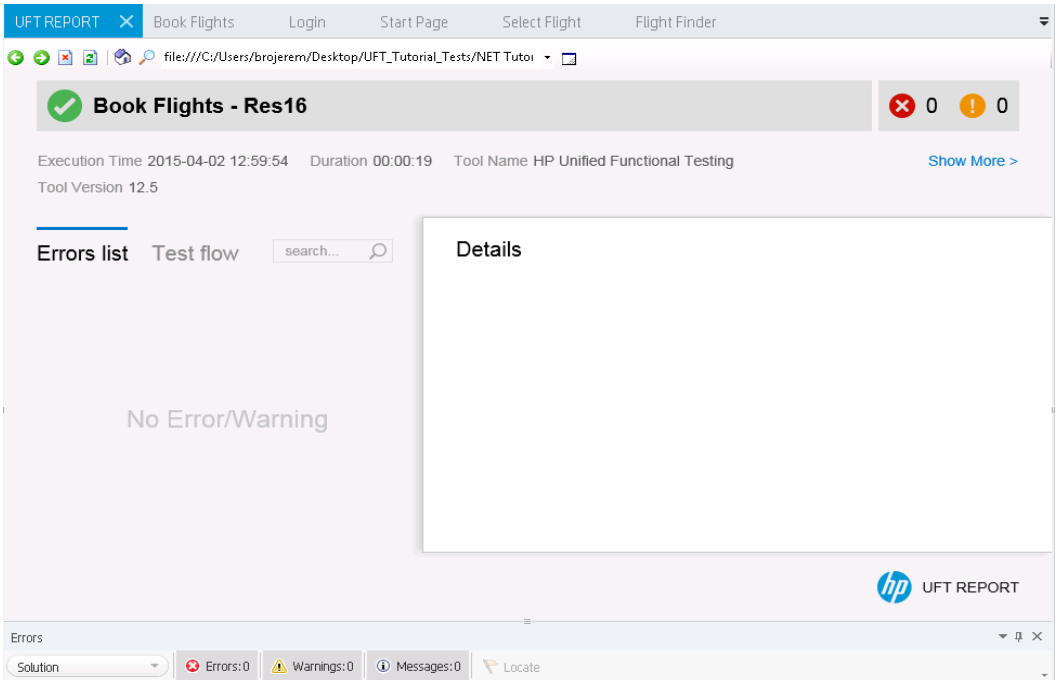
When the test run is complete, the run results open as a separate tab in the document pane. Continue to "Exercise 4b: Navigate the run results" below to learn more about the run results.

Exercise 4b: Navigate the run results

In "Exercise 4a: Run a test" on page 74, you ran the Book Flights test you created. After the test run is finished, the run results automatically display the results for this test run.

Note: By default, the run results are displayed in an HTML-based report. You can also choose to have the run results displayed in the Run Results Viewer in the **Run Sessions** pane of the Options dialog box (**Tools > Options > General** tab > **Run Sessions** node). The lessons in this tutorial are based on the HTML-based report.

When the run results open, UFT displays the following:



By default, the run results display the following:

Test Flow	A graphic representation of the results in a tree, organized according to the actions and the application pages visited during the test run. The steps performed during the run are represented by icons in the tree, which can be expanded (**arrow**) to view each step. You can instruct UFT to run a test or action more than once using different sets of data in each run. Each run is called an iteration, and each iteration is numbered. (The test you ran had only one iteration.)
Error list	A list of all the errors and warnings, presented in a list.
Step summary information	A high-level results overview report, containing general information about the test, which steps passed or failed, and details about each test step..
Links to external resources	Links to external resources used with your test or from the test run, including: • Data table • Run-time movies • Application logs

In this exercise, the test run succeeded because UFT was able to navigate the flight booking application according to the steps you added. If an error occurred and your test did not run successfully, the error will be shown in the run results. In such cases, go back and make sure that the steps are configured exactly as described in this tutorial.

Now that you know what the run results display, continue to "Exercise 4c: Analyze the run results" below to learn about the details of the run results.

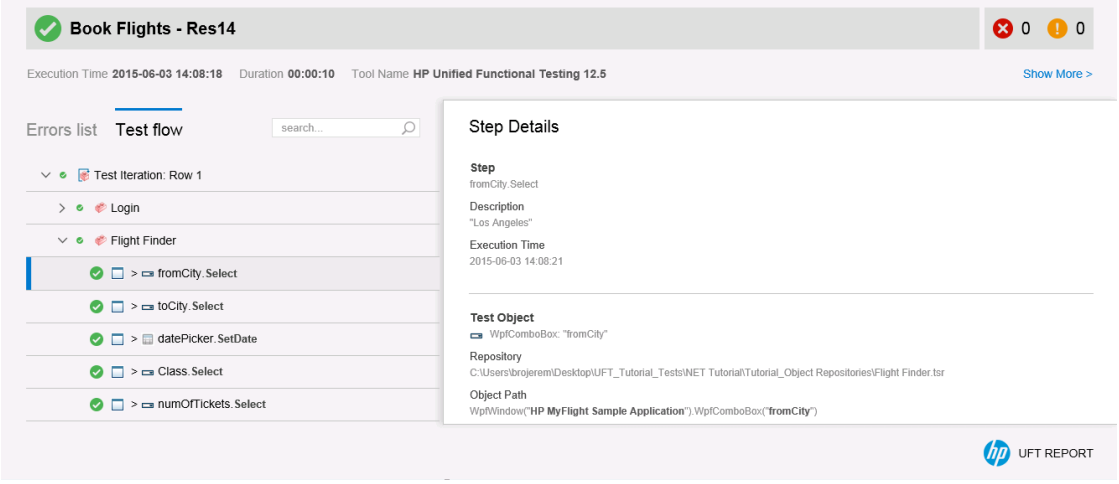
Exercise 4c: Analyze the run results

In this exercise, you will inspect the steps UFT performed when it ran your test in "Exercise 4a: Run a test" on page 74. You can view snapshots of the application window for each step.

1. **View the run results for a specific step.**

   In the Test Flow, under the results tree, find the **Flight Finder** node to see all the steps performed on the Flight Finder page of the flight reservation application.

   In the results tree, select the **fromCity.Select** step:



   The run results now displays the following information:

   - The Test Flow, with the step highlighted
   - A summary of the test step, displaying details of the highlighted step

2. **Close the run results.**

   In the document pane, close the tab containing the run results.

Now that you have set up and run your first test, you can continue to learn about different ways of enhancing your tests. Select one of the following to learn more:

- "Lesson 5: Parameterize steps and objects" on the next page
- "Lesson 6: Creating checkpoints and output values" on page 95
- "Lesson 7: Create functions and function libraries" on page 131
- "Lesson 8: Using Insight in your Test" on page 141

Lesson 5: Parameterize steps and objects

In "Lesson 3: Add steps to a test" on page 46, you created test steps to check that a series of steps performed on the flight reservation application ran smoothly. In "Lesson 4: Run and analyze GUI tests" on page 73, you ran the test, but only with a single set of data. However, when you test your applications, you may want to see the same operations performed with multiple sets of data.

For example, you may want to run a test on your application using ten different sets of data. You can create ten separate tests, each with its own set of data, or you can add ten sets of parameters to a single test. If you add the parameters, your test will run ten times, each time using a different set of data.

In this lesson, you will add parameters to your test and run the test with multiple sets of data.

This lesson includes the following:

Parameterizing tests, actions, and objects

When you use data to parameterize a test, action or test object value, you can provide the data source from a number of places:

- **Data table:** an Excel spreadsheet with parameter names and values
- **Environment variable:** a variable set in your test with a fixed value
- **Random number:** a random number generated by UFT in the test run

The most common of these is the Data table parameter. The data table is an Excel spreadsheet, displayed in the Data pane at the bottom of the UFT window.

> **Note:** If the Data pane is not displayed, select **View > Data** or click the **Data** button in the toolbar.

In the Data table, there are two different types of sheets:

Global data sheets	Global data sheets contain data parameters and data that is used for and available to all actions in the test. When a parameter is inserted in the global sheet, it can be used in any of the actions and the steps in the actions in the test. The test will run the same number of iterations as there are rows in the global data sheet. Thus, for example, if there are five rows of data, the test will run five iterations.
Action sheets	For each action in your test, UFT adds an additional sheet with that action (with the same name as the action). The data parameters and data is available only to the steps in that action. If you use multiple rows of data within an action sheet, UFT will run the action the same number of times as there are rows in the data sheet (within one test iteration).

In the course of this lesson, you will be using only data table parameters. For details on the other types of parameters, see the section on parameterization in the *HP Unified Functional Testing User Guide*.

Exercise 5a: Create a test for parameterization

In "Exercise 3b: Add steps to the FlightFinder action by recording" on page 54, you reserved a flight from Los Angeles to Sydney. In those step, the Los Angeles and Sydney values are constant values. This means that UFT uses **Los Angeles** and **Sydney** as the departure and arrival city each time it runs the test.

In this exercise, you will create a new test, in which you define the departure and arrival city as a parameter, so that you can use a different departure and arrival city for each test run.

1. **Start UFT and open the Book Flights test.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Save the Book Flights as Book Flights Parameter.**
   a. In the Solution Explorer, select the **Book Flights** node.
   b. Select **File > Save As**. In the Save Test As dialog box, browse to **C:\%HOMEPATH%\My Documents\Unified Functional Testing** and save the test as **Book Flights Parameter**.

      In the Solution Explorer, the **Book Flights** test is replaced by the new **Book Flights Parameter** test. The Book Flights test is still saved separately in the file system.

3. **Add the Book Flights test back to the solution.**

   You can have all of your tests open at the same time if they are both referenced from the same solution. This enables you switch back and forth between them if you want to compare or edit the tests. You can only run a single test at a time.
   a. Select **File > Add > Existing Test**.
   b. In the Add Existing Test Dialog Box, browse to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and select the **Book Flights** test.
   c. Click **Add** to add it to the solution.

      The Book Flights test is again displayed in the Solution Explorer. Note that it appears above the Book Flights Parameter test you just created, as tests are listed in alphabetical order.

   Now that you have created the test for parameterization,continue with "Exercise 5b: Define data table parameters" below to create data table parameters in your test.

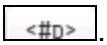
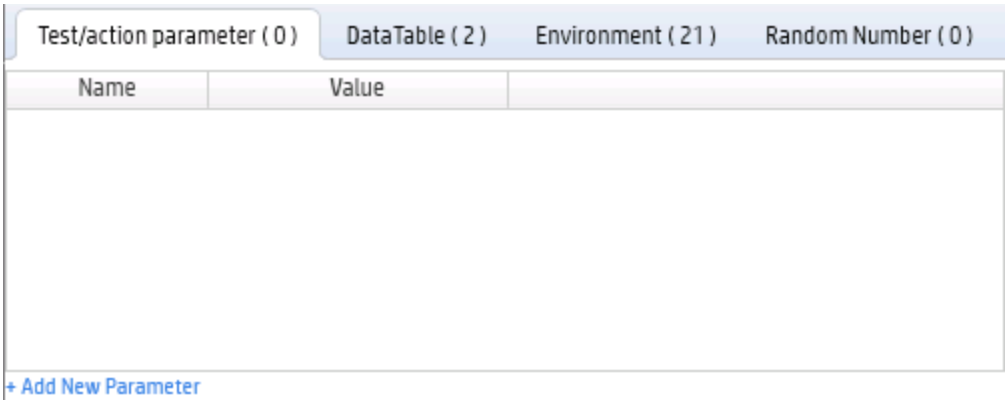
Exercise 5b: Define data table parameters

In this lesson, you will define the departure and arrival cities as parameters, so that you can use a different departure city for each test run.

1. **Start UFT and open the Book Flights Parameter test, if necessary.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Select **File >Open > Solution**. The Open Solution dialog box opens
   c. Navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.
      The Flight Reservation solution opens, including the Book Flights Parameter test you created in "Exercise 5a: Create a test for parameterization" on page 81.
   d. In the Solution Explorer, select the **Book Flights Parameter** node.

2. **Make sure that the Data Pane is visible.**
   If you do not see the Data pane at the bottom of the UFT window, select **View > Data**.

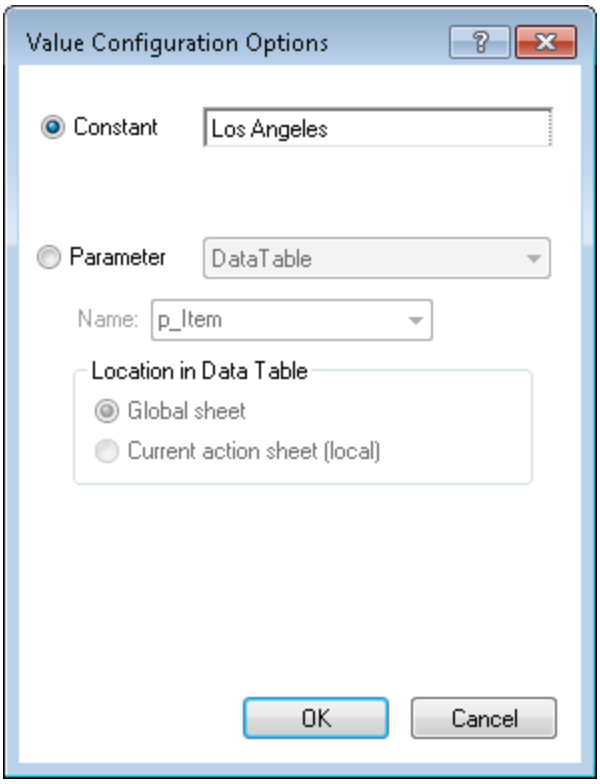
3. **Open the Flight Finder action.**
   a. In the canvas, double-click the **Flight Finder** action. The Flight Finder action is displayed as a separate tab in the document pane.
   b. If necessary, select **View > Keyword View**.

4. **Select the text to parameterize.**
   In the Keyword View, in the **fromCity** row, click the **Value** cell and then click the parameterization button 
.
   The parameter list is displayed:


5. **Set the parameterization properties.**
   a. In the parameter list, select the **Data Table** tab. This enables you to replace the constant value (**London**) with a parameter.

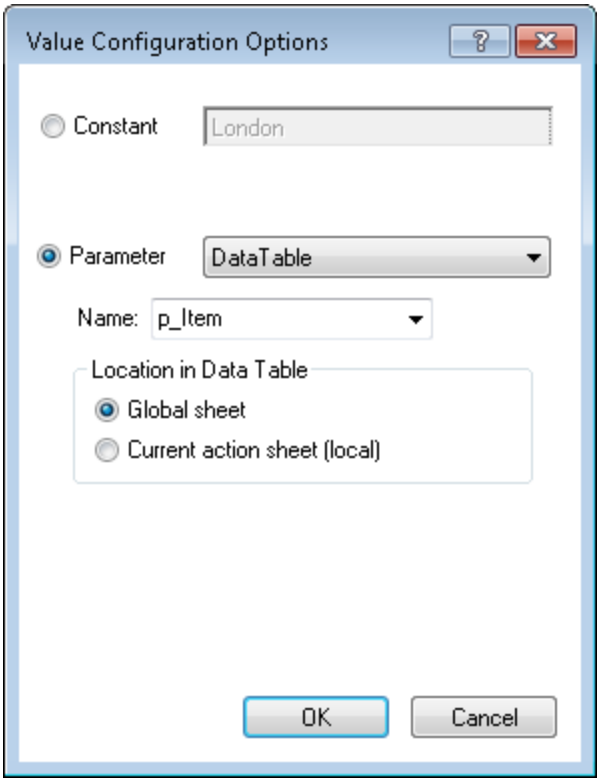
Note that no parameters are displayed, because you have not yet created a Data Table parameter.

b. In the Data Table tab of the parameter list, click the **Add New Parameter** button. The Value Configuration Options dialog box opens:



c. In the Value Configuration Options dialog box, select the **Parameter** radio button.

d. Confirm that the **DataTable** option is selected from the Parameter drop-down menu. This means that the value of the parameter will be taken from the UFT Data pane. The **Name** box is enabled and displays **p_Item**:

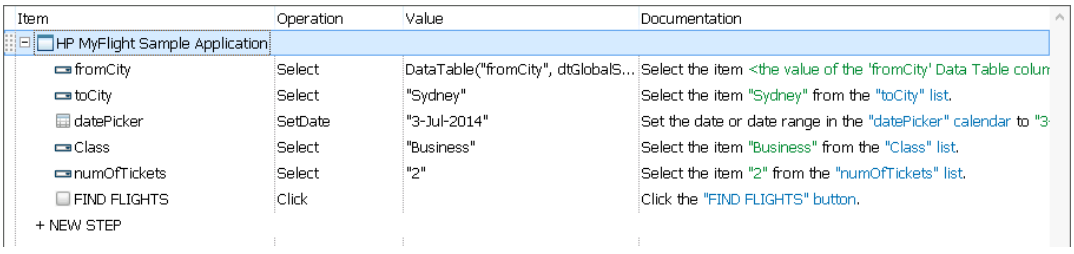


e. Delete the **p_Item** parameter and enter **fromCity**.

f. Click **OK** to close the dialog box.

UFT adds the **fromCity** parameter to the Data pane as a new column and inserts **Los Angeles** (the previous constant value) in the first row in the column.

Los Angeles will be the first of several departure cities that UFT will use during test runs of the application.



Item	Operation	Value	Documentation
HP MyFlight Sample Application			
fromCity	Select	DataTable("fromCity", dtGlobalS...	Select the item <the value of the 'fromCity' Data Table colun
toCity	Select	"Sydney"	Select the item "Sydney" from the "toCity" list.
datePicker	SetDate	"3-Jul-2014"	Set the date or date range in the "datePicker" calendar to "3-
Class	Select	"Business"	Select the item "Business" from the "Class" list.
numOfTickets	Select	"2"	Select the item "2" from the "numOfTickets" list.
FIND FLIGHTS	Click		Click the "FIND FLIGHTS" button.
+ NEW STEP			

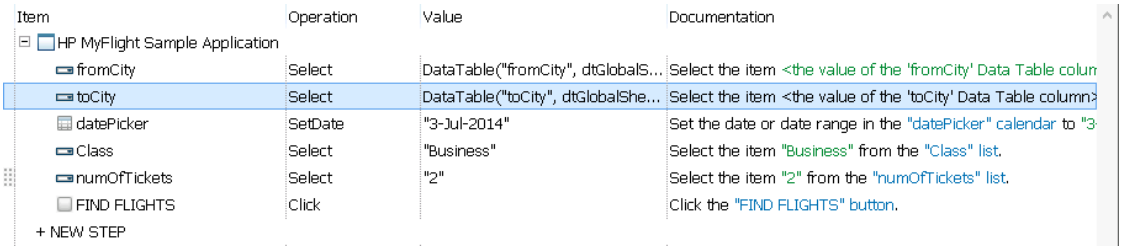
Note the change in the step's appearance in the Keyword View. Previously, the step was displayed as **fromCity Select Los Angeles**. Now, when you click the **Value** cell, the following information is displayed, indicating that the value is parameterized using a Data pane parameter called **fromCity**:

DataTable("fromCity", dtGlobalS...

6. **Add a data table parameter for the toCity step.**

   Using the process described in the previous step, add a data table parameter for the toCity object named toCity.

   After you are done, your test should look like this:



Item	Operation	Value	Documentation
HP MyFlight Sample Application			
fromCity	Select	DataTable("fromCity", dtGlobalS...	Select the item <the value of the 'fromCity' Data Table colum
toCity	Select	DataTable("toCity", dtGlobalShe...	Select the item <the value of the 'toCity' Data Table column>
datePicker	SetDate	"3-Jul-2014"	Set the date or date range in the "datePicker" calendar to "3
Class	Select	"Business"	Select the item "Business" from the "Class" list.
numOfTickets	Select	"2"	Select the item "2" from the "numOfTickets" list.
FIND FLIGHTS	Click		Click the "FIND FLIGHTS" button.
+ NEW STEP			

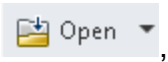
7. **Save your test.**

   Select **File > Save**.

Continue with "Exercise 5c: Add parameter values to a data table" below to learn how to populate the data table with the values used for a test run.

Exercise 5c: Add parameter values to a data table

As you learned in "Exercise 5b: Define data table parameters" on page 82, UFT displays parameter values in the Data pane. In this exercise, you will add another departure city (for the fromCity object) to the Data pane, so that UFT can test the application with this data.

1. **Start UFT and open the Book Flights Parameter test, if necessary.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Click the **Open** down arrow 

, and select **Open Solution**. The Open Solution dialog box opens.
   c. Navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The Flight Reservation solution opens, including the Book Flights Parameter test you created in "Exercise 5a: Create a test for parameterization" on page 81.

d. In the Solution Explorer, select the **Book Flights Parameter** node.

2. **Open the Flight Finder action.**

   In the Solution Explorer, double-click the **Flight Finder** action.

   The Flight Finder action opens as a separate tab in the document pane.

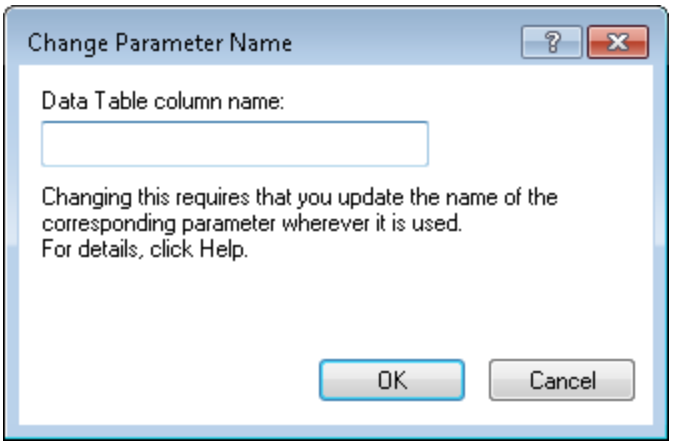
3. **Enter an additional cities in the fromCity column.**

   In the Data pane, enter the following in the Data pane for the **fromCity** parameter:

Row	Value
2	Denver
3	Frankfurt
4	London

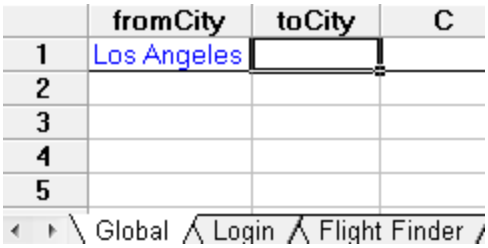
4. **Create a data table parameter and values for the toCity object.**

   a. In the Data pane, double click the header row for the **B** column. The Change Parameter Name dialog box opens.



b. In the Change Parameter name dialog box, enter **toCity** for the parameter name and click **OK**.

The column header (which was previously **B**) is updated with the new parameter name:



c. Enter the values for the **toCity** parameter as follows:

Row	Value
1	Sydney
2	Los Angeles
3	London
4	Frankfurt

After you add the second parameter and its values, the Data pane should look like this:

5. **Parameterize the toCity step.**

a. In the Flight Finder action, in the **toCity** row, click the **Value** cell and then click the parameterization button <#D>. The Value Configuration Options dialog box opens.

b. In the Value Configuration Options dialog box, select the **Parameter** radio button.

c. In the drop-down menu for the **Parameter** type, select **DataTable**.

d. In the Location in Data Table area, select **Global sheet**. The **Name** drop-down menu changes to reflect the Global data sheet's parameters.

e. In the Name box, select the toCity parameter and click **OK**.

In the Keyword View, the **Value** cell for the **toCity** object is updated to show the parameterization:

6. **Save the test.**

   In the toolbar, click **Save**  .

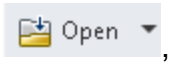
Now that you have added parameters and values, and linked your test steps with these values, you are ready to run a parameterized test. Continue to "Exercise 5d: Run a parameterized test" below

Exercise 5d: Run a parameterized test

In "Exercise 5b: Define data table parameters" on page 82 and "Exercise 5c: Add parameter values to a data table" on page 86, you created Data Table parameters for the **toCity** and **fromCity** objects in the Flight Finder action. This enables you to substitute the constant object values with changing values from the test's data table.

However, if you were to run the test at this time, it would only run one time, with the data from the first row of the Global data sheet. Since the purpose of parameterization is to see how your application runs with different sets of data, you need to instruct UFT to run the test multiple times.

In this lesson, you will configure UFT and your test to ensure that the entire test runs multiple times and uses the data in the test's Data table.

1. **Start UFT and open the Book Flights Parameter test, if necessary.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Click the **Open** down arrow 

, and select **Open Solution**. The Open Solution dialog box opens.
   c. Navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The Flight Reservation solution opens, including the Book Flights Parameter test you created in "Exercise 5a: Create a test for parameterization" on page 81.
   d. In the Solution Explorer, click the node for the **Book Flights** test. The test flow canvas opens as a separate tab in the document pane.

2. **Change the Record and Run Settings so that the flight reservation application**

does not open automatically.

In the Book Flights test (the test from which this test was created), you configured the Run and Record Settings to automatically open the flight reservation application at the beginning of the test run. For the purposes of this test run, you want UFT to open the application as part of a test step.

a. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

b. In the **Windows Applications** tab of the Record and Run Settings dialog box, select the **Record and run test on any open Windows application option** and click **Apply**.

c. Click **OK** to close the dialog box.

3. **Add additional actions for the open and close of the application.**

When you run a parameterized test, UFT runs the entire test multiple times, depending on the number of rows in your data table. However, in order to do this, we must add steps to open and close the application, in order for UFT to run the actions for each of the four application pages (**Login**, **Flight Finder**, **Select Flight**, and **Flight Details/Confirmation**.)

a. In the document pane, select the **Book Flights** tab (with the test flow canvas).

b. In the toolbar, click the **Insert Call to New Action** button  . The Insert Call to New Action dialog box opens.

c. In the Insert Call to New Action dialog box, name the new action **Open Application**. Leave all other settings and options with the default.

A new action block is added to the end of the test flow with the name **Open Application**.

d. In the Book Flights tab (with the test flow canvas), right-click the **Open Application** action and select **Move Up**. The Open Application action block moves up above the Flight Confirmation action.

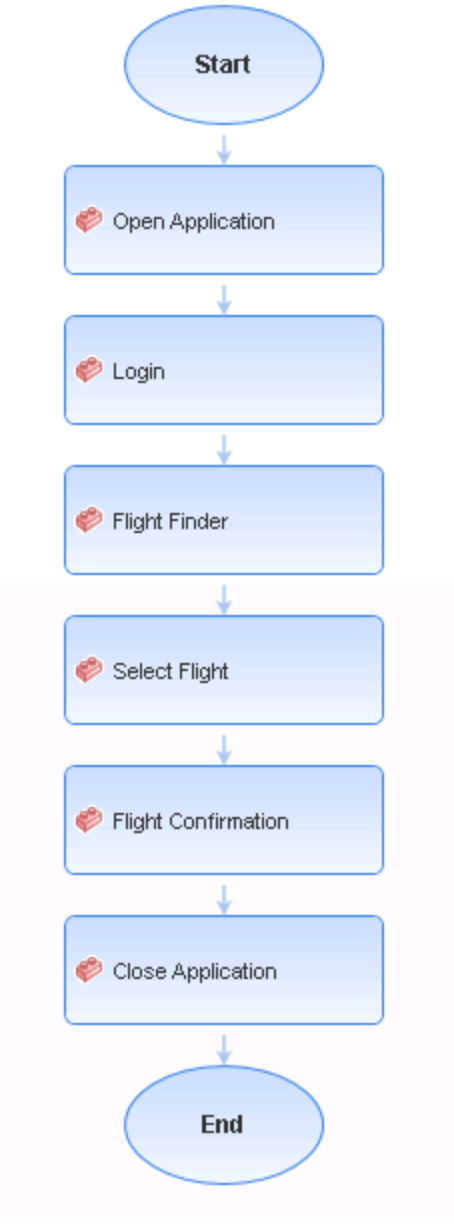
e. Right-click and select **Move Up** until the Open Application block is the first action in the test.

> **Note:** You can also drag and drop the action blocks in the test flow as needed.

f. In the toolbar, click the Insert **Call to New Action** button  again.

g. In the Insert Call to New Action dialog box, name the new action **Close Application**. Leave all other settings and options with the default.

After you have inserted this two new actions, your test flow should look like this:



4. **Add statements to open and close the application.**

   When you created the Book Flights test, you instructed UFT to open the application automatically using the Run Settings for the test. In this test, you need to add the opening and closing of the application as a separate step. To do so, you are going to use a **SystemUtil** statement.

   a. In Solution Explorer, double-click the **Open Application** action node. The Open Application action opens as a separate tab in the document pane.

b. Select **View > Editor** to open the Editor.
c. In the Editor, paste the following line:

```
SystemUtil.Run "C:\Program Files (x86)\HP\Unified Functional
Testing\samples\Flights Application\FlightsGUI.exe"
```

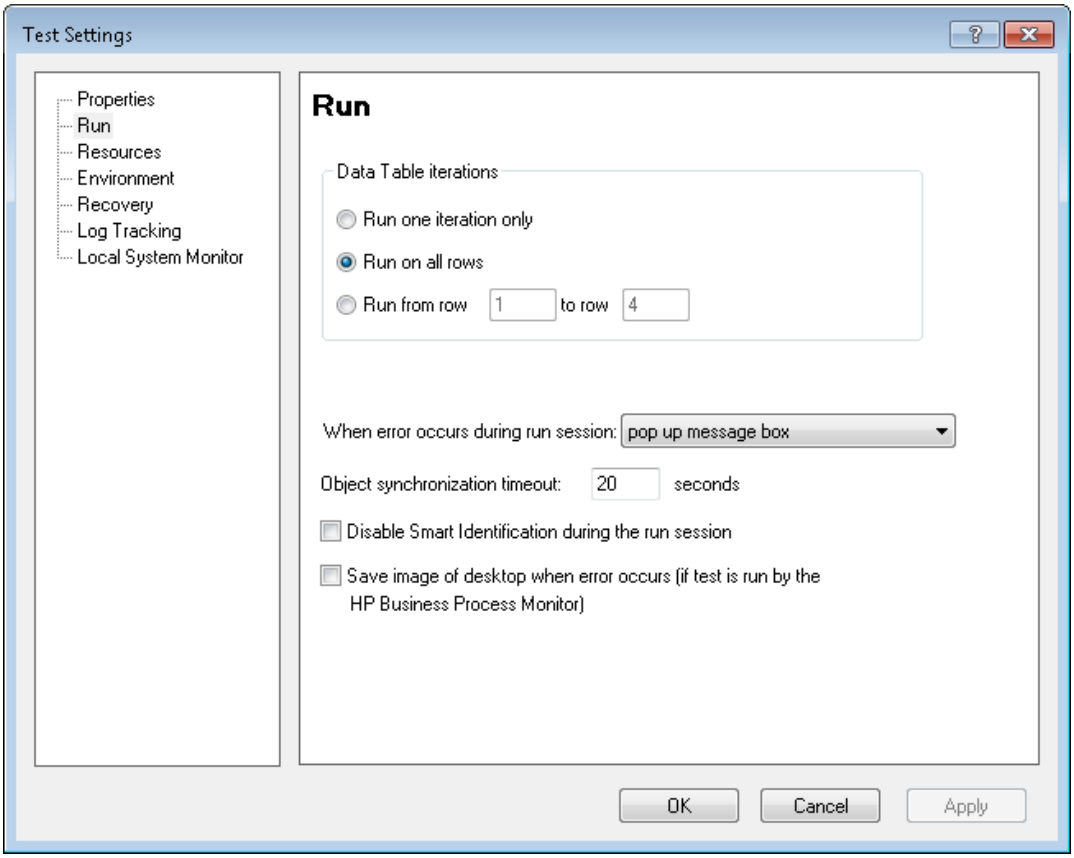
> **Note:** If you are pasting this line from the PDF copy of the tutorial, make sure to edit the pasted text so this method is all on a single line.

d. In the Solution Explorer, double-click the **Close Application** action node. The Close Application action also opens as a separate tab in the document pane.
e. In the Editor, paste the following line:

```
SystemUtil.CloseDescendentProcesses
```

5. **Instruct UFT to run an iteration for each row in the data table.**
   a. Select **File > Settings**. The Settings dialog box opens.
   b. In the Settings dialog box, select the **Run** node.

c. In the **Data Table iterations** section, select the **Run on all rows** option. This ensures that UFT runs an iteration of the test for each row in the Global data sheet.



Now, when you run your test, UFT will run multiple iterations of the test, corresponding to the four rows in the Global data sheet.

6. **Run the Book Flights Parameter test.**

   a. Click the **Run** button . The Run dialog box opens.

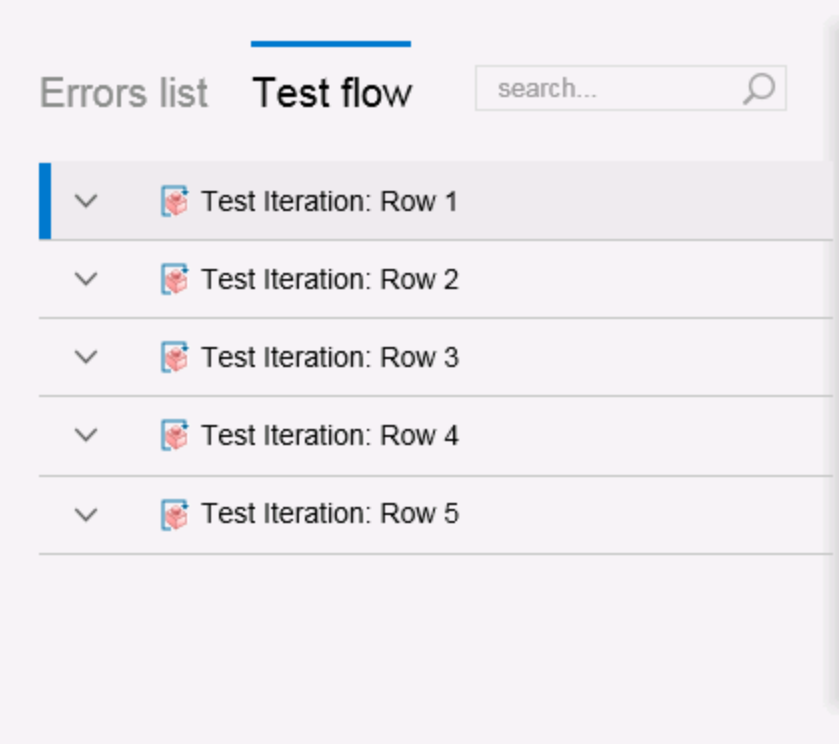
   b. In the Run dialog box, in the **Results Location** tab, select **New run results folder** and accept the default folder name.

   c. Click **OK**. When the test run is completed, the run results open.

7. **Analyze the run results.**

   In the Run Results Viewer, right-click the top node in the results tree and select **Expand All**.

Note that the results display five different nodes for each iteration of the test. This corresponds to the mulitple rows of the Global data table:



If you search under the Flight Finder Summary nodes until you see the **fromCity.Select** or **toCity.Select** steps, you will notice that the step details for the step is modified to match the values in the Data table.

8. **Close the run results.**

   In the document pane, close the tab containing the run results..

Now that you have learned how to use data to parameterize your test, continue with "Lesson 6: Creating checkpoints and output values" on the next page to learn how to enhance your tests with checkpoints and output values.

Lesson 6: Creating checkpoints and output values

In "Lesson 4: Run and analyze GUI tests" on page 73, you ran a test that you created in previous lessons, to check that a series of steps performed on the flight reservation application ran smoothly and correctly.

After you create basic test steps, one of the enhancements you can make is to add checkpoints and output values for your tests and test steps. Checkpoints verify that expected information is displayed in your application while a test is running. Output values export a value to use in other places in the test as a parameter.

In this lesson, you will insert checkpoints and use a function to check the validity of some of the objects in the flight reservation application checkpoint.

This lesson includes the following:

Understanding checkpoint and output value types

In UFT, you can insert **checkpoints** to check to see that your application is running correctly. These checkpoints run as a separate test step in the overall test flow. You use **output values** to take a value produced by a specific step or object and pass this value to another step.

Checkpoints

You can check a variety of different application objects using checkpoints:

Object Type	Description of Checkpoint	Example of Use
Standard	Checks the values of an object's properties.	Check that a radio button is selected.
Image	Checks the property value of an image. You check an image by selecting the **Standard Checkpoint** option and then selecting an image object.	Check that the image source file is correct.
Table	Checks information in a table. You check a table by selecting the **Standard Checkpoint** option and then selecting a table object.	Check that the value in a table cell is correct.
Page	Checks the characteristics of a Web page. You check a table by selecting the **Standard Checkpoint** option and then selecting a Web page in a browser.	Check how long a Web page takes to load or if a Web page contains broken links.

Text	Checks that a text string is displayed in the appropriate place in an application.	Check whether the expected text string is displayed in the expected location in a test object.
Text Area	Checks that a text string is displayed within a defined area in a Windows-based application.	Check that an area of a dialog box includes text that was entered in another part of the application.
Bitmap	Checks an area of an application after capturing it as a bitmap.	Check that a Web page (or any portion of it) is displayed as expected.
Database	Checks the contents of databases accessed by an application or Web site.	Checks that the value in a database query is correct.
Accessibility	Identifies areas of a Web site to check for Section 508 compliance.	Check if the images on a Web page include ALT properties, required by the W3C Web Content Accessibility Guidelines.
File Content	Checks the text in a document generated or accessed during a run session.	Checks that the headers in a dynamically-generated PDF display the regional corporate headquarters contact information correctly.
XML	Checks the data content of XML documents.	Check the content of an element to make sure that its tags, attributes, and values have not changed. XML file checkpoints are used to check a specified XML file; XML application checkpoints are used to check an XML document within a Web page.

Output Values

You can use a variety of different types of output values:

Type of Object	Description	Example

Standard	Takes the value from most objects in your application and stores it.	Take the string output of an edit field.
File Content	Takes the output of a selected file or part of a selected file.	Take the output of an HTML page.
Table	Takes the output of the cells or selected cells of a table object.	Take the output of the cell in row 1, column 1 in a table object.
Text/TextArea	Takes the text output of an object or an area in the application.	Take the text output of an error message.
Database	Takes the output of database cells or selected database cells	Take the output of the database accessed by an object in your application.
XML	Takes the output of elements included in an XML document.	Take the output of the <price> attribute in an XML defining prices for a product.

You can add most checkpoints and output values either while editing steps in the Keyword View or Editor or recording. The following exercises explain how to create some of the checkpoints described above.

When UFT creates a checkpoint or output values, it assigns a name based on information inside the checkpoint or output value - the checked value, for example. The checkpoint or output value name remains unchanged, even if you modify the information on which it was based. Keep this in mind when looking for checkpoints or output values displayed in the Keyword View. Note also, that UFT may shorted the name displayed in the Keyword View.

For additional details on checkpoints and output values, see the *HP Unified Functional Testing User Guide*.

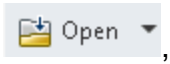
Continue with "Exercise 6a: Create a checkpoint test" on the next page to create test in which to use checkpoints.

If you would like to add output values, continue to "Exercise 6g: Create an output value test" on page 120.

Exercise 6a: Create a checkpoint test

In this exercise, you will save the Book Flights test that you ran in "Lesson 4: Run and analyze GUI tests" on page 73 as a new test to create your checkpoints.

> **Note:** Checkpoints do not need to be managed in separate tests. You are only creating a new test now for the sake of this tutorial. During your regular working process, you can add checkpoints to any test.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open it as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Click the **Open** button down arrow 

, and select **Open Solution**. The Open Solution dialog box opens.
   c. Navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.
   The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Save the test as Book Flights Checkpoint.**
   a. In the Solution Explorer, right-click the **Book Flights** test node and select **Save As**.
   b. In the Save Test As dialog box, browse to the C**:\%HOMEPATH%\My Documents\Unified Functional Testing** directory, and save the test as `Book Flight Checkpoint`.
   In the Solution Explorer, the Book Flights test is replaced with the **Book Flights Checkpoint** test. The Book Flights test is still saved separately in the file system.

3. **Add the Book Flights test back to the solution.**
   You can have both the Book Flights and Book Flights Checkpoint tests open at the same time if they are included in the same solution. This enables you to switch back and forth between them if you want to compare or edit the test.

   > **Note:** You can only run a single test at a time.

a. Select **File > Add > Existing Test**.

b. Navigate to the **Book Flights** test, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Add**.

The Book Flights test is added again to the Solution Explorer. Note that tests are listed alphabetically in the Solution Explorer.

The solution is saved automatically.

Continue with "Exercise 6b: Check object values" below to begin inserting checkpoints.

Exercise 6b: Check object values

In this exercise, you will add a standard checkpoint to the test you created in "Exercise 6a: Create a checkpoint test" on the previous page. This checkpoint verifies the value entered for the **Passenger Name** field in the Flight Details window.

> **Note:** The flight reservation application must be open to the Flight Details page before you can insert the checkpoint.

1. **Start UFT and open the Book Flights Checkpoint test.**

   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.

   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

   The Flight Reservation Application solution opens, including the Book Flights Parameter test you created in "Exercise 6a: Create a checkpoint test" on the previous page.

   c. In the Solution Explorer, double-click the **Book Flights Checkpoint** node.

   The Book Flights Checkpoint test opens as a separate tab in the document pane.

2. **Display the action in which you want to add a checkpoint.**

   In order to add a checkpoint that checks the property values of the Passenger Name edit box, after the test automatically enters the passenger's name, you must add it to the appropriate action in the test.

   In the canvas, double-click the **Flight Confirmation** action to open it.

3. **Open the flight reservation application to the Flight Details page.**

   a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.

b. Enter the login information:
   - **Username:** john
   - **Password:** hp

c. Click **OK**. The Flight Finder page opens.

d. Enter the flight search details:
   - **Departure City:** Los Angeles
   - **Arrival City:** Sydney
   - **Date:** tomorrow's date
   - **Class:** Business
   - **Tickets:** 2

e. Click the **Find Flights** button. The Select Flight page opens.

f. In the Select Flight page, select the first row and click **Select Flight**. The Flight Details page opens.

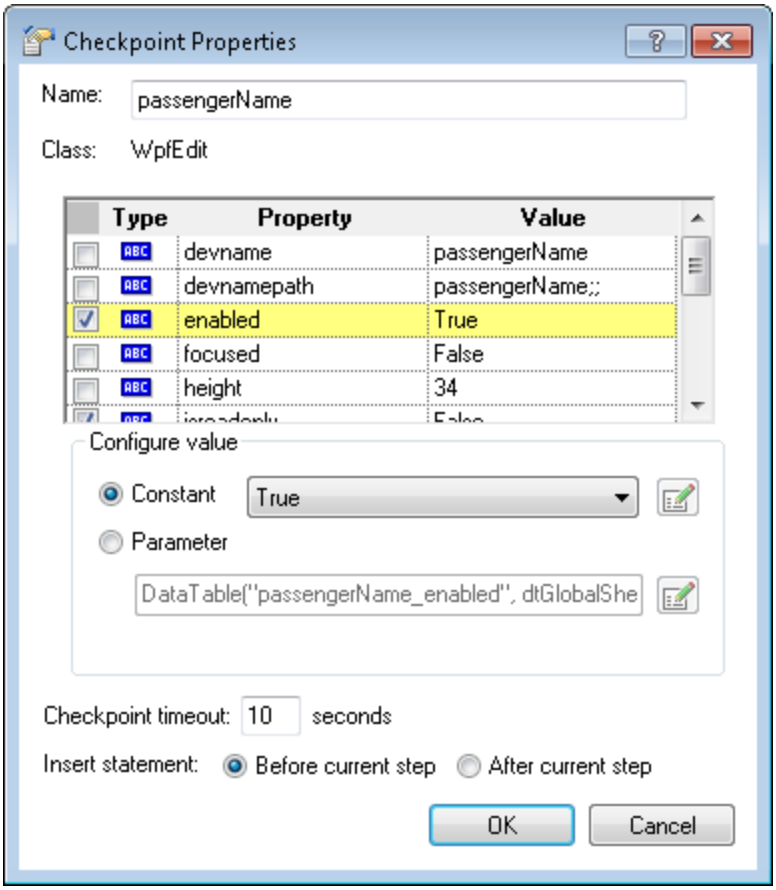
4. **Create a standard checkpoint.**

a. If the Editor is displayed, click the **Keyword View** button  to display the Keyword View.

b. In the Keyword View, select the **passengerName** row by clicking in the right margin of the grid.

> **Note:** Do not click in the **Item** column - this selects the object only. You need the entire step to be selected to add a checkpoint.

c. Select **Design > Checkpoint > Standard Checkpoint**. The Checkpoint Properties dialog box opens:



This dialog box displays the passengerName object properties:

- The **Name** is the name of the object as defined in the application. In this case, the name is **passengerName**.
- The **Class** is the type of object. In this case, the type is **WpfEdit**, meaning that the type of object is an edit box.
- The **ABC** icon in the **Type** column indicates that the value of the property is a constant.

  When you insert a checkpoint, UFT recommends default property checks for each object class:

Property	Value	Explanation
enabled	True	This checks whether the object is currently enabled.

isreadonly	False	This checks whether you can enter information into the edit box. Currently, the object is set to allow entry of a text string.
text	No default specified	This checks the text that is entered in the object. Currently the value is empty. You need to enter the same value that you specified for the **passengerName** edit box in the first step of this action.

d. In the **Name** box of the Checkpoint Properties dialog box, enter **CheckName** as the new checkpoint name.

e. Scroll down in the object properties area and select the row containing the property name **text**. The row turns yellow to show that you have selected this row.

f. In the **text** property row, click in the **Value** column.

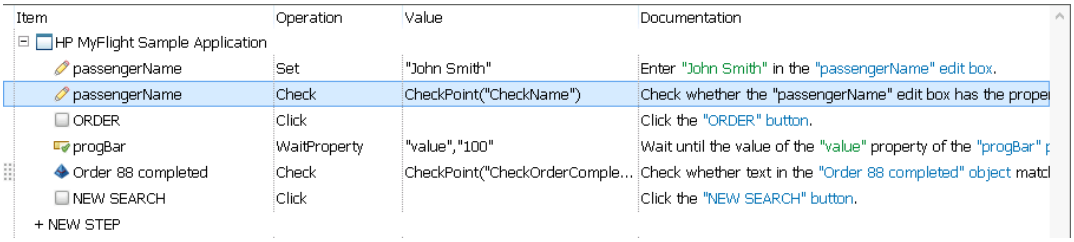
g. In the **Configure value** area below the object properties grid, click the **Constant** radio button.

h. In the **Constant** value edit box, enter John Smith. (This is the name of the value you entered in the Passenger Name box in the first step of the action.) Note that the object properties grid is also updated with this value.

i. In the Insert statement area at the bottom of the Checkpoint Properties dialog box, select **After current step**. This insert the checkpoint after the **passengerName Set** step.

j. Accept the rest of the settings as default and click **OK**.

UFT adds a standard checkpoint step to your test below the selected step:



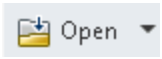
5. **Save the test.**

In the toolbar, click **Save**  .

Using this process, you can insert many different types of checkpoints. Continue with "Exercise 6c: Check table values" on the next page to learn how to check table objects in your application.

Exercise 6c: Check table values

In "Exercise 6b: Check object values" on page 100, you added a checkpoint for an object in your application. In this exercise, you will add a table checkpoint to your test. The table checkpoint will check a value in the flights grid on the Select Flights page.

1. **Start UFT and open the Book Flights Checkpoint test.**
   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Click the Open button down arrow 

, and select Open Solution. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing** and click **Open**.
   The Flight Reservation Application solution opens, including the Book Flights Parameter test you created in "Exercise 6a: Create a checkpoint test" on page 99.
   d. In the Solution Explorer, double-click the **Book Flights Checkpoint** node.

2. **Locate the step where you want to add a table checkpoint.**
   a. If the Select Flight action is not already open, in the Solution Explorer, double-click the **Select Flight** action node. The action is displayed as a separate tab in the document pane.
   b. If the Editor is displayed, select **View > Keyword View** to show the Keyword View.
   c. In the Keyword View, select the **flightsDataGrid** step (the step that selects the flight to book).

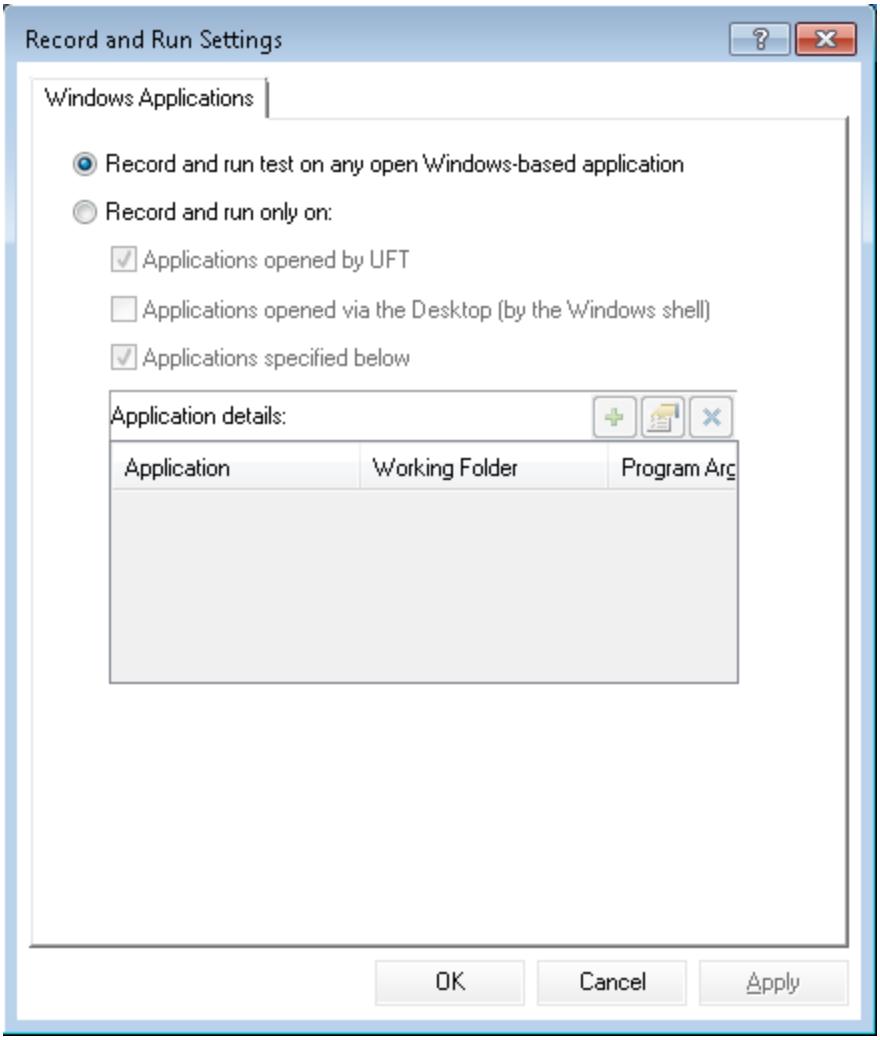
3. **Open the flight reservation application to the Select Flight page.**
   a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.
   b. Enter the login information:
      - **Username:** john
      - **Password:** hp
   c. Click **OK**. The Flight Finder page opens.

d. Enter the flight search details:
   - **Departure City:** Los Angeles
   - **Arrival City:** Sydney
   - **Date:** tomorrow's date
   - **Class:** Business
   - **Tickets:** 2

e. Click the **Find Flights** button. The Select Flight page opens.

4. **Configure UFT to record on the open application page.**

a. In UFT, select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens:



b. In the Windows Applications tab, select the **Record and run test on any open Windows-based application** option.

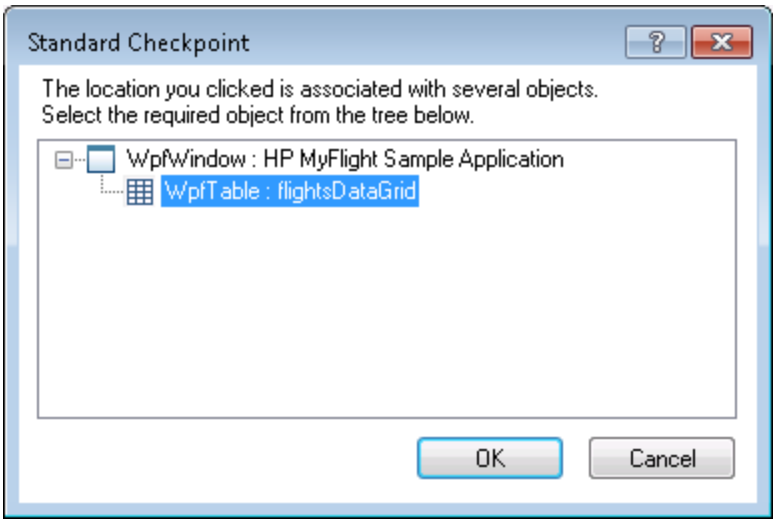
c. Click **OK** to close the dialog box.

5. **Create a table checkpoint.**

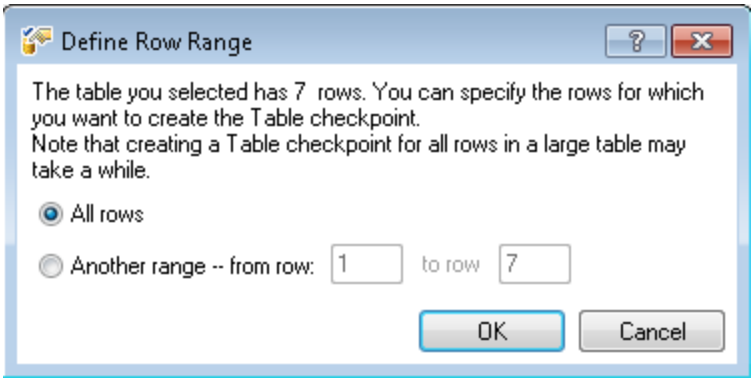
   a. In the toolbar, click the **Record** button  . UFT begins a recording session and the main UFT window is hidden.

   b. In the Record Toolbar, click the **Insert Checkpoint or Output Value** drop-down arrow  and select **Standard Checkpoint**. The mouse pointer turns into a pointing hand.

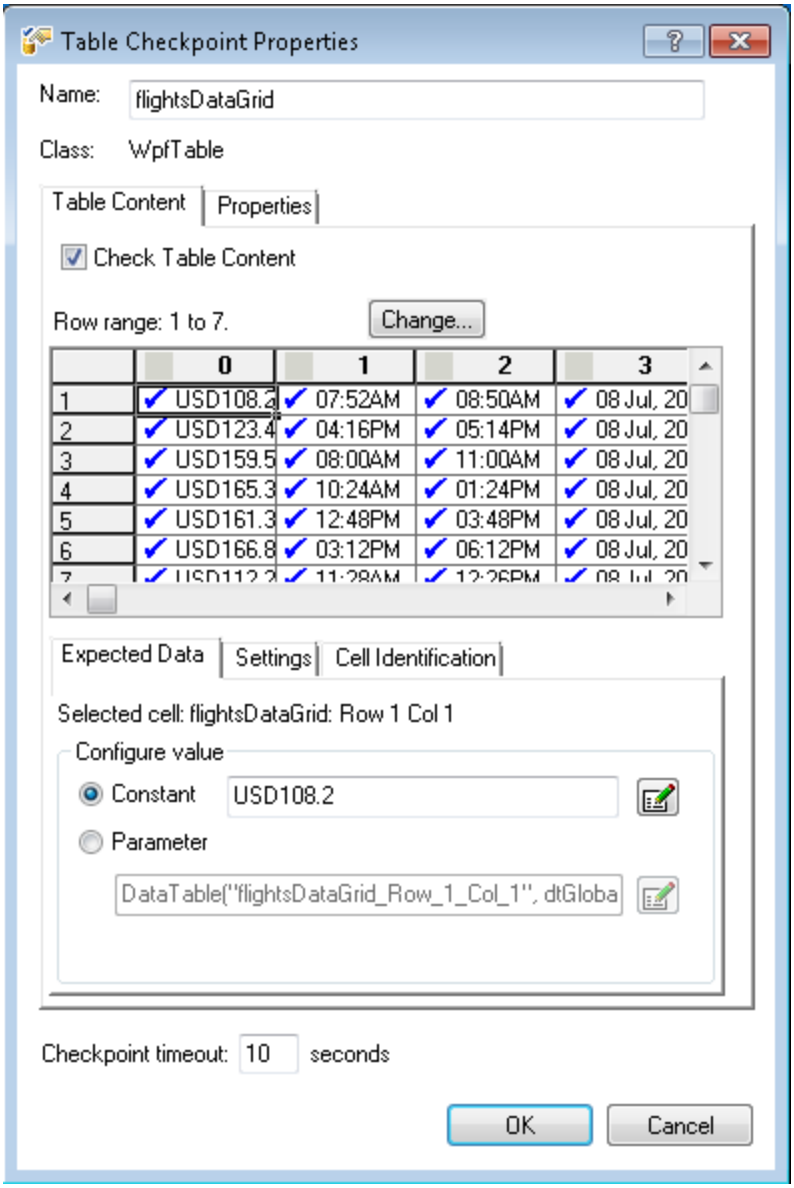
   c. In the flight reservation application, click the flights table. The Standard Checkpoint object selection dialog box opens:



   d. In the Standard Checkpoint selection dialog box, select the **WpfTable: flightsDataGrid** object and click **OK**. The Define Row Range dialog box opens:



e. In the Define Row Range dialog box, select the **All rows** radio button and click **OK**. The Table Checkpoint Properties dialog box opens:



Note that by default, check marks appear in all cells. You can double-click a cell to select or clear the cell selection, or double-click a row or column header to select or clear the selection for all the cells in that row or column.

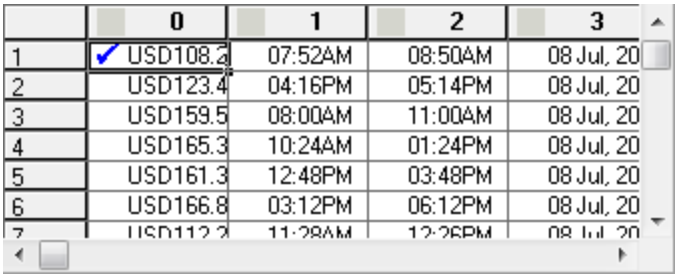
f. In the Table Checkpoint Properties dialog box, enter **CheckCost** as the new checkpoint name in the **Name** box.

g. In the grid, double-click each column header to clear the check marks.

> **Note:** You will need to scroll to the right in the grid to view all the table

> object columns.

h. In the grid, double-click row **1**, column **0** to select this cell. (UFT checks only those cells containing check marks.)



> **Note:** The data displayed in the table is date-sensitive. If you create this checkpoint on one day, but return to run this test on a different day, you need to update this checkpoint using the steps above to ensure the checkpoint passes.

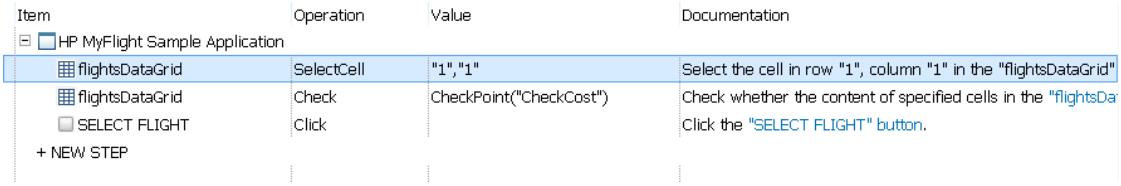
i. Scroll through the rows and columns to make sure that only the cell in row **1**, column **0** is checked. If any other cells are checked, double-click them to remove the check.

j. Accept the rest of the settings as default and click **OK**.

6. **Stop the recording session.**

   In the Record Toolbar, click **Stop**  .

   After you defined the table object's checkpoint properties, UFT added a table checkpoint step to your test. It is displayed in the Keyword View as a new step under the flightsDataGrid object step:



7. **Save the test.**

   In the toolbar, click **Save**  .

Now that you have added a checkpoint for a table object, continue to add checkpoints in "Exercise 6d: Check text values" on the next page.

Exercise 6d: Check text values

In the previous exercises, you added checkpoints to a regular test object and a table object. In this object, you will add a text checkpoint to your test to check the text inside an object that appears at the end of the order process.

1. **Start UFT and open the Book Flights Checkpoint test.**
   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the Recent Solutions area, click the **Flight Reservation Application** solution.
      The Flight Reservation Application solution opens, including the Book Flights Parameter test you created in "Exercise 6a: Create a checkpoint test" on page 99.
   c. In the Solution Explorer, double-click the **Book Flights Checkpoint** node.
      The Book Flights Checkpoint test opens as a separate tab in the document pane.

2. **Locate the step where you want to add a text checkpoint.**
   a. In the Solution Explorer, double-click the **Flight Confirmation** action node. The Flight Confirmation action opens as a separate tab in the document pane.
   b. If the Editor is open, click the **Keyword View** button  to display the Keyword View.
   c. In the Keyword View, highlight the **progBar** step (in the next-to-last row, if you have all of the steps fully expanded).

3. **Open the flight reservation application to the Flight Details page.**
   a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.
   b. Enter the login information:
      - **Username:** john
      - **Password:** hp
   c. Click **OK**. The Flight Finder page opens.

d. Enter the flight search details:
   - **Departure City:** Los Angeles
   - **Arrival City:** Sydney
   - **Date:** tomorrow's date
   - **Class:** Business
   - **Tickets:**2

e. Click the **Find Flights** button. The Select Flight page opens.

f. In the Select Flight page, select the first row and click **Select Flight**. The Flight Details page opens.

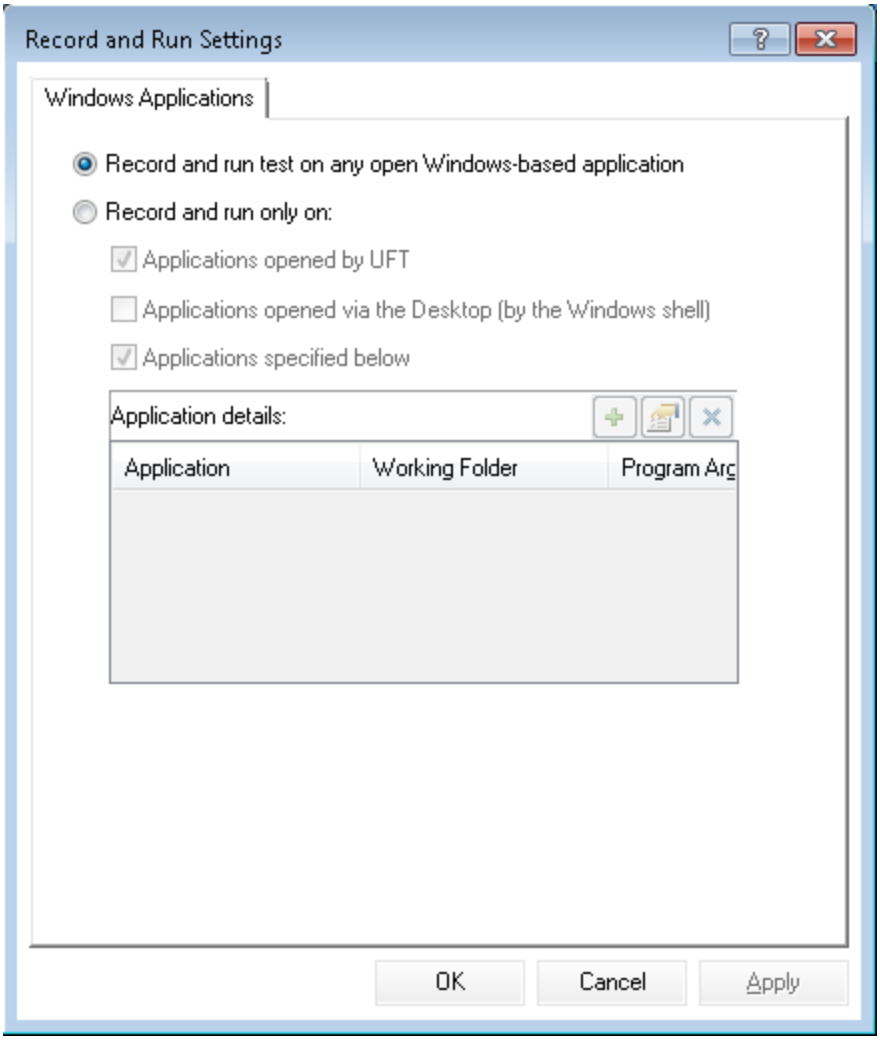
g. In the Flight Details page, in the **Passenger Name** box, enter **John Smith** and click **ORDER**.

   A box is displayed in the middle of the window informing you of the completion of the order. Leave the application like this.

4. **Configure UFT to record on the open application page.**

   a. In UFT, select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

b. In the Windows Applications tab, confirm that the **Record and run test on any open Windows-based application** is selected:



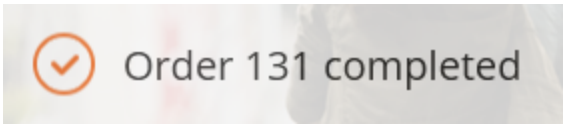
c. Click **OK** to close the dialog box.

5. **Create a text checkpoint.**

a. In the toolbar, click the **Record** button  . The UFT window is hidden, and the Record Toolbar opens at the top of the window.

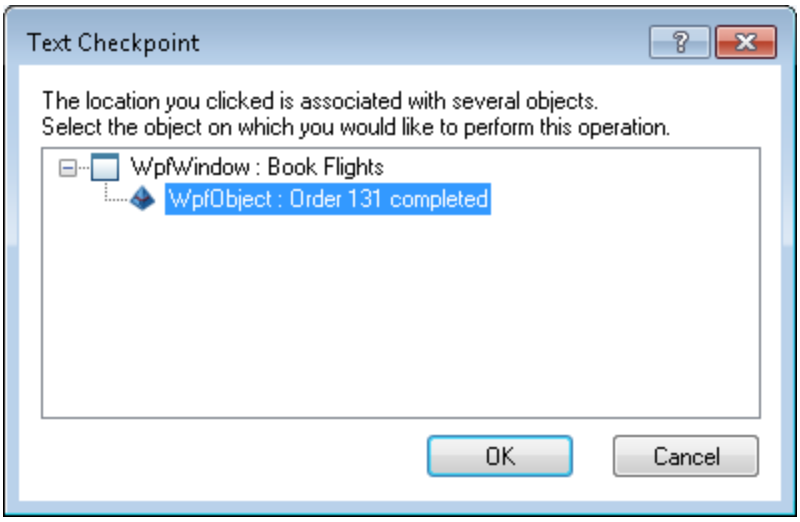
b. In the Record Toolbar, click the **Insert Checkpoint or Output Value** button  and select **Text Checkpoint**. The mouse pointer changes into a pointing hand.

c. In the Flight Details window in the flight reservation application, click the Order # Completed graphic in the middle of the Flight Details window:

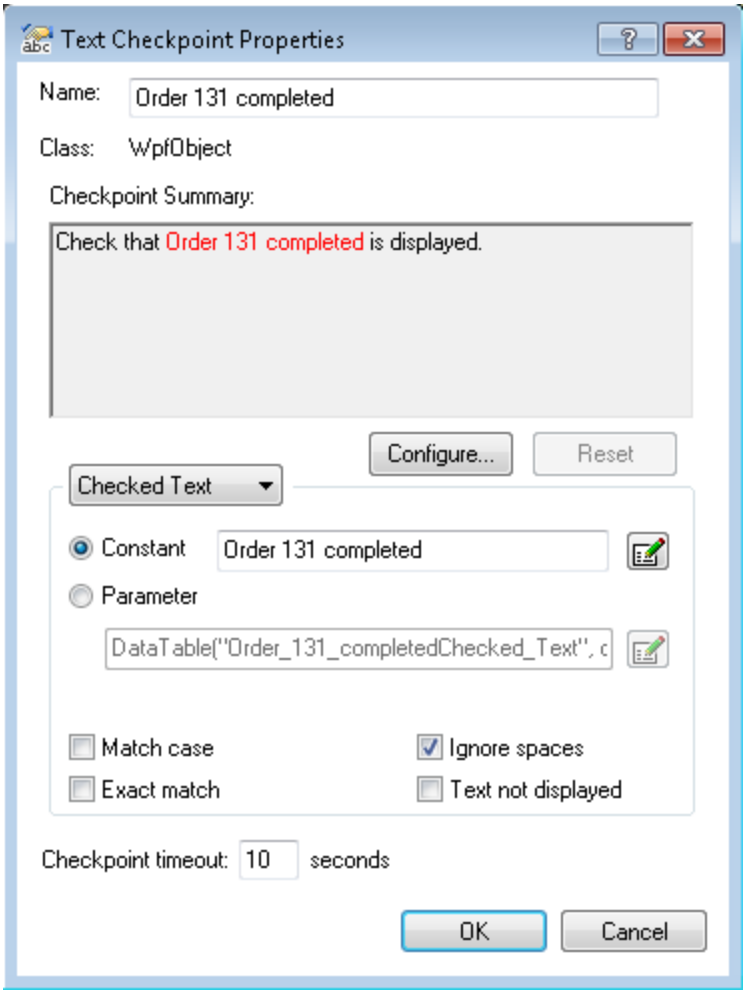


> **Note:** The order number may differ depending on if you have previously run the flight reservation application.

The Text Checkpoint object selection dialog box opens:

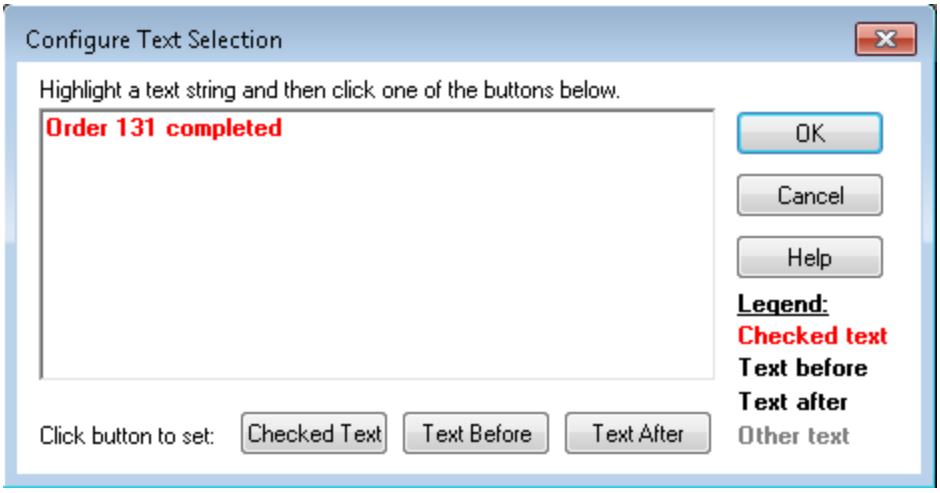


d. In the Text Checkpoint object selection dialog box, select the **WpfObject: Order # Completed** object and click **OK**. The Text Checkpoint Properties dialog box opens:

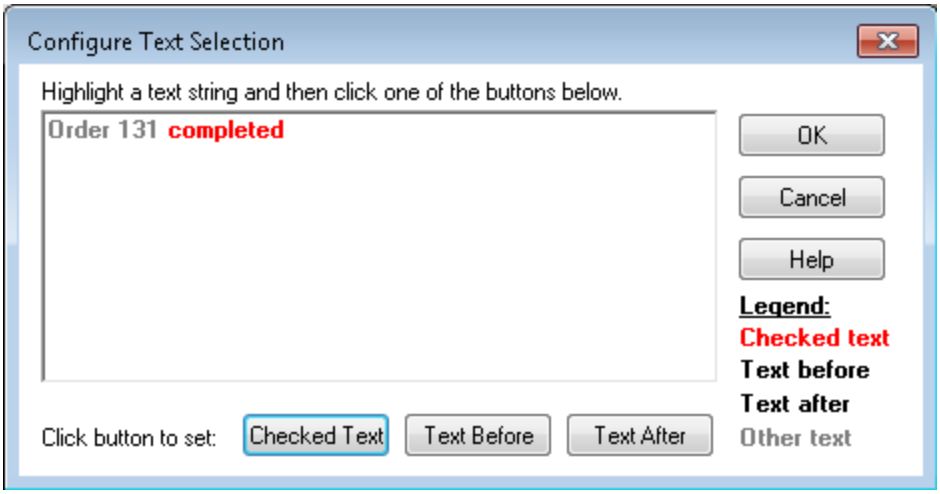


e. In the **Name** box in the Text Checkpoint Properties dialog box, enter **CheckOrderCompletedText** as the new checkpoint name.

f. Under the **Checkpoint Summary** area, click the **Configure** button. The Configure Text Selection dialog box opens:

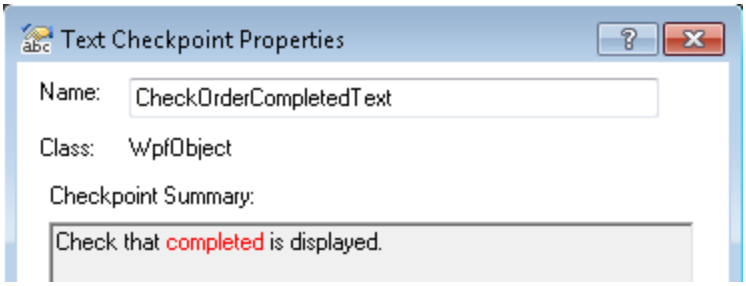


g. In the Configure Text Selection dialog box, highlight the string **completed** and click **Checked Text**. The **Order #** text string changes from red to gray:



h. Click **OK** to close the Configure Text Selection dialog box.

In the Text Checkpoint Properties dialog box, the Checkpoint Summary Area is updated to reflect your selection:

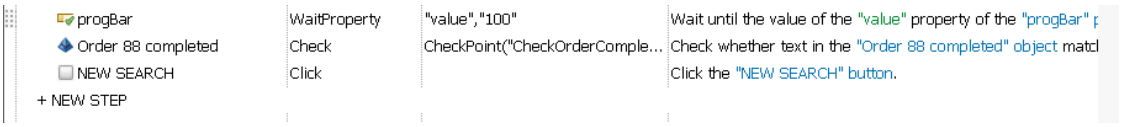


i. Accept the rest of the settings as the default and click **OK**.

6. **Stop the recording session.**

   On the Record Toolbar, click **Stop** to stop recording.

   UFT adds the step with the text checkpoint to your test, below the step containing the **progBar** object. It is displayed in the Keyword View as a checkpoint operation on the **Order # Completed** object:

   In the Editor, the statement looks like this:

   ```
   WpfWindow("Book Flights").WpfObject("Order 89 completed").Check CheckPoint
   ("CheckOrderCompletedText")
   ```

7. **Save the test.**

   Click **Save**.

Now that you have added a couple of different types of checkpoints, learn more about checkpoint management in "Exercise 6e: Manage checkpoints in the object repository" below.

Exercise 6e: Manage checkpoints in the object repository

In the previous exercises, you added a number of different types of checkpoints in your actions. In addition to working with and viewing checkpoints in the context of a specific action, you can also view them in the object repository and modify their properties.

By modifying them in an object repository, you can use the same checkpoint in more than one location in your test. For example, if you want to verify that your organization's logo appears on every page of your application, you can create a checkpoint and insert it in different actions or places in the test.

For the purposes of this exercise, you will not be reusing checkpoints.

1. **Start UFT and open the Book Flights Checkpoint test.**
   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.

c. In the Open Solution dialog box, navigate to the **Flight Reservation Application** solution, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

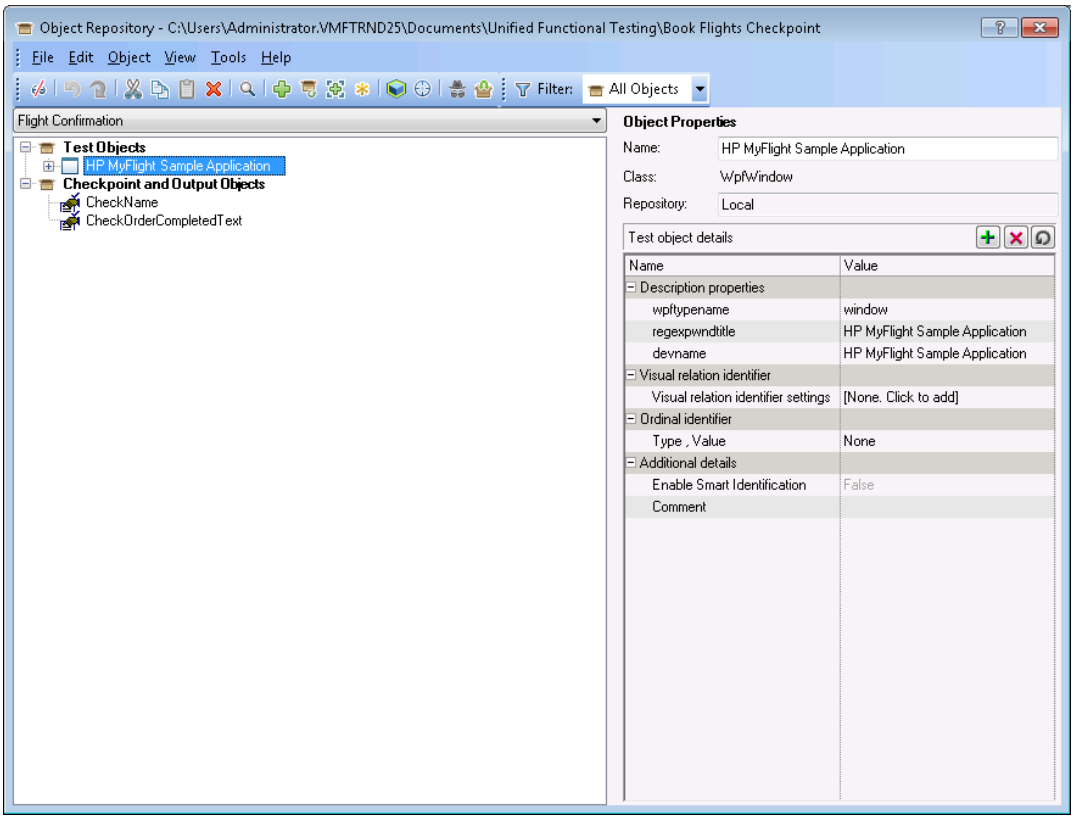
The Flight Reservation Application solution opens, including the Book Flights Parameter test you created in "Exercise 6a: Create a checkpoint test" on page 99.

2. **Open the Object Repository Window.**

a. In the Solution Explorer, double-click the **Flight Confirmation** action. The Flight Confirmation action opens as a separate tab in the document pane.

b. In the toolbar, click the **Object Repository** button  . The Object Repository window opens, displaying a tree of all test objects and all checkpoint and output objects in the current action.

The tree includes all local objects and all objects in any shared object repositories associated with the action:



3. **Select an action to view its checkpoints.**

a. In the action drop-down menu directly above the object tree, select an action to display its test objects, checkpoint objects, and output value objects.

b. Close the Object Repository window when you are done.

> **Note:** For the purposes of this exercise, you do not need to modify any object or checkpoint properties.

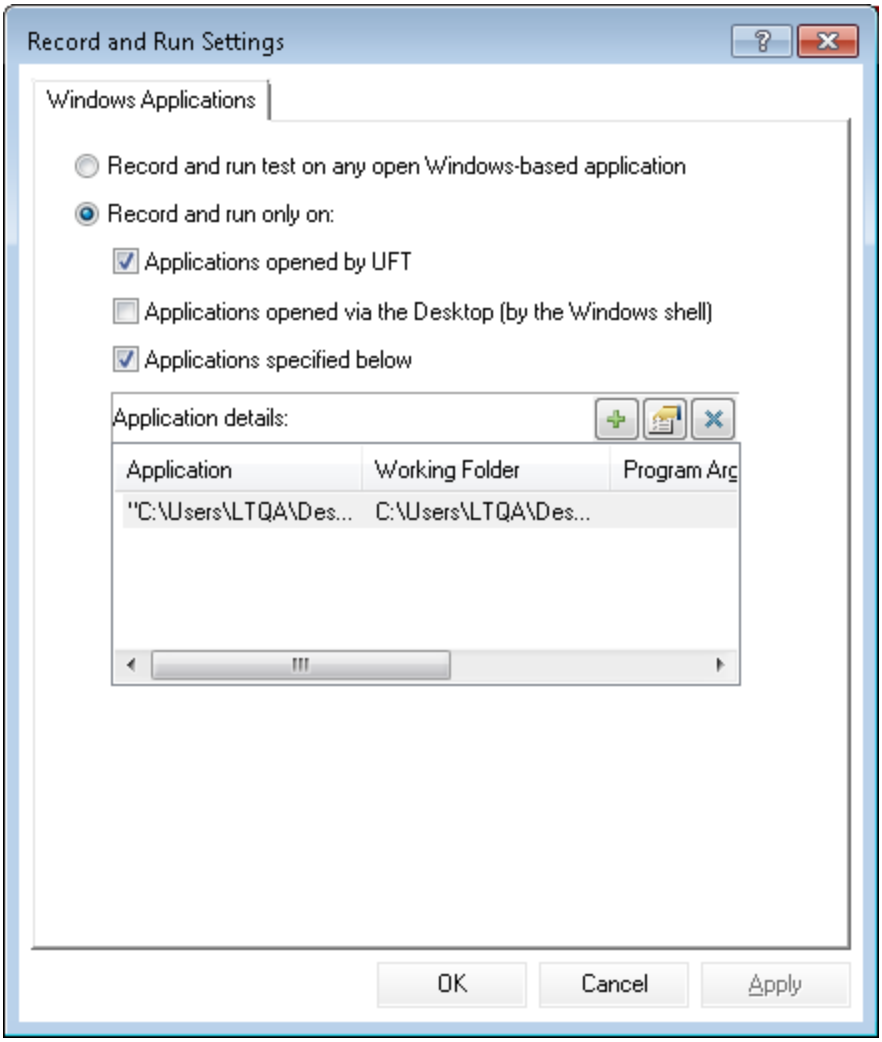
You are now ready to run the test with your checkpoints. Continue with "Exercise 6f: Run and analyze a test with checkpoints" below to learn about running and viewing run results for a test containing checkpoints.

Exercise 6f: Run and analyze a test with checkpoints

Now that you have created a test using checkpoints, you should run the test to see how the checkpoints perform. In this exercise, you will run the test and analyze the checkpoint results.

1. **Configure UFT to open the flight reservation application.**
   a. In UFT, select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

b. In the Windows Applications tab, select the **Record and run only on:** option:



> **Note:** The Application details should still be saved, as you set them in "Lesson 4: Run and analyze GUI tests" on page 73.

c. Click **OK** to close the dialog box.

2. **Start running your test.**

a. In the toolbar, click the **Run** button . The Run dialog box opens.

b. In the **Results Location** tab, ensure that the **New run results folder** is selected. Accept the default results folder name.

c. Click **OK**.

UFT opens the flight reservation application and performs the steps. At the end of the test run, the run results open.

3. **View the run results.**

   When the run results are displayed, the run results should be **Passed**, indicating that all the checkpoints passed. If one or more of the checkpoints had failed, the test run is listed as **Failed**.

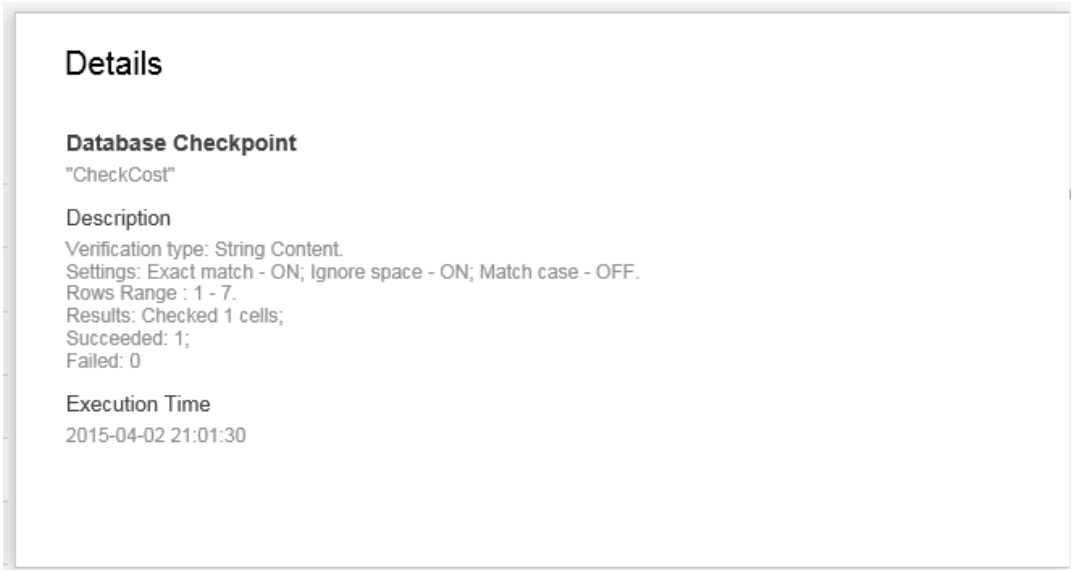
4. **View the results of the standard checkpoint.**
   a. In the Test Flow, find the **Flight Confirmation** node.
   b. Under the Flight Confirmation node, under the **passengerName.Set** node, select the **Standard Checkpoint: "CheckName"** node.

      The summary displays the details of the standard checkpoint, including the properties that were checked and their values. The checkpoint passed because the actual values of the object properties matched the expected values:

      s

5. **View the results of the table checkpoint.**
   a. In the results tree, expand the **Select Flight** node.
   b. Under the Action: Select Flight node, find the **Check Cost** node.
   c. Expand the Check Cost node and select the **Standard Checkpoint: CheckCost** node.

      The summary displays the details of the table, checkpoint:

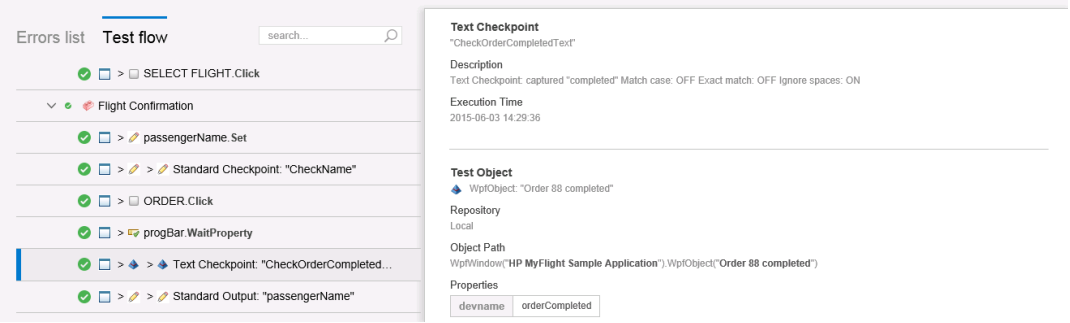
      In some cases, the step summary will also display details about the table checkpoint, including the captured data from the table object.

6. **View the results of the text checkpoint.**

a. In the Test Flow, find the **Flight Confirmation** node.

b. Under the Flight Confirmation node, under the **progBar.Wait** step, select the **TExt Checkpoint: CheckOrderCompleted** node.

The step summary displays the details of the checkpoint. The checkpoint passed because the actual text matches the expected text:



7. **Close the run results.**

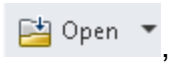
In the document pane, close the tab containing the run results.

Exercise 6g: Create an output value test

In this exercise, you will create a test in which you will add an output value step. This test is based on the Book Flights test you created in "Lesson 1: Create a GUI test and actions" on page 29.

1. **Start UFT and open the Book Flights test.**

a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.

b. Click the **Open** button down arrow 

, and select **Open Solution**. The Open Solution dialog box opens.

c. Navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Save the test as Book Flights Output Value.**

a. In the Solution Explorer, select the **Book Flights** test node, and then select **File > Save As**.

b. In the Save Test As dialog box, browse to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** directory, and save the test as **Book Flights Output Value**.

   In the Solution Explorer, the Book Flights test is replaced by the new Book Flights Output Value test. The original Book Flights test is still saved in the file system.

3. **Add the Book Flights test back to the solution.**

   You can have both the **Book Flights** and the **Book Flights Output Value** tests open at the same time if they are included in the same solution. This enables you to switch back and forth between them if you want to compare or edit tests.

   > **Note:** You can only run a single test at a time.

   a. Select **File > Add > Existing Test**.

   b. In the Add Existing Test dialog box, navigate to the Book Flights test, stored in **C:%\HOMEPATH%\My Documents\Unified Functional Testing**, and click **Add**.

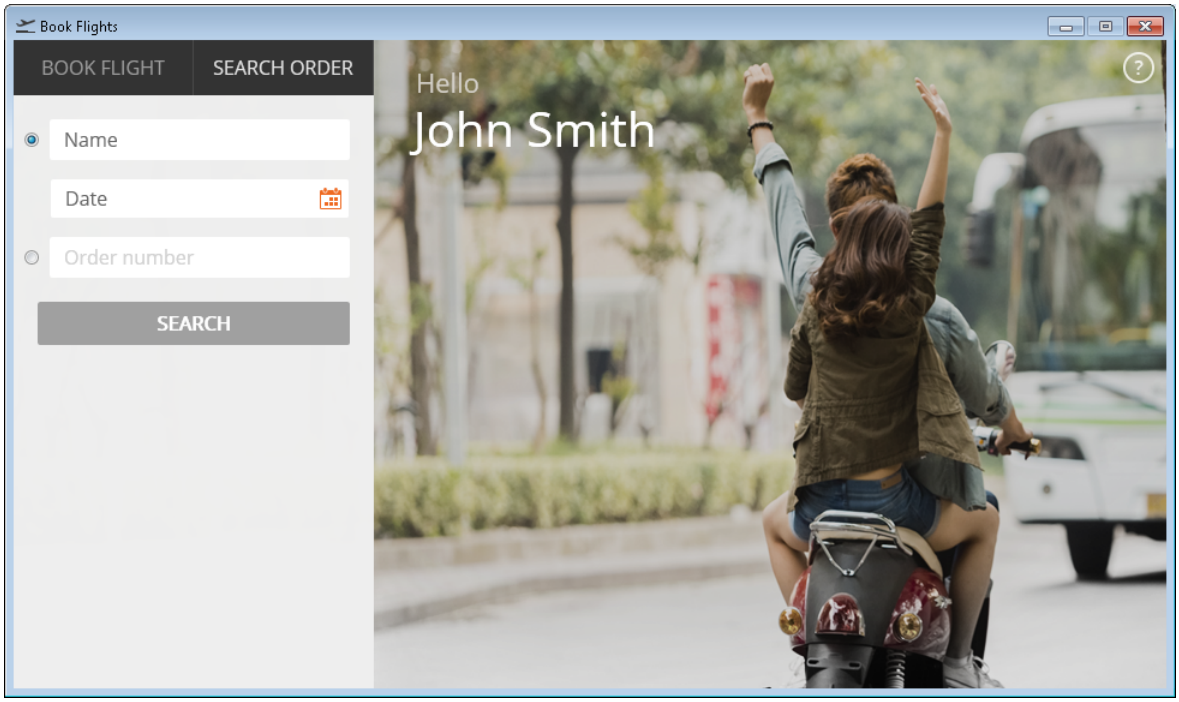
      The Book Flights test is again displayed as a separate node in the Solution Explorer.

   Now that you have created a test in which to add an output value, continue to "Exercise 6h: Add an output value step" below to add an output value step to a test.

Exercise 6h: Add an output value step

When you created the original Book Flights test, you created an action for the main application flow, where you logged into the application, entered the departure and arrival details, selected a flight, and booked the flight for a specific customer.

In the flight reservation application, there is an additional area of the application that enables you to search for all the previously created flight orders:



In this exercise, you will create an output value step that takes the output of a step in the Flight Details page (in the Flight Confirmation action of the test, and uses this output as the parameter for an object in the Search page.

1. **Create an action for the test steps on the search page.**
   a. In the Solution Explorer, click the **Book Flights** test node. The test flow canvas opens as a separate tab in the document pane.
   b. In the toolbar, click the **Insert Call to New Action** button . The Insert Call to New Action dialog box opens.
   c. In the Insert Call to New Action dialog box, enter the name of the new action as `Flight Order Search`.
   d. Leave the other settings as default and click **OK**.
      A new action block called **Flight Order Search** is added to the canvas at the end of the test flow, and the Flight Order Search action opens as a separate tab in the document pane.

2. **Create object repositories for the other application pages.**
   When you created the Book Flights test, you created object repositories only for the main application pages. In order to create test steps for the Search pages, you need to create additional object repositories for the search pages.

a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.
b. Enter the login credentials for the application:
   - **Username:** john
   - **Password:** hp
c. Click **OK** to log in. The Flight Finder page opens.
d. In the Flight Finder page, in the upper-left corner of the window, click the **Search Order** button. The Search Details page opens.
e. In UFT, select **Resources > Object Repository Manager**. The Object Repository Manager window opens.
f. In, the Object Repository Manager, use the Navigate and Learn process as described in "Exercise 2b: Create object repositories using Navigate and Learn" on page 42.
g. After you learn all the objects in this page, click **File > Save**.
h. Navigate to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Object Repositories** folder, and save the object repository with the name **Search**.
i. In the flight reservation application, in the Search Details page, enter **John Smith** in the **Name** box and click **Search**. A list of flights reserved for John Smith is displayed in a separate page.
j. In UFT, open the Object Repository Manager window again.
k. Use the Navigate and Learn process on the Select Order page to learn the objects for this page.
l. After you learn all the objects in the Select Order page, click **File > Save**.
m. In the Save Object Repository dialog box, navigate again to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Object Repositories** folder, and save the object repository with the name **Search_Results**.
n. Close the Object Repository Manager.

3. **Associate the new object repositories with the Flight Order Search action.**
   a. In UFT main window, in the Solution Explorer, under the Book Flights Output Value test node, right-click the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** action node and select **Associate Repository with Action**.
   b. In the Open Shared Object Repository window, navigate to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Object Repositories** folder and select the **Search.tsr** file.
   c. Click **Open** to associate the object repository.

d. Repeat the process to associate the Search Results.tsr object repository (also stored in the **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Object Repositories** folder).

Both the **Search.tsr** and **Search Results.tsr** object repository files are displayed as sub-nodes of the Flight Order Search action.

4. **Add steps to use the Search and Flight Orders pages.**
   a. In the Solution Explorer, double-click the **Flight Order Search** action node. The Flight Order Search action is displayed as a separate tab in the document pane.
   b. If the Keyword View is displayed, select **View > Editor** to display the Editor.
   c. In the Editor, paste the following lines:

```
WpfWindow("HP MyFlight Sample Application").WpfTabStrip
("WpfTabStrip").Select "SEARCH ORDER"
WpfWindow("HP MyFlight Sample Application").WpfEdit("byName").Set "John
Smith"
WpfWindow("HP MyFlight Sample Application").WpfButton("SEARCH").Click
WpfWindow("HP MyFlight Sample Application").WpfTable
("ordersDataGrid").SelectCell 1, 1
WpfWindow("HP MyFlight Sample Application").WpfButton("SELECT
ORDER").Click
```

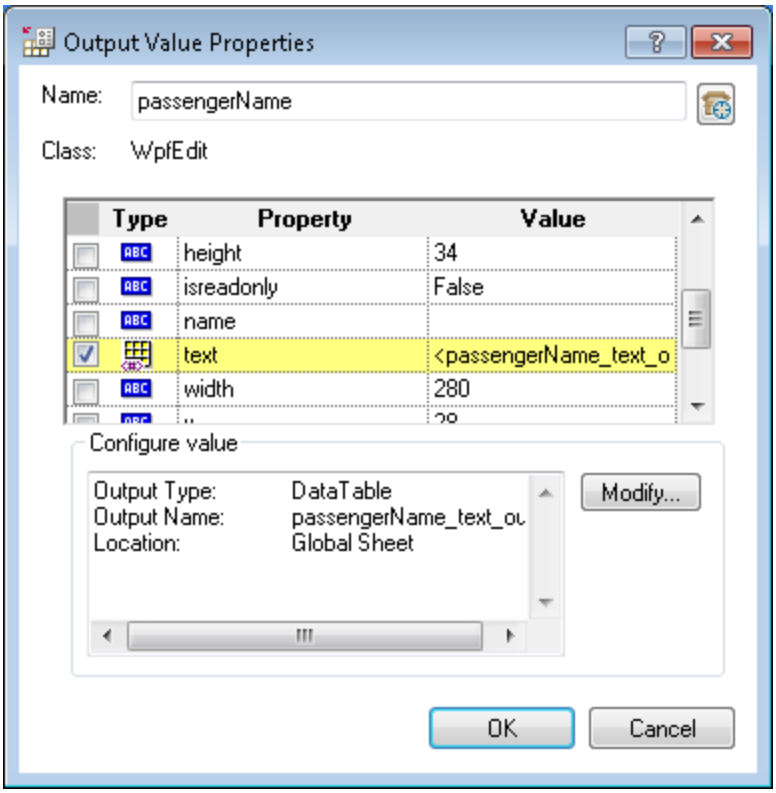
5. **Open the flight reservation application to the Flight Details page.**
   a. In the Search Results page of the flight reservation application, click the **BACK** button. The Search Details page is displayed.
   b. In the Search Details page, in the upper left hand corner, click the Book Flight button. The Flight Finder page opens.
   c. In the Flight Finder page, enter the flight details:
      - **Departure City:** Los Angeles
      - **Arrival City:** Sydney
      - **Date:** tomorrow's date
      - **Class:** Business
      - **Tickets:** 2
   d. Click the **Find Flights** button. The Select Flight page opens.
   e. In the Select Flight page, select the first row and click **Select Flight**. The Flight Details page opens.

6. **Add an output value step to the Flight Confirmation action.**
   a. In UFT, in the Solution Explorer, under the Book Flights test node, double-

click the **Flight Confirmation** action node. The Flight Confirmation action opens as a separate tab in the document pane.

b. If the Editor is displayed, select **View > Keyword View** to display the Keyword View.

c. In the Keyword View, right-click the byName step and select Insert Output Value. The Output Value Properties dialog box opens:

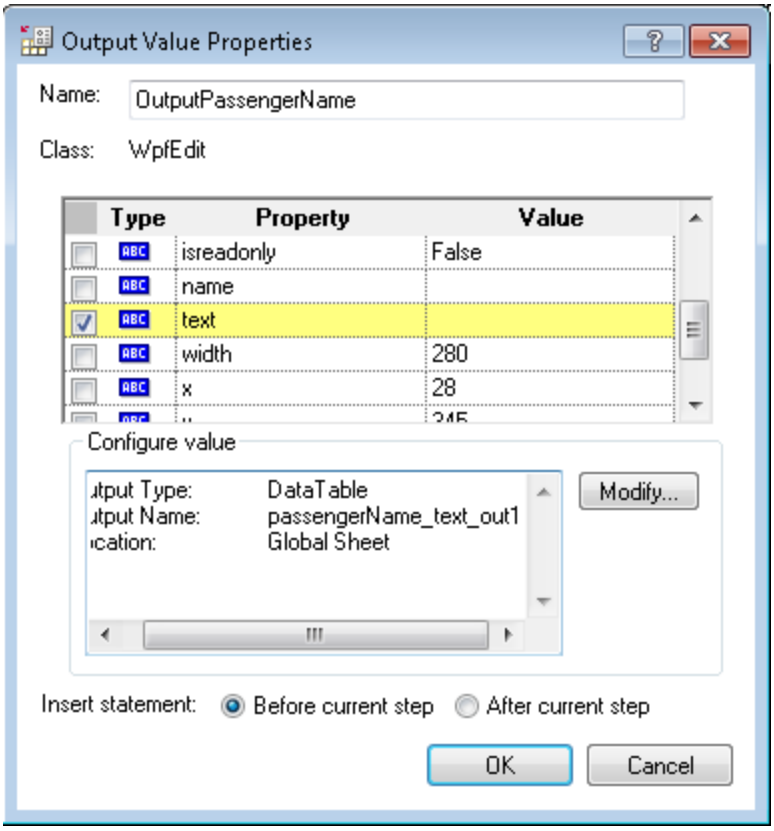


The dialog box displays the properties on which you are inserting the output value:

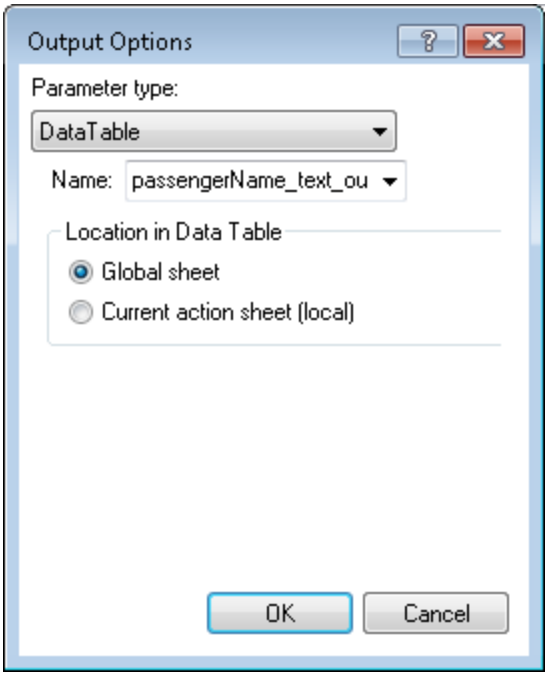
- The **Name** is the name of the object as defined in the application, in this case **byName**.
- The **Class** is the type of object, in this case **WpfEdit**, indicating that the object is an edit box.
- The **ABC** icon in the Type column indicates that the value of the property is a constant.
- The grid lists the object properties you can choose to output, including the **Property** name and the **Value** of the object that can be sent as output.

d. In the **Name** box, enter **OutputPassengerName** as the new output value name.

e. In the object properties grid, scroll through the properties and select the **text** property row. Note that there is no value provided for this property.



f. Below the properties grid, click the **Modify** button. The Output Options dialog box opens. (This dialog enables you to determine where to store the output of this test step.)



When you create an output value, you have the choice of places to store the output value:

- DataTable parameter
- Test/action parameter (if you have created one)
- Environment variable
- Component parameter (if you have created one)

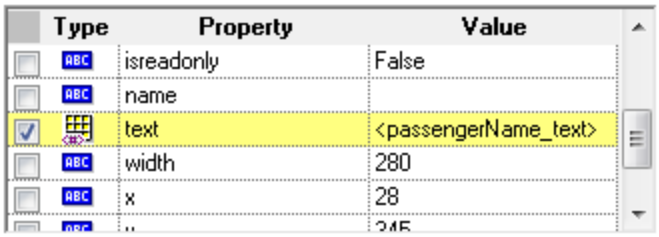
For the purpose of this tutorial, we will save the output value in the Data Table.

g. In the **Parameter type** drop-down menu, select **DataTable**. UFT updates the fields of the dialog box, and suggests a default name for the parameter.

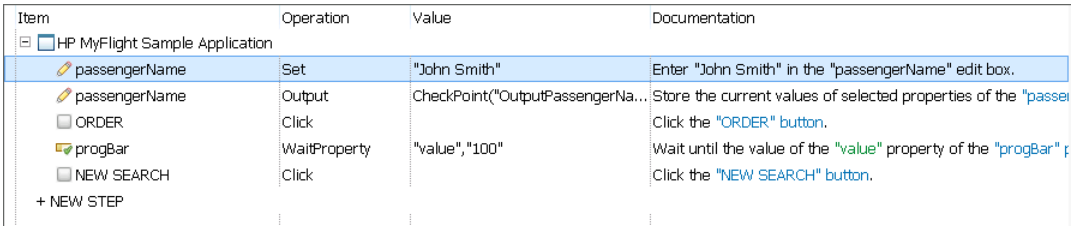
h. In the **Name** field, enter **passengerName_text**.

i. In the **Location in Data Table** area, ensure that the **Global sheet** is selected. This ensures that the output is saved in the Global data sheet, which makes the value accessible to other actions in the test.

j. Click **OK** to close the dialog box. UFT updates the value in the Output Value Properties dialog box to reflect the DataTable parameter storage option:



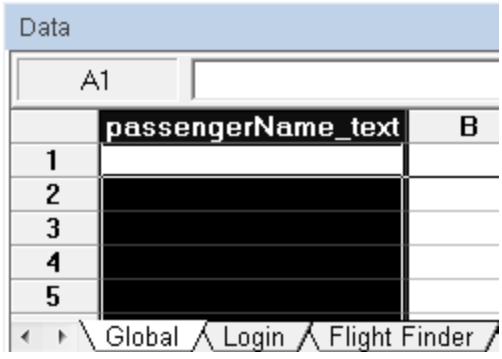
k. In the **Insert statement** area, select the **After current step** option, and click **OK**.
UFT inserts a **Output** step immediately following the **passengerName.Set** step:



In the Editor, the steps look like this:

```
WpfWindow("Book Flights").WpfEdit("passengerName").Set "John Smith"
WpfWindow("Book Flights").WpfEdit("passengerName").Output CheckPoint
("OutputPassengerName")
WpfWindow("Book Flights").WpfButton("ORDER").Click
WpfWindow("Book Flights").WpfProgressBar("progBar").WaitProperty "value",
"100"
WpfWindow("Book Flights").WpfButton("NEW SEARCH").Click
```

The Global sheet in the Data table is also updated accordingly:



7. **Parameterize the Search action with the stored output value.**

a. In the Solution Explorer, double-click the **Flight Order Search** action node. The Flight Order Search action opens as a separate tab in the document pane.

b. In the Flight Order Search tab, select the **byName** row.

c. In the **Value** column of the **byName** row, click the **Configure the value** button <#D>. The Value Configuration Options dialog box opens.

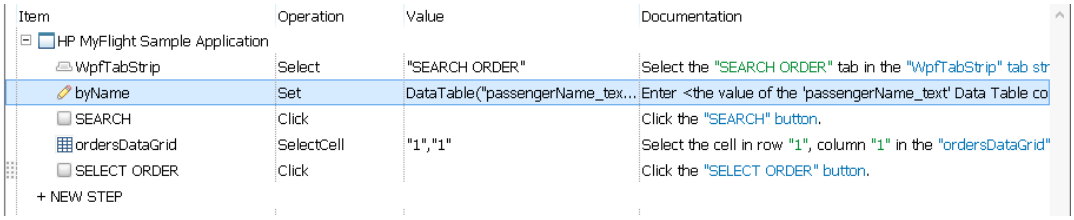
d. In the Value Configuration Options dialog box, select the **Parameter** radio button.

e. In the **Parameter** drop-down list, select **DataTable**. The dialog box updates the other fields accordingly.

f. In the **Location in Data Table** area, select the **Global sheet** option.

g. In the **Name** drop-down list, select the **passengerName_text** parameter and click **OK**.

UFT updates the **byName** row to reflect that the value is now provided by a Data Table Parameter:



In the Editor, the statement looks like this:

```
WpfWindow("HP MyFlight Sample Application").WpfEdit("byName").Set
DataTable("passengerName_text", dtGlobalSheet)
```

8. **Save the test.**

Click **Save**.

9. **Run the test and view the run results.**

a. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

b. In the Windows Applications tab of the Record and Run Settings dialog box, select the **Record and Run only on:** option. The application details should be saved from previous test runs.

c. Click **OK** to save the settings and close the dialog box.

d. In the toolbar, click the **Run** button.

e. In the **Results location** tab, ensure that the **New run results folder** option is selected. Accept the default results folder name.

f. Click **OK**. When the test run is completed, the run results open

g. In the Test Flow, find the **Flight Confirmation** action node.

h. Under the Flight Confirmation Summary node, select the **Standard Output: "OutputPassengerName"** node. The run results display a summary of the test step.

   The summary shows the details of the output value:

i. In the Test Flow, find the **Flight Order Search** node.

j. Under the Flight Order Search node, select the **byName.Set** node. The run results display a summary of the test step.

   The summary shows the result of this step, including the value used for the **Set** operation. This value should be the output value.

10. **Close the run results.**

    When you are finished viewing the run results, close the tab displaying the run results.

Lesson 7: Create functions and function libraries

UFT provides many built-in functions and methods that can satisfy many of your testing needs. However, there may be times when you need to perform a specific task that is not available by default for a particular test object class. In these cases, you can create a user-defined function for this task. You then save this function in a function library file which is associated with tests, and then insert the function call as a step whenever you need to perform the task.

In "Lesson 2: Creating Object Repositories" on page 33, you created shared object repositories and associated them with the action in your test. In this lesson, you will use a similar process by creating a function and a function library, and then associating the function library with your test. By associating this function library with your test, you can call any of the functions in the test.

This lesson includes the following:

Functions and function libraries

In UFT, you can create functions to perform special tasks not supported by UFT's standard classes and methods. A **function** is a set of coded steps that perform a particular task for which no suitable method exists by default. You may want your test to include such a task, and even repeat this task several times. Therefore, you need the function to be easily accessible.

Once you create functions, you can store the functions in a **function library**. These function libraries serve as a repository for your user-defined functions. Each function library can be assigned to a test (or multiple tests), which enables the test to then call the function as a test step.

In this lesson, you will create a function that checks the date format on a page generated by the flight reservation application, and then add the function call to your test.

Exercise 7a: Create a function

In this exercise, you will create a function that will be called from your test. This function checks whether the date is displayed in the proper format. The function also checks that the date is valid, for example, that the month does not exceed **12** or the day exceed **31**.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The solution is displayed in the Solution Explorer, including the Book Flights test.

2. **Create a new function library.**

   a. In the toolbar, click the **New** button down arrow 

and select **Function Library**. The New Function Library dialog box opens.

b. In the New Function Library dialog box, enter the function library details as follows:

Look in:	i. Browse to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** folder. ii. In this folder, create a new folder named **Tutorial_ Function Libraries**. iii. Open this folder.
File name:	CheckDate Function

c. Click **Create**. UFT opens the blank function library as a separate tab in the document pane.

3. **Create a function.**

In the CheckDate function library, paste the following code:

```
'The following function checks whether a date string (dateStr)
'has the characters representing DD-<MMM string>-YYYY

Function check_data_validity( dateStr )
        Dim firstDashPos, secondDashPos
        Dim mmPart, ddPart, yyyyPart
       firstDashPos = inStr( dateStr , "-" )
        secondDashPos = inStrRev( dateStr, "-" )
        If ( (firstDashPos <> 2 and firstDashPos <> 3) or (secondDashPos <>
6 and secondDashPos <> 7)) Then
            reporter.ReportEvent micFail,"Format check", "Date string is"&"
missing at least one dash ( - )."
        check_data_validity = False
            Exit function
        End If

        if firstDashPos = 2 Then
                        ddPart = mid( dateStr, 1, 1)
               else
                        ddPart = mid( dateStr, 1,2 )
               End If
        mmPart = mid( dateStr, firstDashPos+1, 3 )
        yyyyPart = mid( dateStr, secondDashPos +1 , 4 )

        If  inStr(mmPart, "Jan") and inStr(mmPart, "Feb") and inStr(mmPart,
"Mar") and inStr(mmPart, "Apr") and inStr(mmPart, "May") and inStr(mmPart,
"Jun") and inStr(mmPart, "Jul") and inStr(mmPart, "Aug") and inStr(mmPart,
"Sep") and inStr(mmPart, "Oct") and inStr(mmPart, "Nov") and inStr(mmPart,
```

```
"Dec") Then
            reporter.ReportEvent micFail, "Format Check", "The month"&"
value is invalid. It is not a valid month string."
            check_data_validity = False
            Exit function
        End If

        If ddPart > 31 Then
             reporter.ReportEvent micFail, "Format Check", "The date"& "
value is invalid. It exceeds 31."
             check_data_validity = False
             Exit function
         End If

        If yyyyPart < 2013 Then
                reporter.ReportEvent micFail, "Format Check", "The year"& "
value is invalid. (Prior to this year.)"
        check_data_validity = False
             Exit function

         End If

    check_data_validity = True

End Function
```

4. **Save the function library.**

   Click **Save**  .

5. **Close the function library.**

   Select **File > Close**. The function library tab is closed and the test remains open.

Now that you have created the function, you need to associate it with your test in order to use these functions in a test step. Continue with "Exercise 7b: Associate a function library with your test" below to see how to associate function libraries with your test.

Exercise 7b: Associate a function library with your test

In "Exercise 7a: Create a function" on page 132, you created a function and a function library to run a date check on a test object. However, before you can use this function in a test, you must associate the function library with a test.

In this exercise, you will associate the function library with your test.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The solution is displayed in the Solution Explorer, including the Book Flights test.

2. **Associate the CheckDate Function.qfl with the Book Flights test.**
   a. If the Solution Explorer is not already open, in the toolbar, click the **Solution Explorer** button  .
   b. In the Solution Explorer, right-click the **Book Flights** test node and select **Add > Associate Function Library**. The Open Function Library dialog box opens.
   c. In the Open Function Library dialog box, navigate to the **CheckDate Function.qfl** file, located in **C:\%HOMEPATH%\Unified Functional Testing\Tutorial_Function Libraries**, and click **Open**.
   d. In the Automatic Relative Path Conversion dialog box, click **Yes**. (This converts the path to the function library into a relative path.

      The function library is now associated with the Book Flights test, and is displayed in the Solution Explorer in the **Function Libraries** folder of the Book Flights test.

      > **Note:** Using a relative path keeps the path valid when you move folders containing tests and other files from one location to another, as long as the folder hierarchy remains the same.

3. **Save the test.**

   In the document pane, select the Book Flights canvas tab, and click **Save**  .

Now that you have associated the function library with your test, you can use its functions in test steps. Continue with "Exercise 7c: Perform a check using a function" below to use the function in a test step.

Exercise 7c: Perform a check using a function

In "Lesson 6: Creating checkpoints and output values" on page 95, you created a number of checkpoints to check objects in the flight reservation application.

In this exercise, instead of using the UFT user interface to create checkpoints, you will use the function you created in "Exercise 7a: Create a function" on page 132 to check the date format of a calendar object in the Flight Finder action.

1. **Start UFT and open the Book Flights test.**
   a. If UFT is not currently open, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. Select **File > Open > Solution**. The Open Solution dialog box opens.
   c. In the Open Solution dialog box, navigate to the **Flight Reservation Application.ftsln** file, located in **C:\%HOMEPATH%\My Documents\Unified Functional Testing**, and click **Open**.

      The solution is displayed in the Solution Explorer, including the Book Flights test.

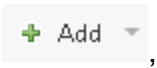
2. **Save the test as Book Flights Function.**
   a. In the Solution Explorer, right-click the **Book Flight**s node and select **Save As**. The Save Test As dialog box opens.
   b. In the Save Test As dialog ox, browse to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** folder.
   c. Enter **Book Flights Function** as the test name.

      In the Solution Explorer, the **Book Flights** test is replaced by the **Book Flights Function** test. The Book Flights test is still saved separately in the file system.

3. **Add the Book Flights test back into the solution.**

   You can have the Book Flights and Book Flights Function tests open at the same time, if they are included in the same solution. This enables you to switch back and forth between them if you want to compare or edit the test.

   > **Note:** You can only run a single test at a time.

   a. In the toolbar, click the **Add** button drop-down arrow 

, and select **Add Existing Test**. The Add Existing Test to Solution dialog box opens.
   b. In the Add Existing Test to Solution dialog box, navigate to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** folder.
   c. In the folder, select the **Book Flights** test, and click **Open**.

      The Book Flights test node is again added to the Solution Explorer. Note that tests are organized alphabetically.

4. **Display the Flight Finder page in the flight reservation application.**

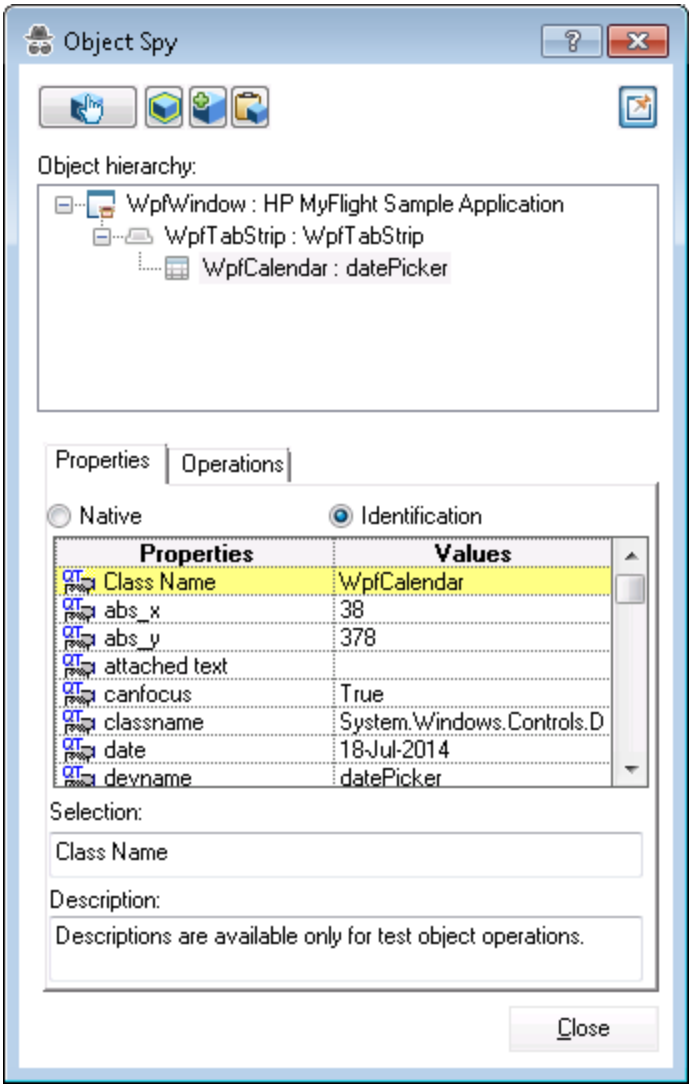
a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.
b. In the log in screen, enter the login user credentials:
   - **User name:** john
   - **Password:** hp
c. Click **OK** to sign in. The Flight Finder page opens.

5. **Spy on the object properties for the checkpoint object.**

   In this exercise, you are using a function to check the date format for a calendar object. In order to do so, you must learn the properties of the object on which you want to set the checkpoint.

   a. In the UFT window, in the toolbar, click the **Object Spy** button  . The Object Spy dialog box opens.
   b. Move the Object Spy dialog box to the edge of the window so you can spy on your application with the Object Spy dialog box still open and visible.
   c. In the Object Spy dialog box, click on the pointing hand button  . When you press the pointing hand, UFT is hidden and the Object Spy dialog box is displayed over the flight reservation application.
   d. In the flight reservation application, click on the date entry field object:

The Object Spy dialog displays the object properties:



e. In the properties grid (bottom of the dialog box), scroll through the properties until you find the property with the value of the date (in the format **DD-MMM-YYYY**). Note the name of this property, as you will need it for the function call step inserted in your test later.

f. Click **Close** to close the Object Spy and return to your test.

6. **Open the action for the function call.**

   In the Solution Explorer, expand the **Book Flights Function** test node and double-click the **Flight Finder** action.

   The Flight Finder action opens as a separate tab in the document pane.

7. **Add the function call in the Editor.**

In this step, you will add the check_data_validity function you added to your function library in "Exercise 7a: Create a function" on page 132.

a. If the Keyword View is displayed, select **View > Editor** to display the Editor.

b. In the Editor, place your cursor after the **WpfCalendar.SetDate** step and press **ENTER**.

c. Add a step to the Editor to retrieve the date property for the **datePicker** object:

```
departureDate = WpfWindow("HP MyFlight Sample Application").WpfCalendar
("datePicker").GetROProperty("date")
```

This step retrieves the value of the date property in order for UFT to run the checkpoint function on that object when it checks that property.

d. Add another step to the Editor to call the checkpoint function:

```
If check_data_validity(departureDate) Then
	reporter.ReportEvent micPass, "Date is valid", departureDate
End If
```

After you have added these steps, the action should look like this (although the date you use will be different):

```
WpfWindow("HP MyFlight Sample Application").WpfComboBox
("fromCity").Select "Los Angeles"
WpfWindow("HP MyFlight Sample Application").WpfComboBox("toCity").Select
Sydney
WpfWindow("HP MyFlight Sample Application").WpfCalendar
("datePicker").SetDate "17-Jul-2014"
departureDate = WpfWindow("HP MyFlight Sample Application").WpfCalendar
("datePicker").GetROProperty("date")
If check_data_validity(departureDate) Then
	reporter.ReportEvent micPass, "Date is valid", departureDate
End If
WpfWindow("HP MyFlight Sample Application").WpfComboBox("Class").Select
"Business"
WpfWindow("HP MyFlight Sample Application").WpfComboBox
("numOfTickets").Select "2"
WpfWindow("HP MyFlight Sample Application").WpfButton("FIND
FLIGHTS").Click
```

e. After you paste the steps, make sure that the following step still starts on its own line after the pasted steps.

```
WpfWindow("HP MyFlight Sample Application").WpfComboBox("Class").Select
```

```
"Business")
```

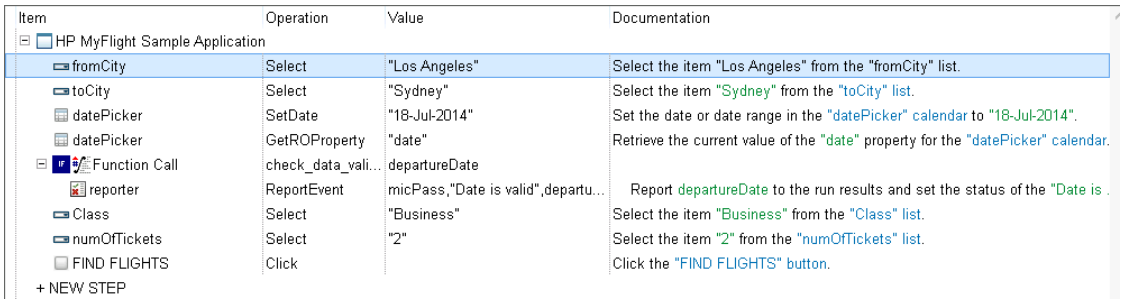
If it does not, place your cursor before this step and press **ENTER**.

You want UFT to run this function before it edits the next field in the application.

8. **Look at these steps in the Keyword View.**

   Select **View > Keyword View** to switch to the Keyword View.

   Note the function call step as a separate step under the **GetROProperty** step:



Item	Operation	Value	Documentation
HP MyFlight Sample Application			
fromCity	Select	"Los Angeles"	Select the item "Los Angeles" from the "fromCity" list.
toCity	Select	"Sydney"	Select the item "Sydney" from the "toCity" list.
datePicker	SetDate	"18-Jul-2014"	Set the date or date range in the "datePicker" calendar to "18-Jul-2014".
datePicker	GetROProperty	"date"	Retrieve the current value of the "date" property for the "datePicker" calendar.
Function Call	check_data_vali...	departureDate	
reporter	ReportEvent	micPass,"Date is valid",departu...	Report departureDate to the run results and set the status of the "Date is .
Class	Select	"Business"	Select the item "Business" from the "Class" list.
numOfTickets	Select	"2"	Select the item "2" from the "numOfTickets" list.
FIND FLIGHTS	Click		Click the "FIND FLIGHTS" button.
+ NEW STEP			

9. **Save your test.**

   Select **File > Save**.

10. **Run the test.**

    a. In the toolbar, click the **Run** button  . The Run dialog box opens.

    > **Note:** Before running the test, ensure that the flight reservation application is closed.

    b. In the Run dialog box, in the **Result locations** tab, ensure that the **New run results folder** option is selected. Accept the default results folder name.

    c. Click **Run**. UFT runs the steps in sequence.

    After the test run is completed, the run results open.

11. **Analyze the run results.**

    a. In the run results, in the Test Flow, under the **Flight Finder** node, select the **Date is valid node**.

    Note that a green checkmark is displayed next to the step name. This informs you that the checkpoint passed per the function you added.

Lesson 8: Using Insight in your Test

Sometimes, when you are creating your test and test objects, normal object identification does not help identify an object in your application for testing purposes.- Standard object identification relies on the object's properties, such as position in the application or browser window, time of appearance in the window, or a number of other properties - does not help identify an object in your application for testing purposes.

For situations where the regular object identification does not work correctly or does not suit your needs, UFT also has a image-based object recognition mechanism named **Insight**. This mechanism enables you to identify objects by capturing a snapshot of the image and using the captured image as the object during the test run.

In this lesson, you will be learning how to use Insight to identify objects and use these objects in your test.

This lesson includes the following:

Insight object identification

In UFT, you can use **Insight**, which is an image-based identification ability, to recognize objects in your application. Insight identifies objects based on what they look like, instead of using the object properties which are part of the application/object design. Insight object identification is particularly useful if your application is designed with a technology that UFT does not suppport or with an application running on a remote computer.

When you use Insight object identification, UFT stores an image of the object as part of the Insight test object it creates. Then, when running the test, UFT uses the image as the main object property to identify the object in the application in run-time.

You can create Insight objects both in the object repository or when recording. In this lesson, you will be working only with adding Insight objects in the object repository.

Exercise 8a: Create a test for Insight objects

In this exercise, you create a test to add test steps using Insight objects. This test will be saved separately from the solution containing the **Book Flights** test.

1. **Start UFT.**
   a. If UFT is not open, open it as described in "Create a solution" on page 26. Make sure that the WPF Add-in is open.
   b. In the toolbar, click the **New** button 

. The New Test dialog box opens.
   c. In the New Test dialog box, select **GUI Test**.
   d. Name the test `Insight`. Accept the default location.

      Note: Do not enter a solution name for this test. This test is saved separately from the solution used in the other lessons in this tutorial.

   e. Click **Create** to create the test.

      The test opens in a separate tab in the document pane.

2. **Associate the object repositories with your action.**

   In this test, you will keep all the test steps in one action. However, you will need multiple object repositories in order to have the appropriate test objects.

In this test, you will log in to the flight reservation application, then click a link in a promotional image to order a flight. As a result, you will need to include the object repositories for the Login page and the Flight Finder page of the application.

a. In the Solution Explorer, right-click the **Action1** node and select **Associate Repository with Action**. The Open Shared Object Repository dialog box opens.

b. In the Open Shared Object Repository dialog box, navigate to the **Login.tsr** object repository file you created in "Lesson 2: Creating Object Repositories" on page 33, located in **C:\%HOMEPATH%\Unified Functional Testing\Tutorial_Object Repositories** and click **Open**.

   The **Login.tsr** object repository is displayed as a sub-node of the **Action1** node.

c. Repeat the process described above to add the **Flight Finder.tsr** object repository file to Action1.

3. **Save the test.**

   Click **Save**  .

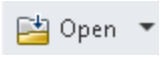
Now that you have created a test in which to work with Insight objects, continue to "Exercise 8b: Add an Insight object to the object repository" below to add the Insight objects to your object repositories so they are available for your test steps.

Exercise 8b: Add an Insight object to the object repository

In "Exercise 8a: Create a test for Insight objects" on the previous page, you created the structure of the test you will use for your Insight objects. However, before you can add these objects to a test, you must add them to an object repository. In this lesson, you will learn how to use the Object Repository functionality to include Insight objects in your tests.

1. **Start UFT.**

   a. If UFT is not open, open it as described in "Create a solution" on page 26. Make sure that the WPF Add-in is open.

   b. In the toolbar, click the **Open** drop-down arrow 

and select **Open Test**. The Open Test dialog box opens.

   c. In the New Test dialog box, navigate to the **Insight** test, saved in **C:%HOMEPATH%\Unified Functional Testing**, and click **Open**.

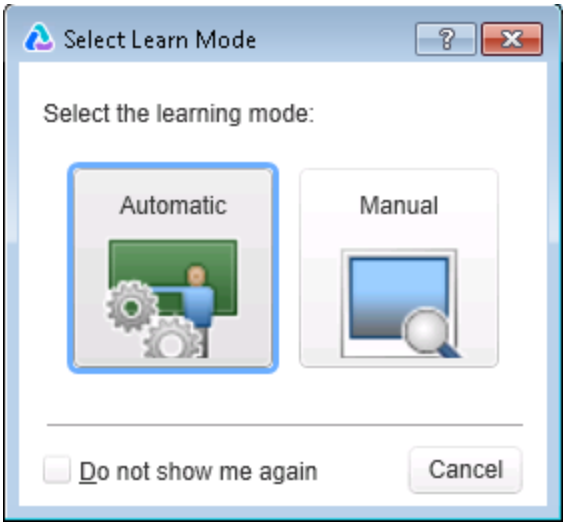
      The test opens in a separate tab in the document pane.

2. **Create a new object repository.**
   a. Select **Resources > Object Repository Manager**. The Object Repository Manager dialog box opens.
   b. In the Object Repository Manager window, select **File > Save**. The Save Shared Object Repository dialog box opens.
   c. In the Save Shared Object Repository dialog box, navigate to the **Tutorial_Object Repositories** folder, located in **C:%HOMEPATH%\Unified Functional Testing**.
   d. Name the object repository **Insight** and click **Save**.

3. **Open the flight reservation application to the Flight Finder page.**
   In the flight reservation application, you will notice that on the Flight Finder page, there is a changing object, displaying advertisement for special flights:

   Because this image is changing periodically in the application, it is the type of object that you can test with Insight object recognition, as traditional object recognition relies on such things as position on the screen relative to other objects, ordinal identifiers, and the like. Furthermore, the **Order** button located inside the image is not identifiable as a separate object. For example, if you use the Object Spy to identify the **Order** button, UFT cannot identify it.
   a. Open the flight reservation application, as described in "Explore the flight reservation application" on page 23.
   b. In the Login window, enter the user credentials:
      - **Username:** john
      - **Password:** hp
   c. Click **OK** to log in to the application. The Flight Finder page opens.

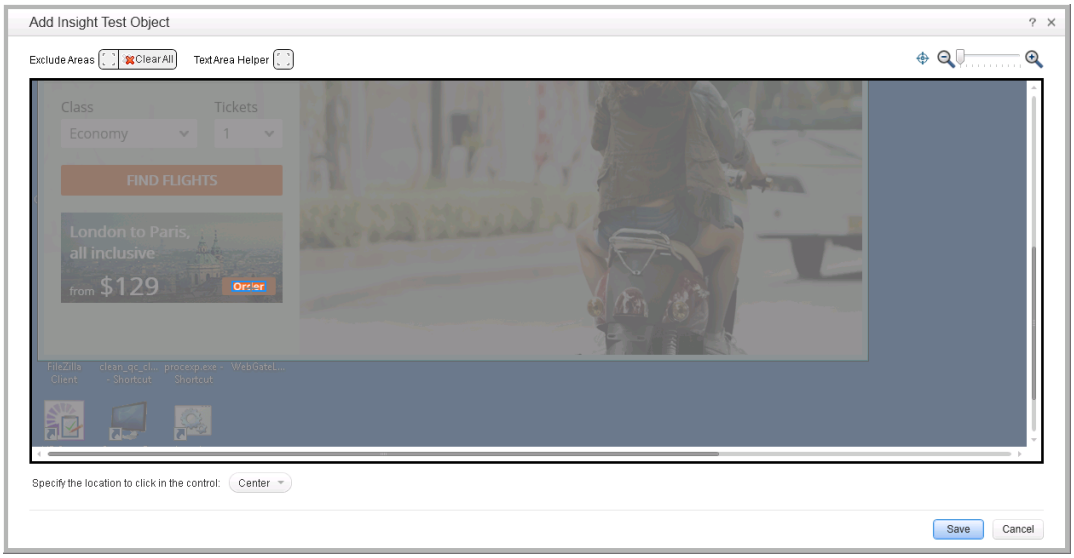
4. **Add the Insight object to the object repository.**
   a. In the Object Repository Manager, in the toolbar, click the **Add Insight Object** button  . The Select Learn Mode dialog box opens:



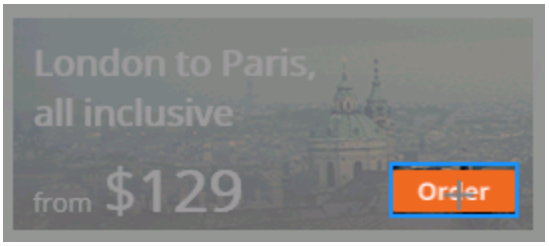
b. In the Select Learn Mode dialog box, click the **Automatic** button. UFT is hidden from view and your application is displayed.

> **Note:** Selecting the **Automatic** mode enables UFT to automatically select the region/object as the Insight object. If you were to select **Manual**, you can specify the region/object to use as the Insight object.

c. In the flight reservation application, click on the **Order** button inside the flight promotion image. The Add Insight Test Object dialog box opens:



d. In the image editor (center of the dialog box), drag the object selection box so that it includes the entire Order button:



e. In the **Specify the location to click in the control** option, select **Center**.

f. Click **Save** to add the object to the object repository.

The Insight object is added as a top-level object in the object repository:

g. In the Object Repository Manager, in the Object Properties pane (right side), rename the object from **InsightObject** to **Promotion Order**.

h. In the toolbar, click **Save** to change the object name and save the object repository.

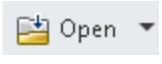
Now that you have created the object repository containing the Insight images, you can use these objects with your test. Continue to "Exercise 8c: Use Insight objects in a test" below to use the Insight objects in your test and run the test.

Exercise 8c: Use Insight objects in a test

In "Exercise 8b: Add an Insight object to the object repository" on page 143, you created Insight objects for an object (image) in your application. In this exercise, you will use these objects in a test and see how the test runs when using Insight object identification.

1. **Start UFT.**

   a. If UFT is not open, open it as described in "Create a solution" on page 26. Make sure that the WPF Add-in is open.

   b. In the toolbar, click the **Open** drop-down arrow 

and select **Open Test**. The Open Test dialog box opens.

c. In the New Test dialog box, navigate to the **Insight** test, saved in **C:%HOMEPATH%\Unified Functional Testing**, and click **Open**.

The test opens in a separate tab in the document pane.

2. **Associate the Insight object repository with your test.**

a. In the Solution Explorer, right-click the **Action1** node and select **Associate Repository with Action**. The Open Shared Object Repository dialog box opens.

b. In the Open Shared Object Repository dialog box, navigate to the **Insight.tsr** object repository file, saved in **C:\%HOMEPATH%\Unified Functional Testing\Tutorial_Object Repositories**, and click **Open**.

The **Insight.tsr** file is now displayed as a sub-node of the **Action1** node in the Solution Explorer and its objects are available for use in your tests.

c. Click **Save** to save the changes.

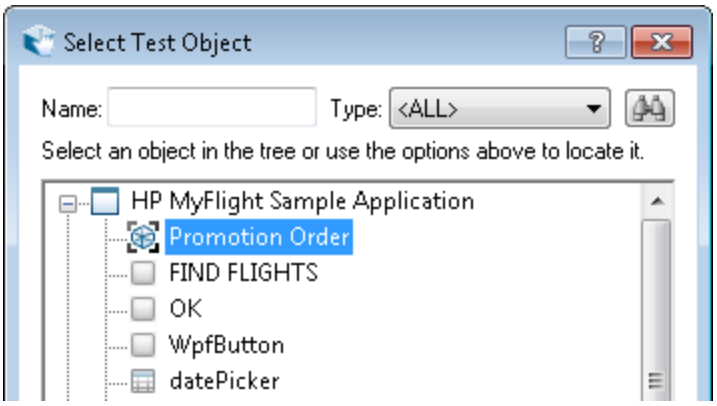
3. **Add the login steps to the test.**

Add steps to your test to log in to the flight reservation application, as described in "Exercise 3a: Add steps to the Login action in the Keyword View" on page 48.

4. **Add the Insight object to your test.**

a. In the Keyword View, below the **OK** button step, click in the Item column and select **Object from Repository** from the drop-down list. The Select Test Object dialog box opens.

b. In the Select Test Object dialog box, in the test object tree, select the **Promotion Order** node:



c. Click **OK** to add the step to your test.

UFT adds a new step for the Insight object to your test.

5. **Run the test and view the run results.**

a. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.
b. In the **Windows Applications** tab, select the **Record and run only on:** option.
c. Under the Record and Run only on option, select the **Applications specified below** option. The application details should be saved from previous run sessions.

> **Note:** If you need to re-enter the application details, see "Exercise 4a: Run a test" on page 74 for the necessary details.

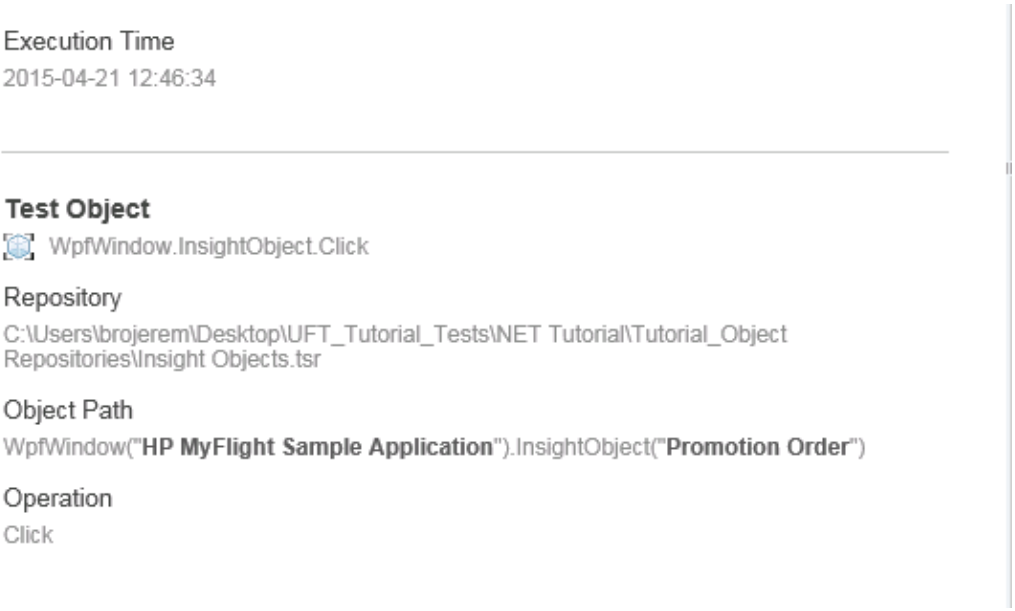
d. Click **OK** to save the changes and apply the settings.
e. In the toolbar, click the **Run** button  . The Run dialog box opens.
f. In the Run dialog box, in the **Results Location** tab, select the **New run results folder**. Accept the default folder name.
g. Click **Run** to start the test run.

> **Note:** Make sure that the flight reservation application is closed before performing the test run.

UFT opens the flight reservation application and runs the test steps in sequential order. When UFT runs the steps containing the Insight objects, it uses the Insight object identification mechanism to find the objects.

When the test run is complete, the run results opens.

h. In the Test Flow, select the **Promotion Order.Click** node. The run results display a summary of the step.
i. In the step details, you can see the object that UFT identified using Insight object identification during the test run:


Execution Time
2015-04-21 12:46:34
Test Object
WpfWindow.InsightObject.Click
Repository
C:\Users\brojerem\Desktop\UFT_Tutorial_Tests\NET Tutorial\Tutorial_Object Repositories\Insight Objects.tsr
Object Path
WpfWindow("HP MyFlight Sample Application").InsightObject("Promotion Order")
Operation
Click

Part 4: Create and run automated API tests

In addition to testing your application's user interface, you should also test the non-GUI (service) layers to ensure that the APIs that run your application are working properly. Using UFT, you can create automated API tests that do this for you.

When you create an API test, you create an overall test flow consisting of test steps which individually test the application's API processes. You provide the input and checkpoint properties for these test steps, and UFT runs the test in your application. When the test run finishes, you can check the results to see how your application is functioning.

In this part, you will learn how to create API tests for a variety of different types of applications and services.

This section includes the following:

Lesson 1: Create an API test

Before creating the content of a test of your application's service layer, you must first create a test and create the test structure.

1. In the UFT toolbar, click the **New** button down arrow  and select **New Test**.
2. In the New Test dialog box, select **API Test**.
3. Enter the following details for your test:

   Name: Basic

   Location: By default, UFT saves documents at **C:\%HOMEPATH%\My Documents\Unified Functional Testing**. For this lesson, you do not need to modify this path.
4. Click **Create**.

   A blank test opens in the canvas with a tab for the test flow (named **Basic**). Inside the test flow canvas is an empty test flow.

   This test is also displayed as a subnode of the **Untitled Solution** node in the Solution Explorer pane. (This is the general solution name used when a test is created but not a named solution.)

You are now ready to begin designing API tests. Continue to "Lesson 2: Create simple API test steps" on the next page to learn how to create API test steps.

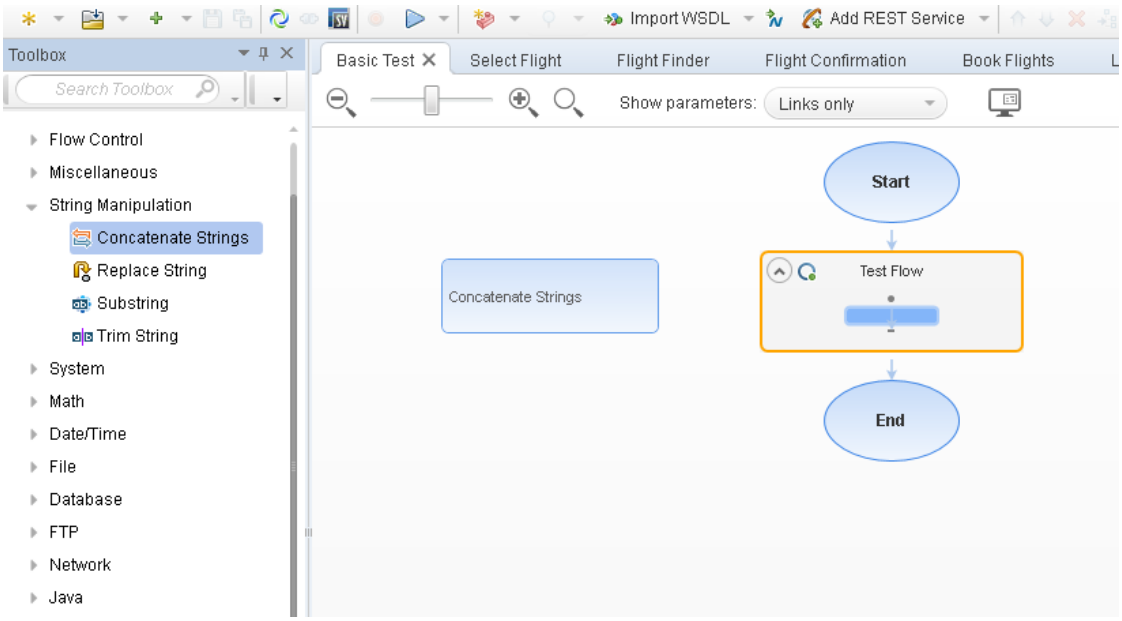
Lesson 2: Create simple API test steps

In UFT API tests, the process of creating tests is a visual one. Your test steps are displayed on a canvas which shows the entire master test flow.

Creating the actual test steps consists of two main parts:

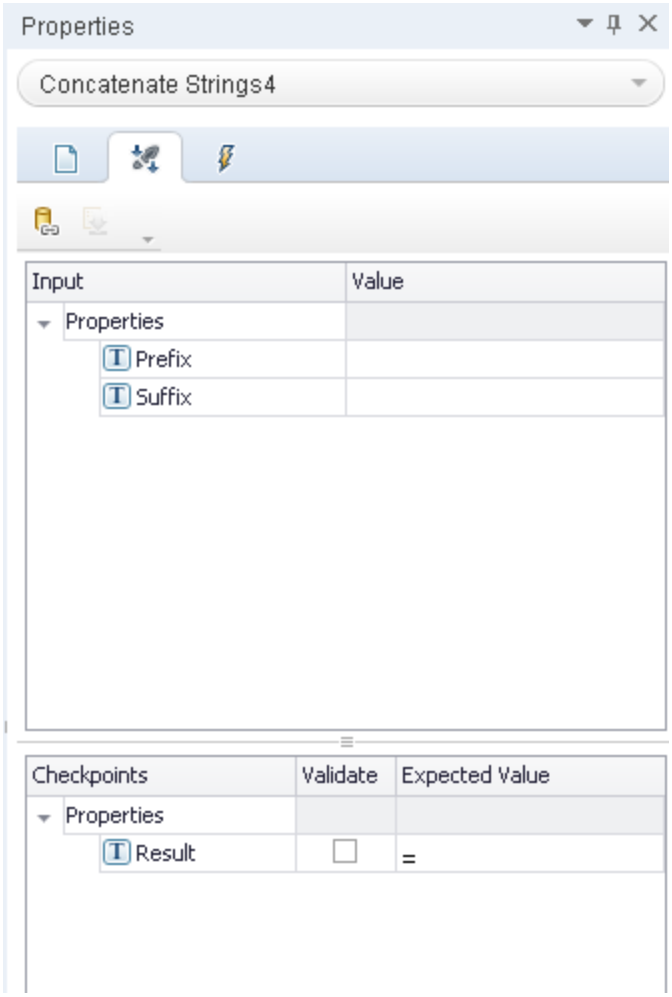
1. **Creating test steps by dragging the appropriate activities to the test flow in the canvas.**

   The Toolbox pane contains all the activities you can use in your test. From the list of activities contained in the canvas, you can drag them to the canvas and add them to the test flow in any place:



2. **Adding step properties (input and checkpoints of the step)**

   After you drag a test step to the canvas, the properties for the step are displayed in the Properties pane each time you select the step in the canvas:



   Each step has two different kinds of properties:

Input	These properties are used by UFT to set the values that the activity needs to run the step.
Checkpoint	These values are compared against the actual values after the step is run to ensure that it runs correctly (or does not run correctly). Checkpoint properties are optional when running a test.

   In this lesson, you will use these basic capabilities to create a basic test.

1. **Start UFT and open the Book Flights test.**

   a. If UFT is not currently open, open it as described in "Create a solution" on

page 26.

b. On the Start Page, in the Recent tests/components area, click **Basic**.

The **Basic** test opens in the document pane.

2. **Add an activity to the canvas and define its properties.**

a. In the toolbar, click the **Toolbox** button  . The Toolbox pane opens and displays different categories of activities.

b. In the Toolbox pane, expand the **String Manipulation** node.

c. From the list of String Manipulation activities, drag the **Concatenate Strings** activity to the canvas.

A new block is added to the canvas named **Concatenate Strings**, and the Properties pane opens to the test step properties.

d. In the Properties pane, select the **Input/Checkpoints** tab  .

e. In the Input/Checkpoints tab, in the **Input** section (top part), enter the following values for the step properties:

- **A:** Hello (with a space after)
- **B:** World.
- **Checkpoint:** Hello world.

3. **Add another activity to the canvas and define its properties.**

In addition to defining input and checkpoint properties, you can define other properties for the test step.

a. In the Toolbox pane, expand the **String Manipulation** node.

b. From the list of String Manipulation activities, double-click the **Replace String** activity to add it to the canvas. The **Input/Checkpoints** tab opens in the Properties pane, displaying the input and checkpoint properties for the step.

c. In the Properties pane, open the **General** tab  .

d. In the Name property row, change **Replace String** to **Change Text** and press **Enter**. The name of the step in the canvas is changed to **Change Text**.

e. Open the Input/Checkpoints tab.

f. In the Input Checkpoints tab, enter the following values for the properties:

- **Source string:** Hello world.
- **Search string:** Hello
- **Replace string:** Goodbye
- **Case-sensitive:** False

4. **Add checkpoint properties to the Change Text step.**

a. In the Checkpoints section of the Input/Checkpoints tab, in the **Result** row, select the checkbox in the **Validate** column. This enables the checkpoint for this step
b. In the **Expected value** column, enter the expected result: **Goodbye world**.

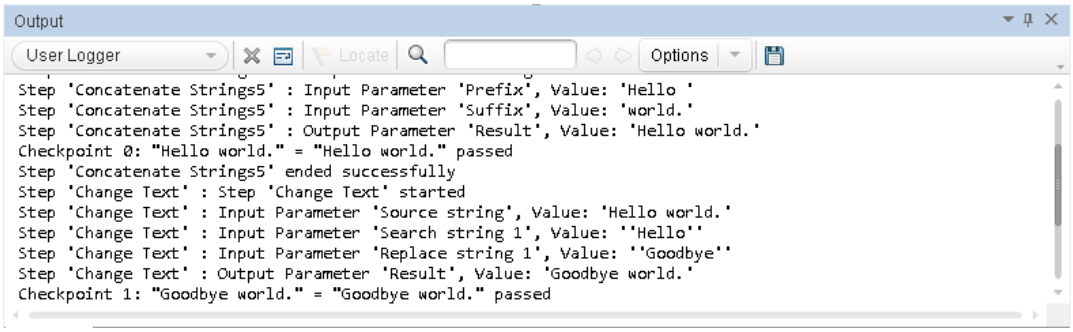
5. **Save the test.**

   Select **File > Save**.

6. **Run the test and view the results.**

   a. In the toolbar, click the **Run** button . The Run dialog box opens.
   b. In the Run dialog box, click **Options** to expand the dialog box.
   c. In the **Results Location** tab, select the **Temporary run results folder** option.
   d. Click **Run** to compile and run the test.

      While UFT runs the test, the Output pane shows the compilation log. In the lines displayed in the compilation log, you can see the input and checkpoint property values you entered in the previous steps:



      After the test run is complete, the run results open.

   e. In the run results, select **Test Flow** to display the steps in the test..
   f. In the test step tree, select the **Concatenante Strings** step node. The run results display a summary of the step.
   g. In the step summary area, click the **Concatenate Strings.xml** link. A separate tab opens in the document pane.
   h. In the new tab, look at the **Prefix** and **Suffix** rows. You will see that UFT ran the step with the exact input and output values you provided in the previous steps.
   i. In the test step tree, select the **Change Text** step.
   j. In the step summary area, click the **Change Text.xml** link.
   k. In the new tab, look again at the input properties you used for this step.
   l. Under the Change Text step, select the **Checkpoints** step.
   m. In the step summary area, click the **Checkpoint.xml** link.

n. In the new tab, note the details for the test step run. Note that that the results show that the checkpoint passed (with a green checkmark) and the expected string that you entered in the previous steps.
o. When you are finished reviewing the results, close the tab with the run results.

Lesson 3: Create API test steps using standard activities

When you create an API test, you are testing to see that the non-GUI (service) layer of your application works properly. The invisible processes that run your application can be any number of things: calls to a database, calls to a Web service, opening a program, sending messages via the Web, and so forth.

To assist you in creating tests, UFT provides a number of standard API activities to use in designing test steps. In this lesson, you will use the standard activities to create a basic test.

This lesson includes the following:

UFT API Testing Standard Activities

When you create an API test, there are a number of standard activities provided with all tests. These test common application processes, including:

- **Flow Control** activities, such as **Wait**, **Break**, and **Condition** step.
- **String Manipulation** activities, such as **Concatenate Strings** and **Replace String**
- **File** system activities for processes performed using the file system
- **Database** activities
- **FTP** activities
- **Network** activities, such as **HTTP Request** and **SOAP Request**
- **JSON** and **XML** activities for application processes that involve interacting with XML or JSON strings/files
- **Math** and **Date/Time** activities
- Other **Miscellaneous** activities, including a **Custom Code** activity, **Run Program** and **End Program** activities, and a **Report** activity.

There are a number of technology-specific activities:

- A **Call Java Class** activity which tests Java processes used in your application
- **JMS** (Java Message Service) activities
- **IBM Websphere MQ** activities
- **SAP** activities to access an SAP iDOC or RFC stored on a SAP server
- **Load Testing** activities which help your test run (after a conversion to a LoadRunner script) with HP LoadRunner
- **HP Automated Testing Tools** activities, which enable you to call a GUI test or action, API test or action, or Virtual User Generator Script from UFT, QuickTest Professional, Service Test, or LoadRunner as part of your test

There are a number of custom activities you can also import into your test, but these types of activities will be discussed in detail in later lessons.

Exercise 3a: Create a test with standard activities

In "Lesson 2: Create simple API test steps" on page 152, you familiarized yourself with the UFT API testing user interface and learned how it is used to create and run test steps.

In this lesson, you will use that knowledge to use standard activities to create a basic API test.

1. **Create a new test.**

   a. In the toolbar, click the **New** button 

. The New Test dialog box opens

   b. In the New Test Dialog Box, select **API Test**.

   c. Enter the following details for your test:

      - **Name: Standard**
      - **Location: C:\%HOMEPATH%\My Documents\Unified Functional Testing**

   d. Click **Create**.

      A blank test opens in the canvas, with a blank test flow. The test is also displayed as a sub-node of the **Solution Untitled** solution in the Solution Explorer pane.

2. **Create the steps in the test flow.**

   In this step, you are going to create a test of an application process which finds a certain string, replaces it, and then writes the result of the string replacement to a file.

   For this test, you will need three activities:

   - Replace a string
   - Create a file to save the results
   - Write the results to the file

   These activities are all provided with UFT's standard activities.

   a. If it is not already open, open the Toolbox pane by clicking the **Toolbox** tab at the lower left corner of the UFT window.

   b. In the Toolbox pane, expand the **String Manipulation** activities node.

   c. In the String Manipulation activities, drag the **Replace String** activity to the canvas. A new block is added to the test flow in the canvas and the

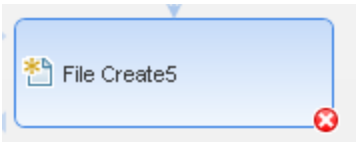
      Properties pane displays the **Input/Checkpoints** tab  for the Replace String step.

   d. In the Toolbox again, expand the **File** activities.

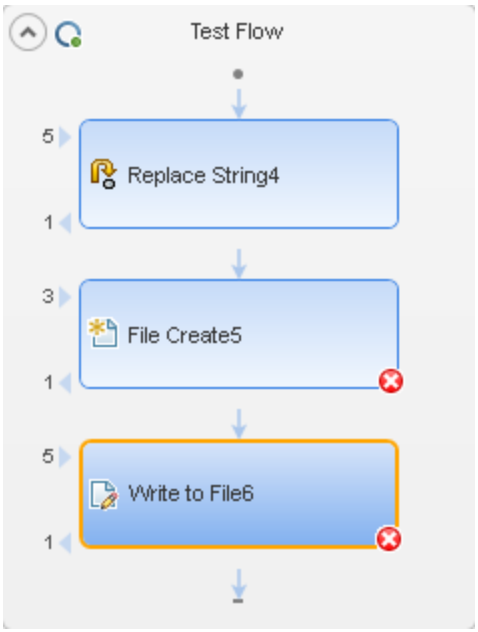
   e. In the File activities, drag the following activities to the canvas:

      - **File Create**
      - **Write to File**

Two new blocks are added to the canvas for each activity, and each of the activities are displayed with a red alert icon:



After you add the activities to the canvas, your test flow should look like this:



3. **Enter the properties for the Replace String activity.**

   a. In the canvas, select the **Replace String** activity. The **Input/Checkpoints** tab in the Properties pane opens.

   b. In the Input/Checkpoints tab, enter the input properties for the step:

      - **Source string:** Hello world.
      - **Search string:** Hello
      - **Replace string:** Goodbye

      > **Note:** You do not need to use the checkpoint properties for this activity.

4. **Enter the properties for the File Create step.**

This step will enable you to create a file on which to write the results of the Replace string operation from the previous step. You will create a file in a specified directory to be used in the next steps.

a. In the canvas, select the **File Create** step. The Input/Checkpoints tab in the Properties pane opens.
b. In the file system, open the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** folder.
c. Inside the Unified Functional Testing folder mentioned in the previous step, create a folder named **Tutorial_Files**. You will use this folder to create the file in the next step.
d. In the Input/Checkpoints tab, enter the input properties for the step:
   - **Folder path: C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Files**

     > **Note:** You can also enter this folder by pressing the Browse button and navigating directly to the folder.

   - **File Name:** Standard Test Result
   - **Overwrite:** true
e. In the **Checkpoints** section of the Input/Checkpoints tab, in the **Result** row, select the checkbox in the **Validate** column. Leave the default value as it is.

   This enables you to see if the file creation step was completed successfully running the test and viewing the run results.

5. **Enter the properties for the Write to File step.**

   This step will write the string resulting from the **Replace String** step to the file you created in the **Create File** step.

   a. In the canvas, select the **Write to File** step. The Input/Checkpoints tab in the Properties pane opens.
   b. In the Input section of the Input/Checkpoints tab, enter the input properties for the step:
      - **Content:** Goodbye world.
      - **File path:** You will use the file created in the previous step. You must manually enter the following: **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Files\Standard Test Result**
      - Keep all other properties with the default.

c. In the **Checkpoints** section of the Input/Checkpoints tab, in the **Result** row, select the checkbox in the **Validate** column (Leave the default value as it is.)

   This enables you to see if the write to file operation was completed successfully when running the test and viewing the run results.

6. **Save the test.**

   Select **File > Save**.

7. **Run the test.**

   a. In the toolbar, click the **Run** button  . The Run dialog box opens.

   b. In the Result Locations tab in the Run dialog box, ensure that the **Termporary run results folder** is still selected.

   c. Click **Run** to compile and run the test.

      After the test run is complete, the run results opens.

   d. In the file system, open the **C:\%HOMEPATH%\My Documents\Unified Functional Testing\Tutorial_Files\Standard Test Result** file.

      In this file, you should see the string **"Goodbye world"** in the first line of the document. This shows that UFT ran the test steps on a real application, just as it would for any application, using the standard API activities provided with UFT.

Now that you have created a basic test of application processes using standard activities, continue to "Lesson 4: Parameterize API test steps" on the next page to learn how to use data in your tests and parameterize test steps.

Lesson 4: Parameterize API test steps

In "Lesson 2: Create simple API test steps" on page 152 and "Lesson 3: Create API test steps using standard activities" on page 157, you learned how to provide the values for API test steps by manually entering the required input and checkpoint values.

However, you can also provide the input and checkpoint values from other sources:

- Data sources included with your test (including Excel files, XML, database data sources, or locally-created tables
- Output of previous steps
- A combination of all the above: manually entering the required data, data sources, and output of previous steps

Using these data sources to populate step input and output values enables you to mimic how your application works, as the input of an application process can come from a data source, the result of a previous application process and the like.

In this lesson, you will learn how to parameterize API test steps using different methods.

This lesson includes the following:

Parameterizing API test steps

When you provide values for the test step properties in an API test, the default way to provide these values is to manually enter them in the Properties pane.

However, using this method does not necessarily provide a realistic test of your application. In many applications, internal API processes receive their information - often dynamically- from data sources, other test step outputs, or a variety of both.

As a result, UFT provides a number of different ways to provide (parameterize) step values:

Manual entry	When you manually enter the step values, you select each step and type or select the appropriate values for the input and checkpoint property values. This method does not provide for easy test maintenance, as each time that your application's properties change, you must update each step and each property in the test.
Linking to a data source	When you link the step properties to a data source, UFT takes the values from the data source during a test run and uses the value provided in the data source. If your data source has multiple sets of data, you can run a test with multiple iterations to provide different values for the input and checkpoints to see how your application performs with different data input. This method provides easier test maintenance, as you only need to update the data source values instead of each test step.
Linking to the output of a previous step	When you link the step properties to the output of a previous step, UFT takes the values from the output of the step and uses these values during the test run. This enables you to mimic real application behavior, in which the output of an application's API process often passes a value to another process as input.
Linking to multiple sources	If your application's input and checkpoints are provided from multiple sources - a static string, data, and output of other steps/processes, you can create custom expressions to perform this in your test. UFT then provides the values using the custom expressions and uses the values during the test run.

Exercise 4a: Parameterize a test step from a data source

As you saw in "Parameterizing API test steps" on the previous page, one of the ways you provide values to test steps is by linking the step property values to a data source. This enables you to run the test step with multiple different values, depending on the structure of the data source.

In your API test, you can add multiple different types of data sources:

- Excel sheets
- XML files or schemas
- Databases
- Locally-stored data tables

Each of these can be added to your test, which makes it available for all steps in the test.

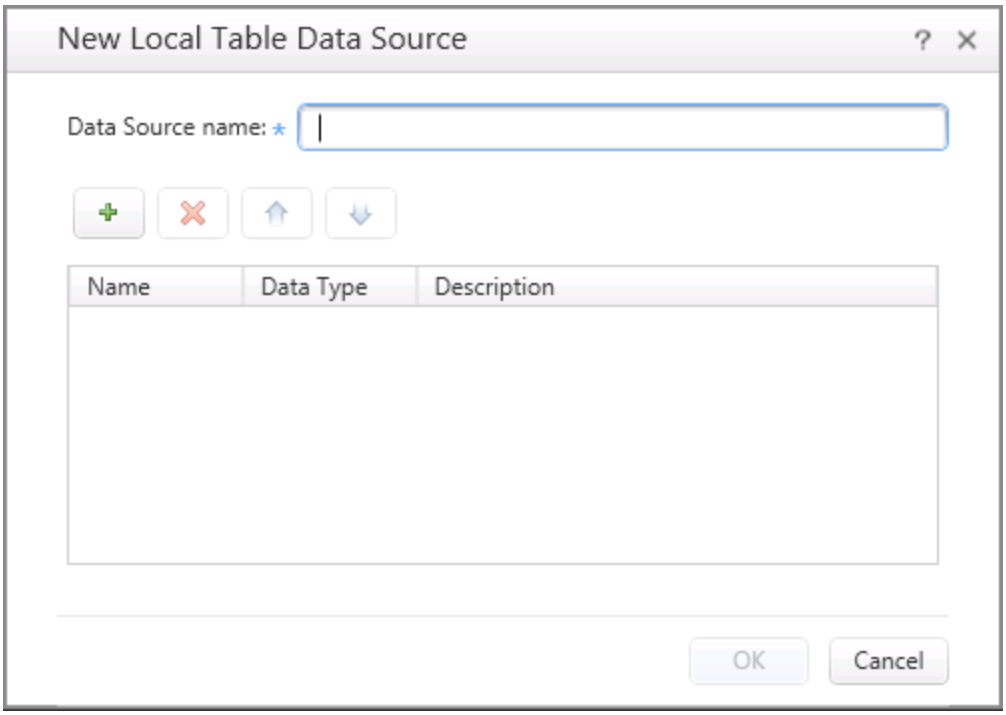
In this lesson, you will link test steps to a data source.

1. **Start UFT and open the Standard test.**
   a. If UFT is not currently open, open it as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent tests/components** area, click **Standard**.
      The Standard test you created in "Lesson 3: Create API test steps using standard activities" on page 157 opens as a separate tab in the document pane. It is also displayed as a sub-node of the **Solution Untitled** solution in the Solution Explorer.

2. **Add a Concatenate Strings step to your test.**
   a. Select **View > Toolbox** to open the Toolbox pane.
   b. In the Toolbox pane, expand the **String Manipulation** node.
   c. In the String Manipulation node, drag a **Concatenate Strings** activity to the canvas, above the **Replace String** activity.

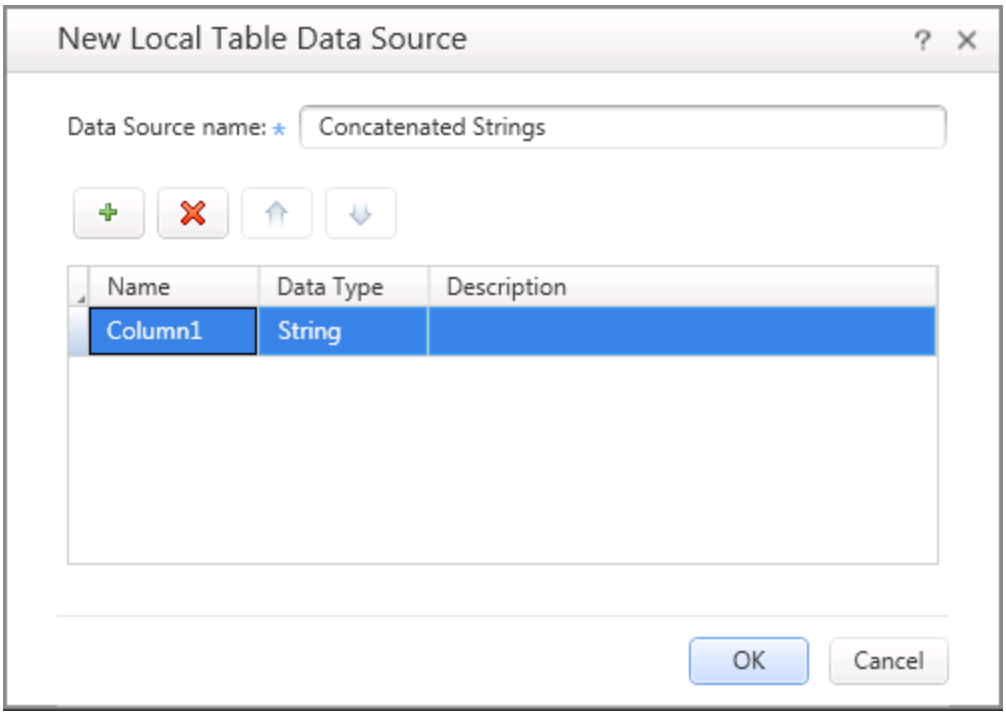
3. **Add a data source to your test.**
   a. If necessary, select **View > Data** to open the Data pane.

b. In the Data Pane, click the **New Data Source** button and select **Local Table**. The New Local Table Data Source dialog box opens:



c. In the **Data Source name** field, name the table **Concatenated Strings**.

d. Below the Data Source name field, click the **Add** button . A new row is added to the table grid.



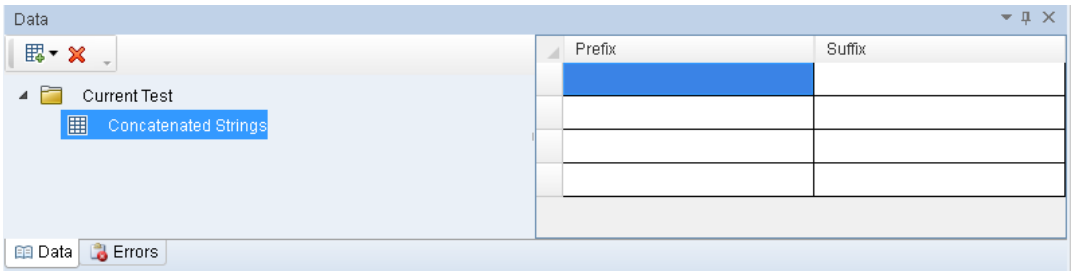
e. In the grid, in the **Name** field, enter **Prefix**. Leave the **Data Type** as **String**.

f. Click the **Add** button again. A second row is added to the table.

g. In the grid, in the **Name** field for the second row, enter **Suffix**. Leave the **Data Type** as **String**.

h. Click **OK** to close the dialog box and add the table to your test.

The table is added to your test, and is displayed in the Data pane as a sub-node of the Current Test data sources:



4. **Add values to the data table.**

In order for your test steps to use values for test steps, you must also ensure that the data source has useable data. For a locally-created and stored table, you have to add the data.

a. In the Data pane, under the **Current Test** node, select the **Concatenated Strings** node. The Data pane updates to show the data for the selected data source. (Right now, there is no data in the table.)
b. In the data grid, click in the **Prefix** column.
c. In the **Prefix** column, enter **Hello**. Make sure to leave a space after **Hello**.
d. Click in the **Suffix** column and enter **World**.
e. Enter additional rows:

Prefix	Suffix
Welcome	to UFT.
I am running	API tests.

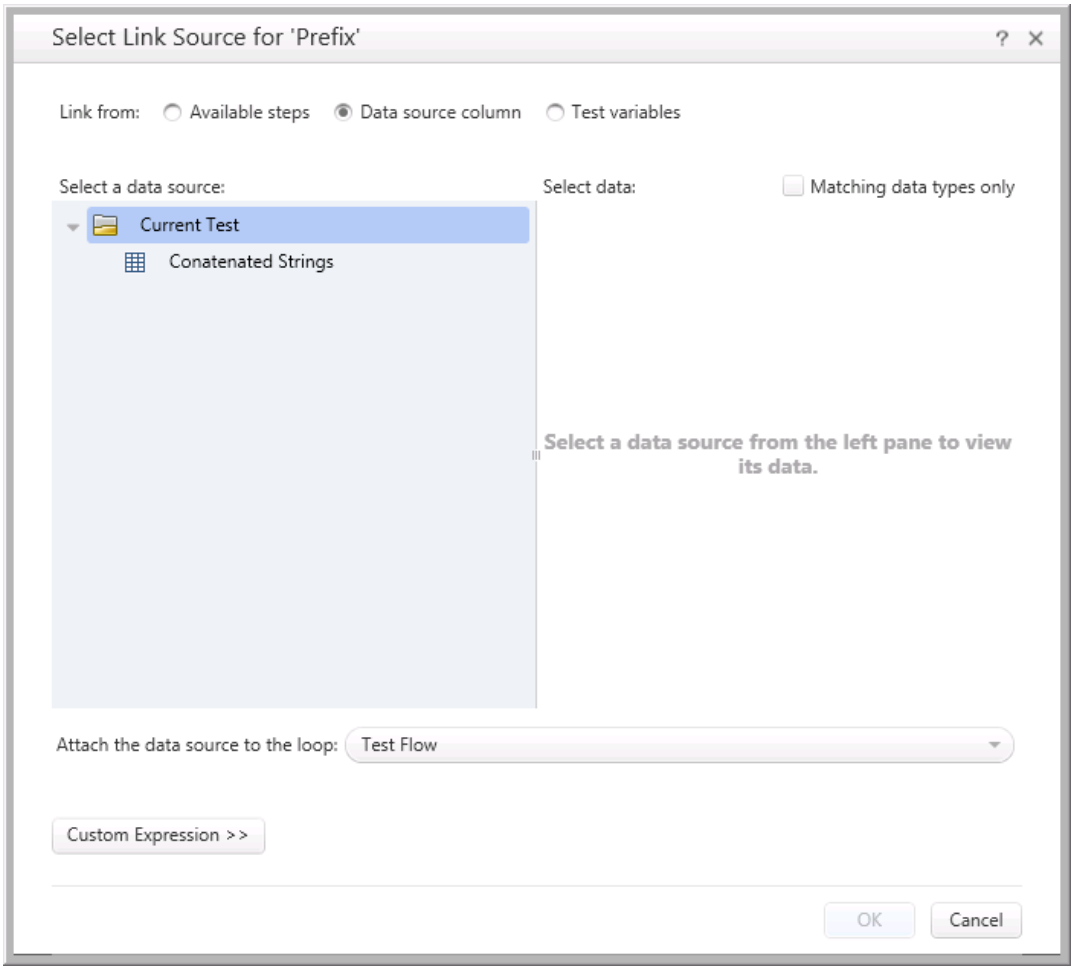
> **Note:** Make sure to enter a space after the string in the **A** column.

5. **Connect the test steps to the data source.**

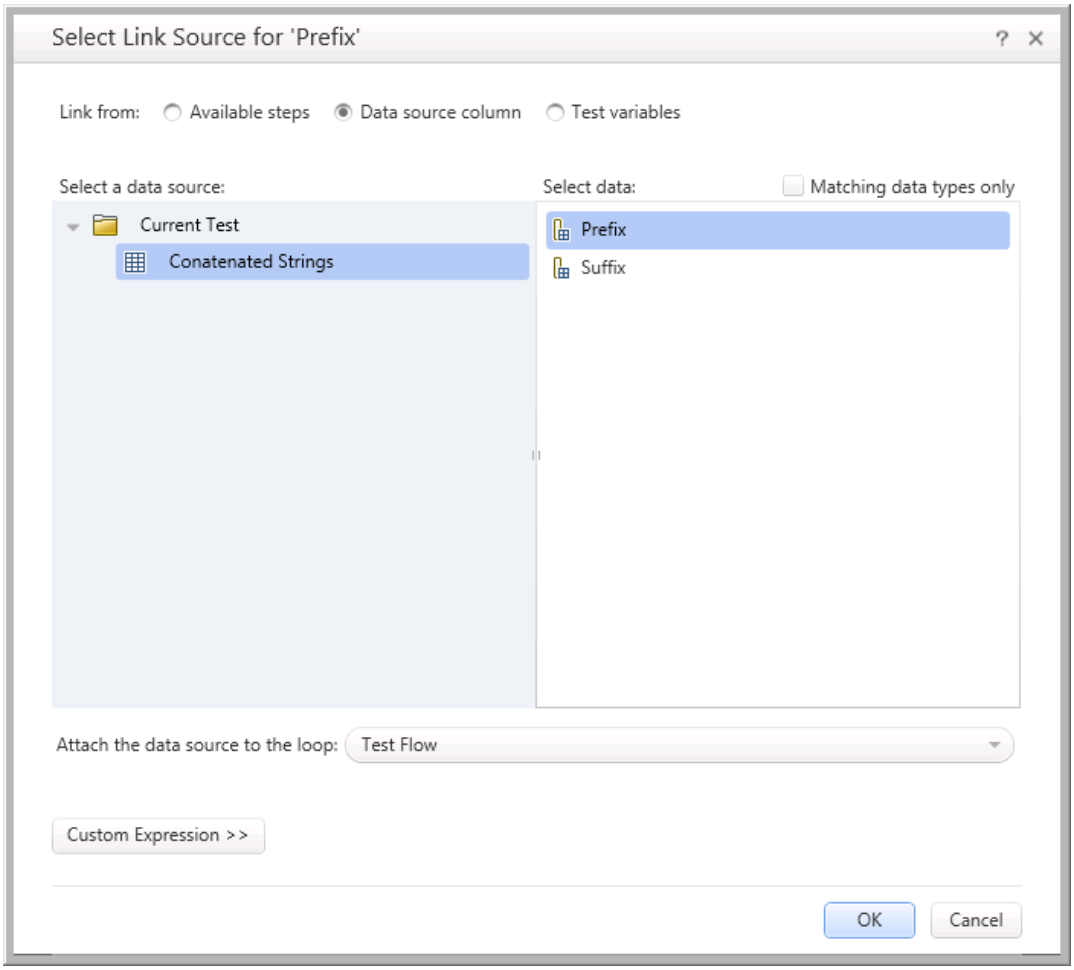
   In order to use the data source values when running the test, you need to link the test step properties to the data source.

   a. In the canvas, select the **Concatenate Strings** step. The **Input/Checkpoints** tab  opens in the Properties pane.
   b. In the Input/Checkpoints tab, in the Input section, click in the **Value** cell for the **Prefix** property.
   c. In the **Value** cell, click the **Link to data source** button . The Select Link Source dialog box opens.

d. In the Select Link Source dialog box, select the **Data source column** radio button. A list of all the test's data sources is shown in the left side of the dialog box:



e. In the **Select a data source** pane (left side), select the **Concatenated Strings** node. (This is the data source you created in the previous step.) A list of data source parameters is displayed in the right pane:



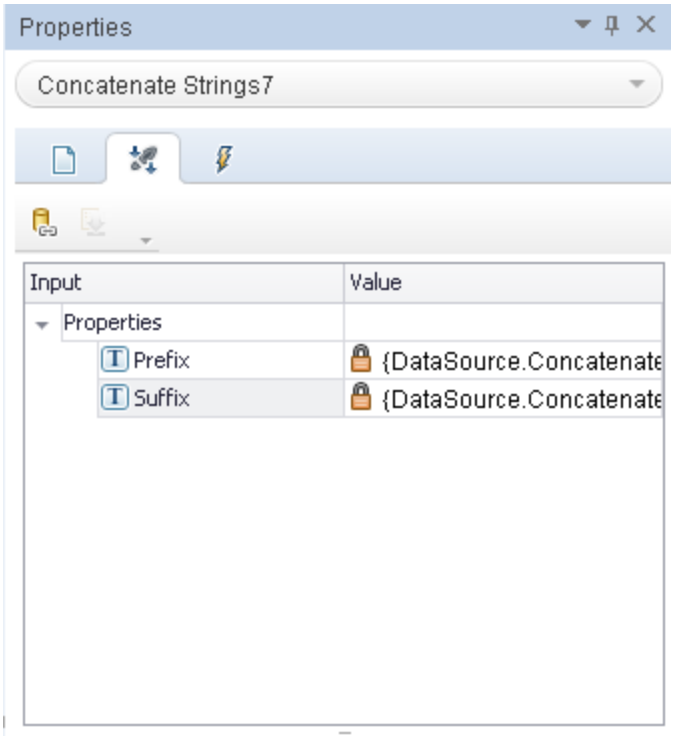
f. In the **Select data** pane (right side), select the **Prefix** node and click **OK**.

The **Value** column in the Input/Checkpoints tab is updated with an expression to show the link to the data source. If you hover over the Value column, the expression is displayed:

```
{DataSource.Concatenated Strings.Prefix}
```

g. Repeat the process described above to link the **Suffix** property to the **Suffix** column in the data table.

After you are done, the Input/Checkpoints tab reflects the links to the data table:

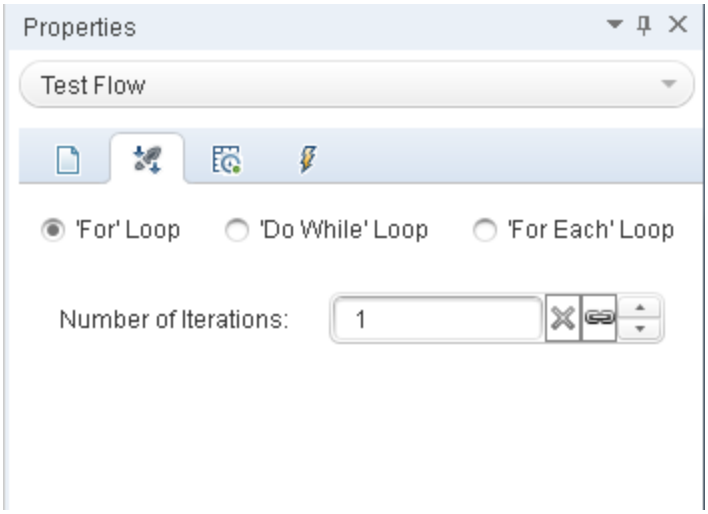


6. **Set the number of iterations for the test.**

   If you were to run the entire test after now, it would only run one iteration, using the data from the first row of the data table.

   Since you entered three rows, you should run three iterations of the test to see how UFT handles the different data.

   a. In the canvas, select the **Test Flow**. The **Input** tab opens in the Properties pane:



b. In the Input tab, select the **'For' Loop** radio button.

c. In the **Number of Iterations**, enter **3** for the number of iterations.

Now, when UFT runs the test, it will run three iterations of the test, using a new row in the data table each time.

> **Note:** You do not have to run the same number of iterations as you have rows in your data source. However, by default. UFT will start at the first row in the data table and use a new row on each subsequent iteration until the end of the test.

7. **Set the data navigation policies for the data table.**

a. In the canvas, select the Test Flow. The Input tab opens in the Properties pane.

b. In the Properties pane, select the **Data Sources** tab. The Data Navigation grid opens.

c. In the Data Sources tab, in the Data Navigation grid, in the **Data Source name** column, select the **Concatenated Strings** data source and click **Edit**. The Data Navigation dialog box opens:

d. In the Data Navigation dialog box, set the following properties:

Start at	First row
Row	1
Move by	1 rows Forward
End at	Last row
Upon reaching the last row	Wrap around

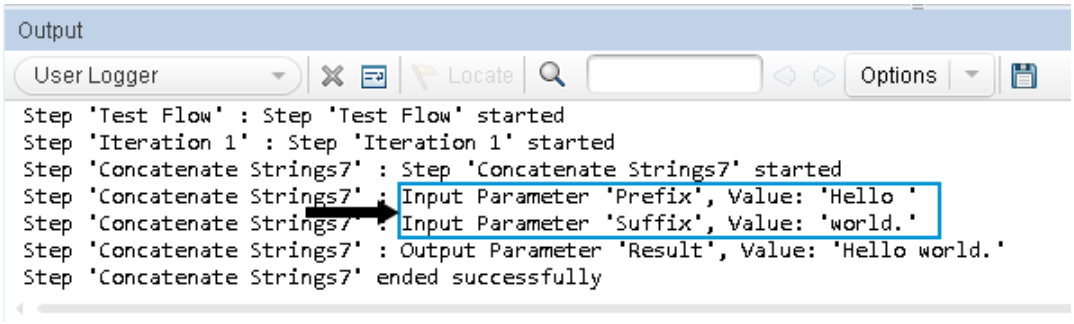
e. Click **OK** to close the dialog box and update the data navigation policies.

8. **Run the test.**

a. In the toolbar, click the **Run** button. The Run dialog box opens.

b. In the Run dialig box, click on the **Options** bar to expand the dialog box.
c. In the **Result Locations** tab, select the **Temporary run results folder** option.
d. Click **Run** to begin the test run.

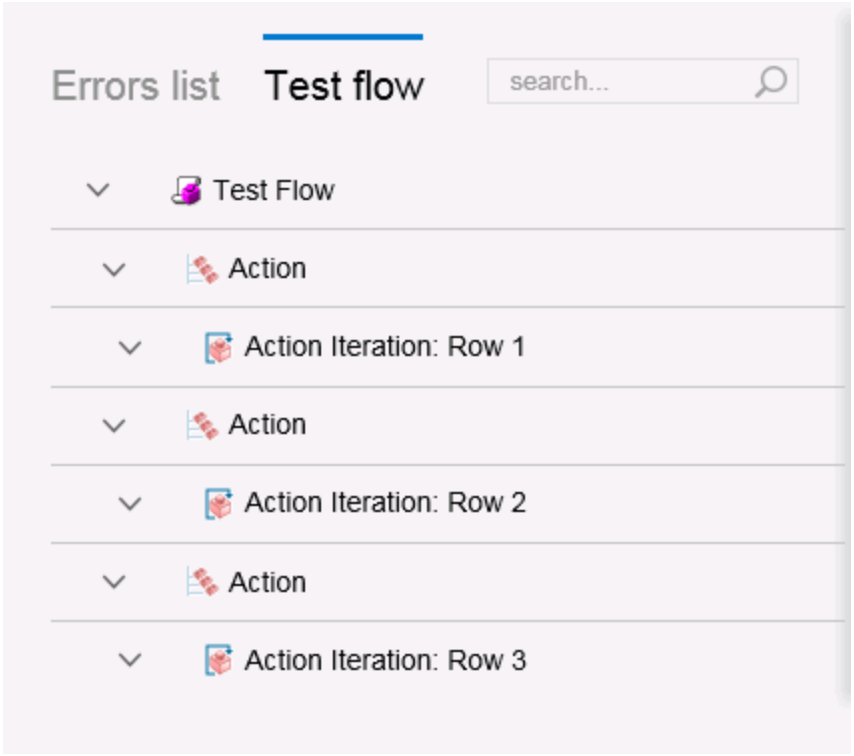
UFT runs the steps in sequence, using the values in the data table for the input of the Concatenante Strings activity. While UFT is running the test, you can see the values UFT takes from the data table in the Output pane:



After the test run is completed, the run results open.

9. **Analyze the run results.**
   a. In the run results, in the Test Flow, view the test results tree.

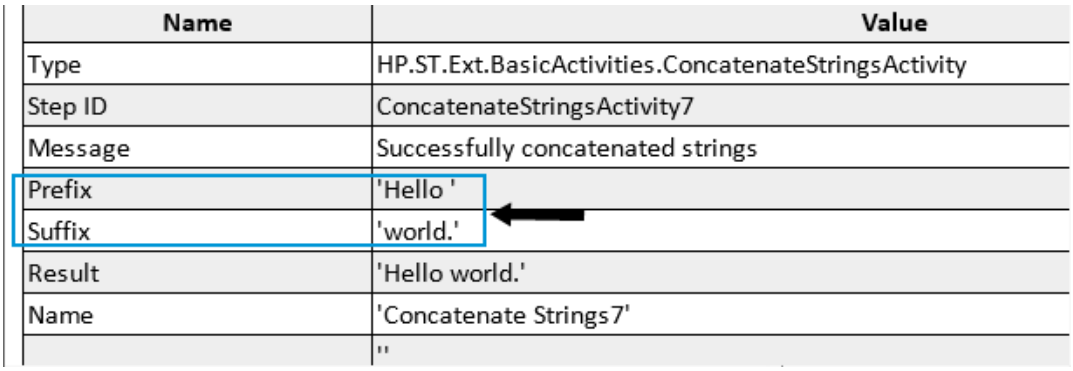
   Notice that there are three **Summary** nodes:



This corresponds to the three iterations you set earlier.

b. Expand the Action Iteration: Row 1 node,and select the **Concatenate Strings** node. The run results display a summary of the step..

c. The captured data shows the data used for the input values of the Concatenate Strings activity:



Name	Value
Type	HP.ST.Ext.BasicActivities.ConcatenateStringsActivity
Step ID	ConcatenateStringsActivity7
Message	Successfully concatenated strings
Prefix	'Hello '
Suffix	'world.'
Result	'Hello world.'
Name	'Concatenate Strings7'
	''

The values used match the first row in the data table.

d. Repeat this process for the Iteration 2 and Iteration 3 Concatenate Strings activities. You will see that the values displayed as part of the step's captured data match the second and third rows of the data table.

10. **Save the test.**

Select **File > Save**.

Now that you have learned how to connect test steps to data sources, continue with "Exercise 4b: Parameterize a test step from the output of a previous step" below to learn how to link step properties with the output of a previous step.

Exercise 4b: Parameterize a test step from the output of a previous step

In "Exercise 4a: Parameterize a test step from a data source" on page 165, you learned how to link the property values of a selected step to a data source.

However, in addition to providing the property values from data, you can also get property values from the output of previous steps. In this lesson, you will learn how to link step values using previous step outputs.

1. **Start UFT and open the Standard test.**

a. If UFT is not currently open, open it as described in "Create a solution" on page 26.

b. On the Start Page, in the **Recent tests/components** area, click **Standard**.

The Standard test you created in "Lesson 3: Create API test steps using standard activities" on page 157 opens as a separate tab in the document

pane. It is also displayed as a sub-node of the Solution Untitled solution in the Solution Explorer.

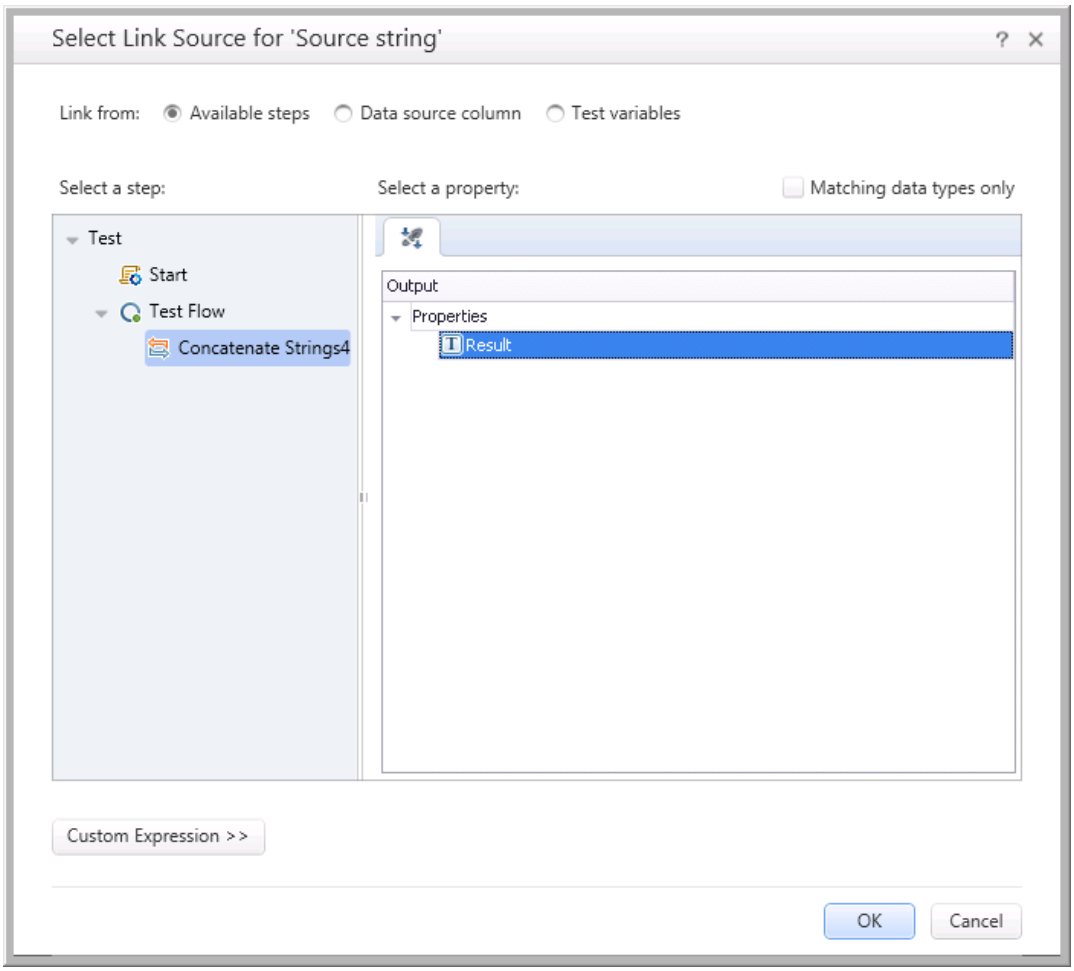
2. **Link the properties of the Replace Strings step to the Concatenate Strings test.**

   a. In the canvas, select the **Replace String** step. The **Input/Checkpoints** tab in the Properties pane opens.

   b. In the Input/Checkpoints tab, in the **Input** area, select the **Source string** row.

   c. In the **Value** cell of the Source string row, click the **Link to data source** button. The Select Link Source dialog box opens.

   d. In the Select Link Source dialog box, select the **Available steps** radio button. The dialog box then displays a list of available steps in the left side of the dialog box:

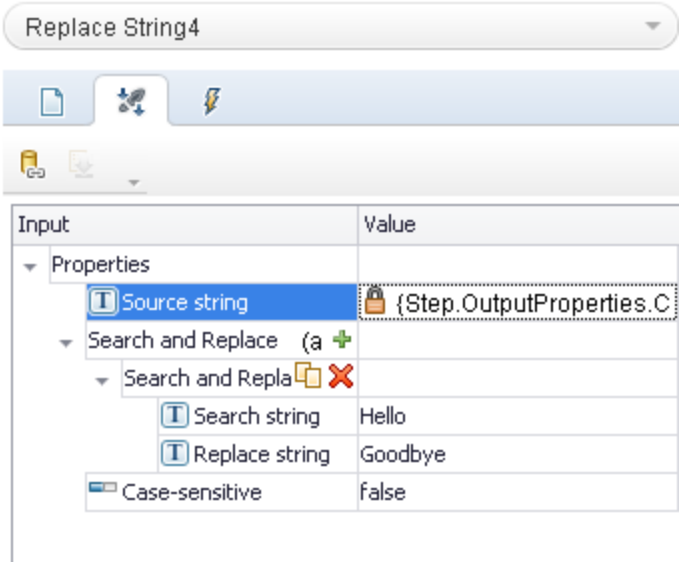


   e. In the **Select a step** pane (left side), select **Concatenate Strings**. The list of output properties for the Concatenate Strings activity is displayed.

   f. In the **Select a property** pane (right side), select the **Result** row.

g. Click **OK** to link the properties and close the dialog box.

The **Value** column in the **Source string** row (in the Properties pane) is now updated to reflect the link to the previous step's output:



If you hover over the **Value** column, you can see the full link statement:

```
{Step.OutputProperties.ConcatenateStringsActivity7.Result}
```

3. **Run the test and see the results of the linked steps.**

a. In the canvas, select the Test Flow. The **Input** tab opens in the Properties pane.

b. In the Input tab, ensure that the **'For' Loop** option is selected.

c. In the **Number of iterations**, enter `1`.

d. In the toolbar, click the **Run** button. The Run dialog box opens.

e. In the Run dialog box, in the **Results Location** tab, ensure that the **Temporary run results folder option** is selected.

f. Click **Run** to start the test run.

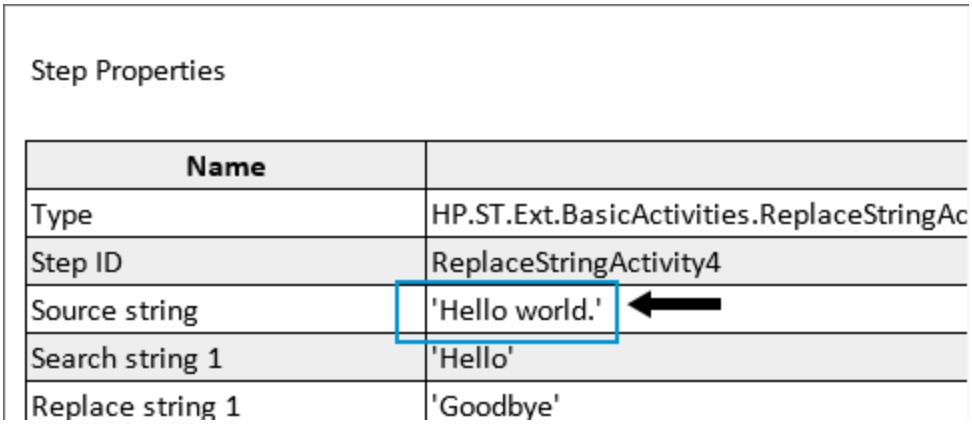
UFT runs the test steps, using the output of the Concatenate Strings step for the input of the Replace string activity.

When the test run is finished, the run results open.

g. In the run results, display the **Test Flow**.

h. Under the Summary node, find and select the **Replace String** node is visible. The run results display the details of the Replace String step.

i. In the step details, note the source string used for this test run:



The source string is the output of the Concatenate Strings activity, as entered in "Exercise 4a: Parameterize a test step from a data source" on page 165.

4. **Save the test.**

   Select **File > Save**.

Now that you have learned how to link test steps to each other, continue with "Exercise 4c: Parameterize a test with multiple sources using a custom expression" below to learn how to parameterize steps by using a combination of manually entering values, linking to a data source, and linking to the output of a previous step.

Exercise 4c: Parameterize a test with multiple sources using a custom expression

In the previous exercises, you learned how link test step property values to a data source or the output of a previous step.

However, there will be times where the step value comes from a variety of places: entering a static value manually, a data source, and/or the output of a previous step. In these cases, you can create a custom expression to link to multiple sources.

In this exercise, you will create a custom expression to write to a file the result of the string replacement operation that uses data from the test's data table, manual entry of a static text string, and the output of another test step.

1. **Start UFT and open the Standard test.**
   a. If UFT is not currently open, open it as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent tests/components** area, click **Standard**.

The Standard test you created in "Lesson 3: Create API test steps using standard activities" on page 157 opens as a separate tab in the document pane. It is also displayed as a sub-node of the **Solution Untitled** solution in the Solution Explorer.

2. **Link the input property for the Write to File test step.**

   In "Lesson 3: Create API test steps using standard activities" on page 157, you created a test with three activities:

   - **Replace String**, where you took a string and replaced a part of it with another string
   - **File Create**, where you created a file to write the replaced string
   - **Write to File**, where you wrote the replaced string

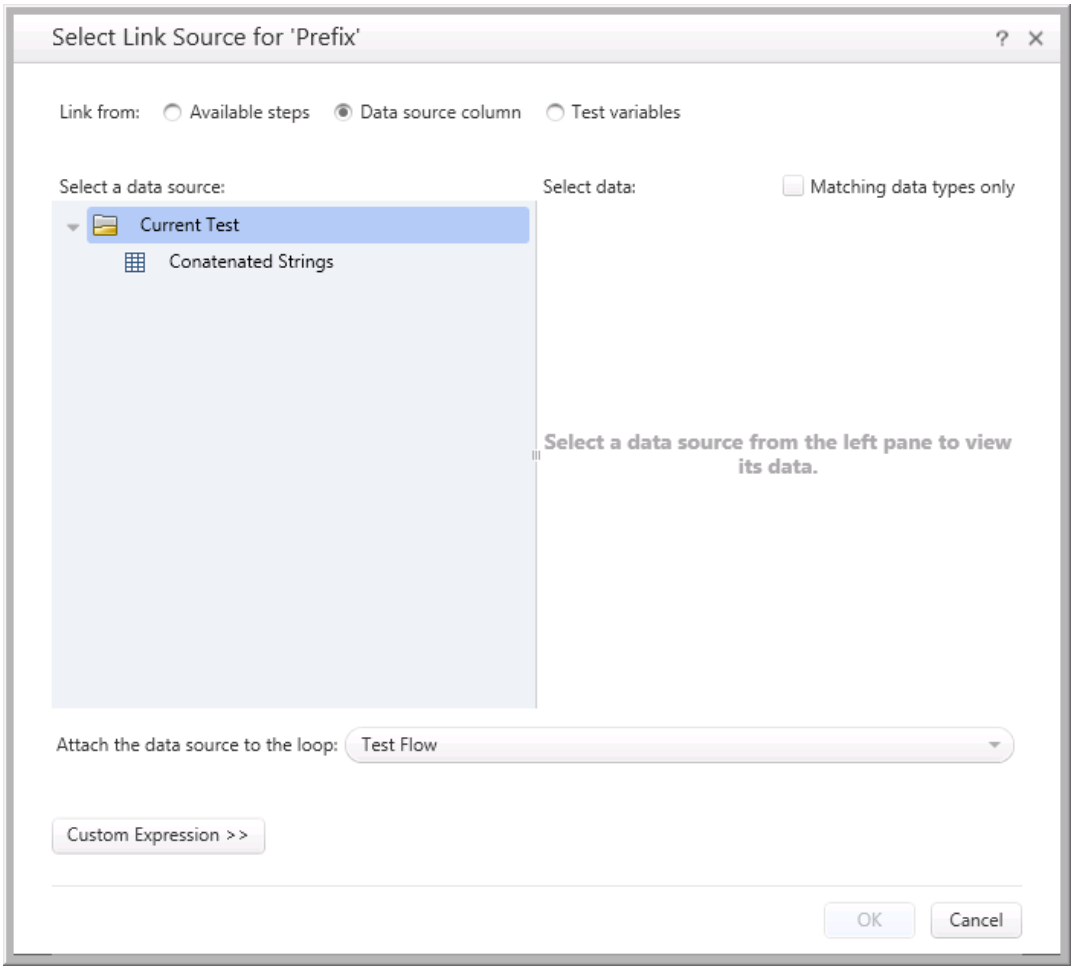
   > **Note:** You added the Concatenate Strings step in "Exercise 4a: Parameterize a test step from a data source" on page 165.

   In this and the following steps, you will work with the **Write to File** activity.

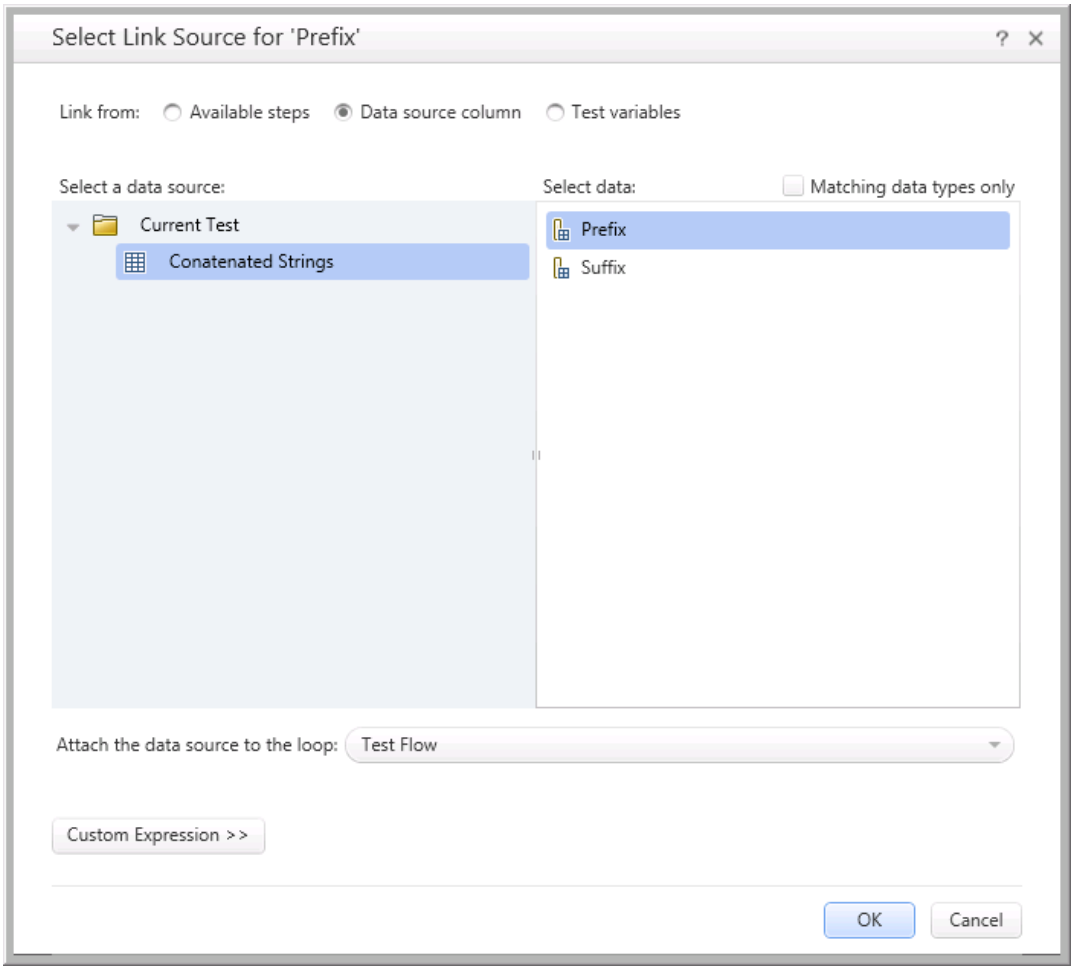
   a. In the canvas, select the **Write to File** step. The **Input/Checkpoints** tab  opens in the Properties pane.
   b. In the Input/Checkpoints tab, in the **Input** section, select the **Content** row.
   c. In the Content row, in the **Value** column, click the **Link to data source** button. The Select Link Source dialog box opens.

3. **Create the first part of the custom expression for the Content property value from the data table**

   a. In the Select Link Source dialog box, select the **Data source** column option. The Select Link Source dialog box displays a list of all the test's data sources:

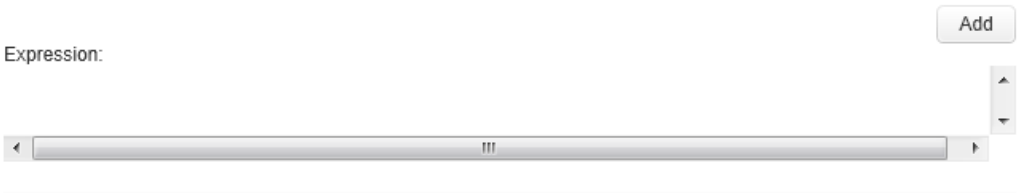

Select Link Source for 'Prefix'
Link from:
Available steps
Data source column
Test variables
Select a data source:
Select data:
Matching data types only
Current Test
Conatenated Strings
Select a data source from the left pane to view its data.
Attach the data source to the loop:
Test Flow
Custom Expression >>
OK
Cancel

b. In the **Select a data source** pane (left side), select the Concatenated Strings node. The dialog box displays a list of data parameters in the Concatenate Strings data table:

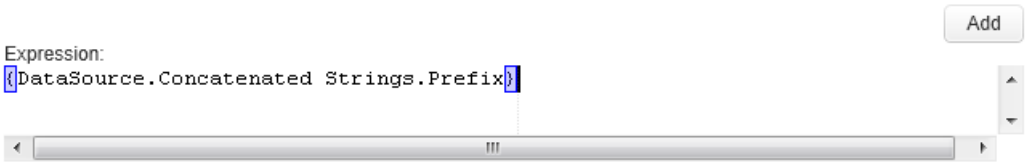


c. In the **Select data** pane (right side), select the **Prefix** node.

d. At the bottom of the dialog box, click the **Custom Expression** button. The dialog box expands to display the **Expression** area:



e. Above the Expression area, while the **Prefix** node is selected, click the **Add** button. UFT adds the expression for the Prefix parameter to the expression:



f. In the **Select data** pane, select the **Suffix** node.

g. In the Expression area, click the **Add** button again. UFT adds the expression for the Suffix parameter to the expression:

4. **Add the middle part of the custom expression by manually entering the string**

   For the middle part of the custom expression, you will manually add a static text string.

   a. In the Select Link Dialog box, in the Expression area, type a space after the **{DataSource.Concatenated Strings.Suffix}** expression.

   b. Enter the text **was replaced by**, followed by another space.

   > **Note:** Do not click add after typing this string. If you click the **Add** button, UFT adds whatever element is selected in the panes at the top part of the dialog box.

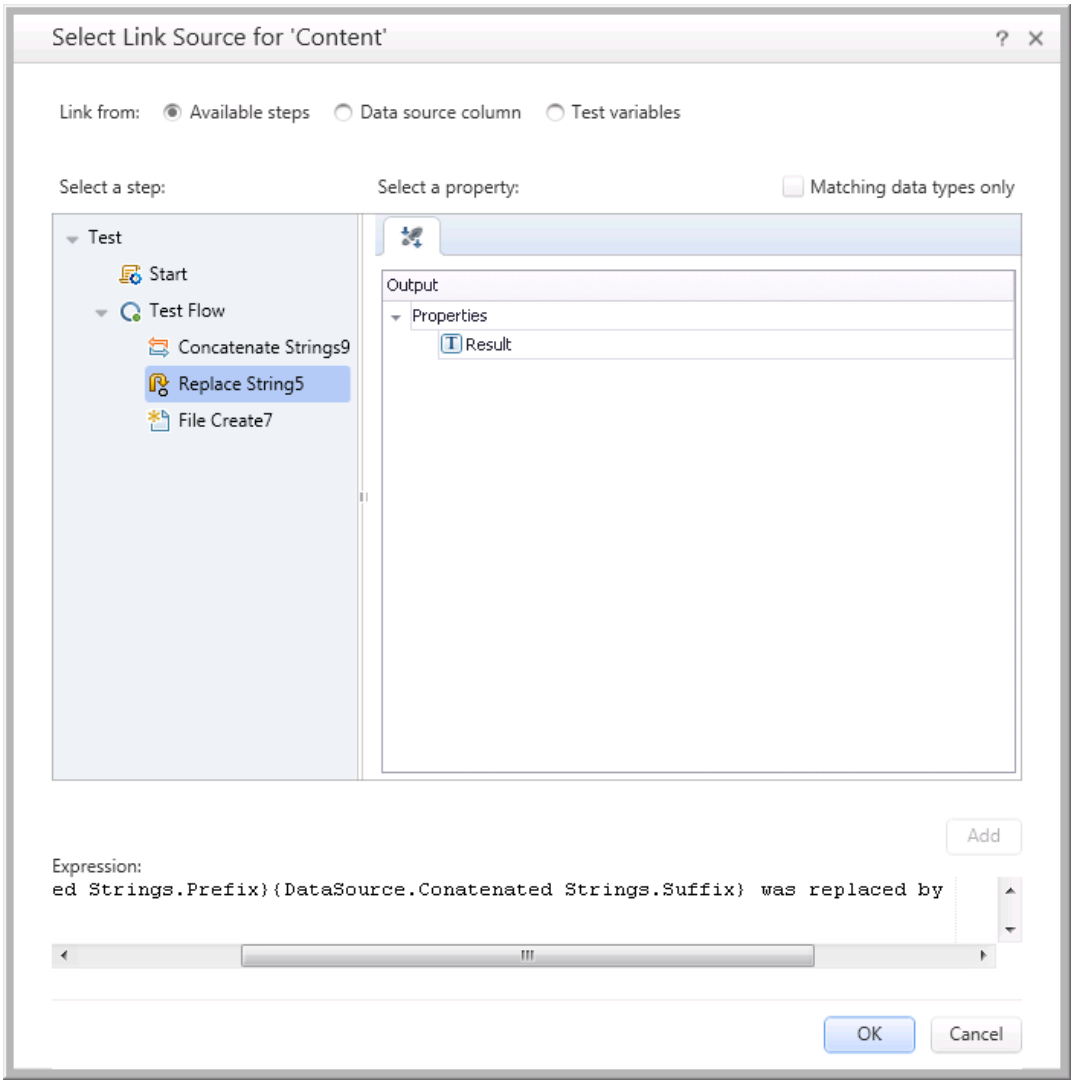
   Your custom expression should now look like this after entering the static text string:

   ```
   {DataSource.Concatenated Strings.Prefix}{DataSource.Concatenated
   Strings.Suffix} was replaced by
   ```

5. **Add the final part of the custom expression by linking it to a previous step input.**

   For the final element of the custom expression, you will link to the output of a previous step.

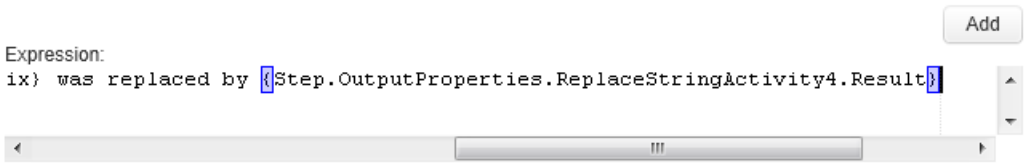
   a. At the top of the Select Link Source dialog box, select the **Available steps** option. The dialog box displays a list of all previous steps:



b. In the **Select a step** pane (left side), select the **Replace String** node. A list of available properties is displayed:

c. In the **Select a property** pane (right side), select the **Result** row.

d. In the Expression area, click **Add**. An additional expression is added to the previous expression to reflect the link to the Replace String step output:



e. Click **OK** to close the dialog box and add this expression as the value for the Content property of the Write to File step.

The **Value** column for the Content property now displays the updated expression:

```
{DataSource.Concatenated Strings.Prefix}{DataSource.Concatenated
Strings.Suffix} was replaced by
{Step.OutputProperties.ReplaceStringActivity4.Result}
```

6. **Set the number of iterations for the test run.**

   a. In the canvas, select the Test Flow. The **Input** tab opens in the Properties pane.

   b. In the Input tab, ensure that the **'For' Loop** option is selected.

   c. In the **Number of iterations**, enter **1**.

7. **Run the test and view the run results.**

   a. In the toolbar, click the **Run** button. The Run dialog box opens.

   b. In the Run dialog box, in the **Results Location** tab, ensure that the **Temporary run results option** is selected.

   c. Click **Run** to start the test run.

   UFT runs the steps, taking the values for the Content property in the Write to File step from the links you created in your custom expression.

   When the test run is finished, the run results open.

   d. In the run results, display the **Test Flow**.

   e. In the Test Flow, find the **Write to File** node. The run results display a summary of the relevant information about the step.

   f. In the step summary, click on the **Write to File.xml** link. A separate tab opens with the captured data for the step.

   g. In the separate tab, note the **Content** property value used for this test run:

Name	
Type	HP.ST.Ext.BasicActivities.FileWriteActivity
Step ID	FileWriteActivity6
Content	'Hello world. was replaced by Goodbye w
File Path	'C:\Users\LTQA\Documents\Unified Func
Encoding	'ASCII'
Mode	'Append to existing file'
Append new line	False

8. **Save the test.**

Click **Save**  .

Lesson 5: Run API tests

In "Exercise 3a: Create a test with standard activities" on page 158, you created a basic API test, using standard activities. Now that you have created this test, you can run the test.

In this lesson, you will learn how to run a test and view the run results.

This lesson includes the following:

Exercise 5a: Run a test

In "Exercise 3a: Create a test with standard activities" on page 158, you created a test using standard API testing activities. In "Lesson 4: Parameterize API test steps" on page 163, you then parameterized the test, using a number of different methods.

In this lesson, you will learn how to prepare UFT to run the test and how to run your API tests.

1. **Start UFT and open the Book Flights solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Tests** area, click **Flight Reservation Application**.
      The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Exercise 3a: Create a test with standard activities" on page 158.

2. **Set the run mode for the test.**
   Before you run your test, you must instruct UFT how you want to run the test. You can run a test in **Release** mode, which runs the test quickly, or **Debug** mode, which runs more slowly as UFT needs to load the debugging tools before the test run.
   a. Select **Tools > Options**. The Options dialog opens.
   b. In the Options dialog box, select the **API Testing** tab.
   c. In the API Testing tab, select the **General** node.
   d. In the General pane, select the **Run test in debugging mode** option.

3. **Start running your test.**
   a. In the toolbar, click the **Run** button  . The Run dialog box opens.
   b. In the Run dialog box, click on the **Options** bar to expand the dialog box.
   c. In the Options area, click the **Results location** tab. This enables you to specify where the test results are saved.
   d. In the Results location, select the **New run results folder** option. Accept the default folder name.
   e. Click **Run** to close the Run dialog box and start running the test.

As UFT runs the test, it provides a log of steps performed in the output pane, including the input and output parameters,and the result of any checkpoints run:

Any errors that occur during the test run are reported as part of the log. You can return to the relevant step to fix these errors.

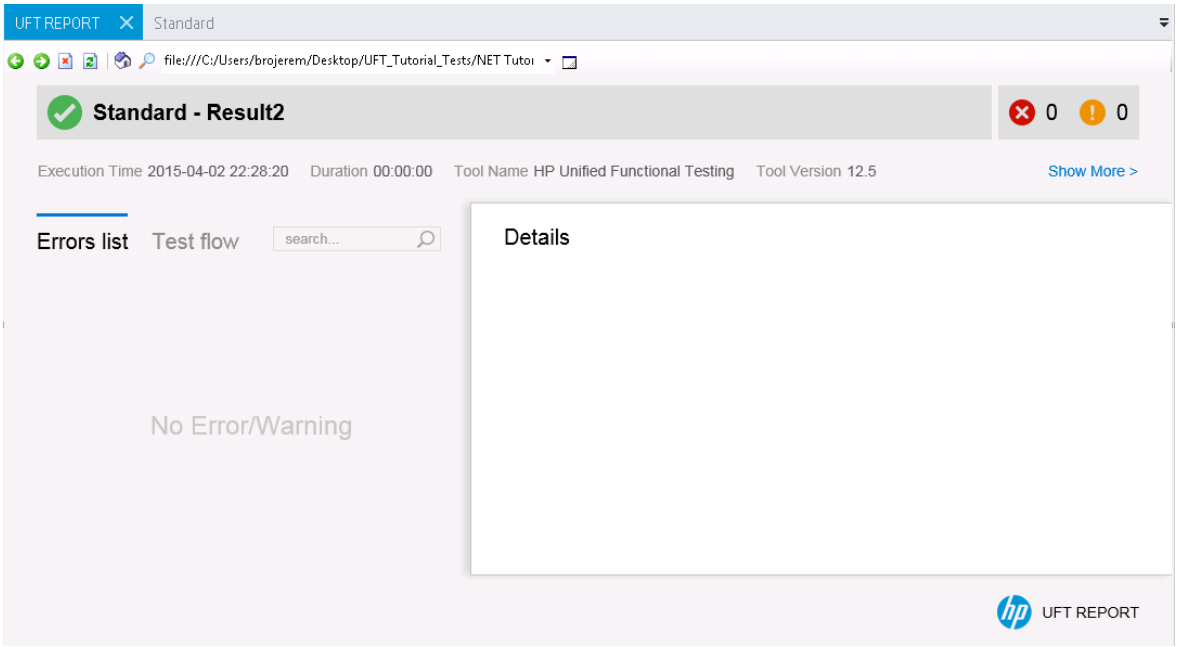
When the test run is complete, the run results open as a separate tab in the document pane. Continue to "Exercise 5b: Navigate the run results" below to learn more about the run results.

Exercise 5b: Navigate the run results

In "Exercise 5a: Run a test" on the previous page, you ran the Standard test you created. After the test run is finished, the run results automatically display the results for this test run.

Note: By default, the run results are displayed in an HTML-based report. You can also choose to have the run results displayed in the Run Results Viewer in the **Run Sessions** pane of the Options dialog box (**Tools > Options > General** tab > **Run Sessions** node). The lessons in this tutorial are based on the HTML-based report.

When the run results open, it displays the following:



Initially, the run results display the following:

Test flow	A graphical representation of the results in a tree, organized accordingly to the steps in the test. You can instruct UFT to run a test more than once using different sets of data in each run. Each run is called an iteration, and each iteration is numbered.
Error list	A list of all the errors and warnings, presented in a list.
Step summary information	A high-level results overview report, containing general information about the test, which steps passed or failed, and details about each test step. The summary also includes a link to open up the captured data for that test step.

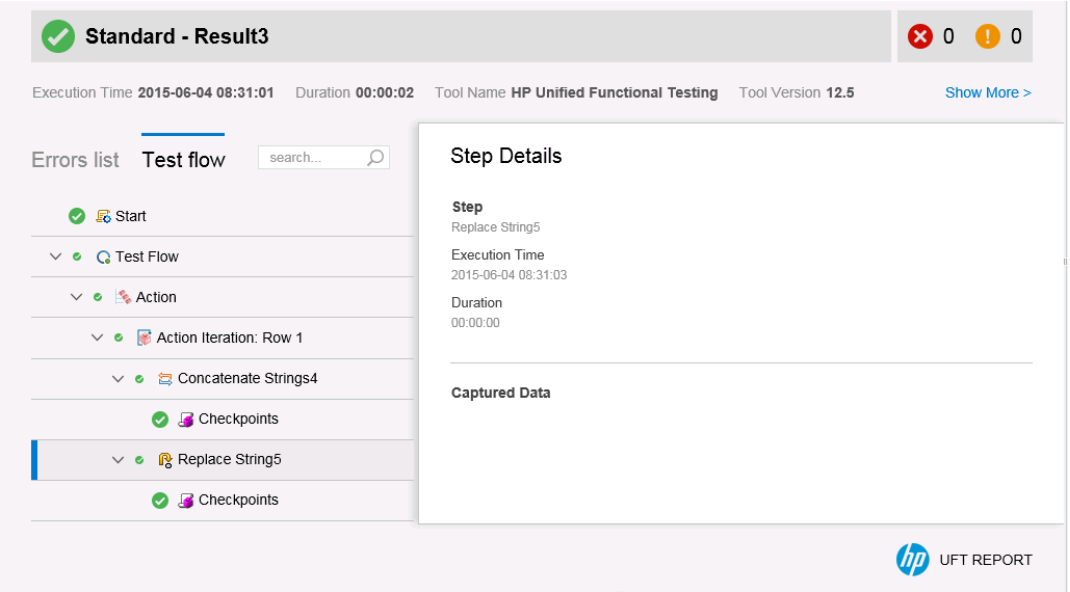
Your test run succeeded because UFT was able to perform all the steps correctly according to the steps you created and the properties you provided. If an error occured and your test did not run successfully, the error is listed in the log in the Output pane. In such cases, go back and make sure that the steps are configured exactly as described in this tutorial.

Now that you know what the run results display, continue to "Exercise 5c: Analyze the run results" on the next page to learn about the details of the run results.

Exercise 5c: Analyze the run results

In this exercise, you will inspect the steps UFT performed when running your test in "Exercise 5a: Run a test" on page 187.

1. **View the results for a specific step.**
   a. In the Test Flow, in the results tree, find the **Test Flow > Summary > Iteration 1** node to see all of the steps performed in this test.
   b. Under the Iteration 1 node, select the **Replace String** node:



   The run results now displays the following information:
   - The Test Flow, with the step highlighted
   - A summary of the test step, displaying details of the highlighted step
   - A link to view the captured data for the selected step.

2. **Close the run results.**
   In the document pane, close the tab containing the run results.

Lesson 6: Create and run API tests of Web services

In "Lesson 3: Create API test steps using standard activities" on page 157, you learned how to create a test using standard API activities.

However, there will also be times where standard activities do not match the processes your application performs. In these cases, you will need to use custom activities that you import into UFT. One of the more widely used types of service activities are Web services. In UFT, you import the service and its methods into UFT, which then makes them available for use in your tests.

In this lesson, you will learn how to create a Web service test and run the test.

This lesson includes the following:

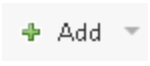
Exercise 6a: Create a Web service test

In "Create a solution" on page 26, you created a solution for the flight reservation application tests. In "Create and running automated GUI tests" on page 28, you created a variety of GUI tests that tested the performance of the flight reservation's user interface.

In this exercise, you will create the first of the API tests necessary to test the service (API) layer of the flight reservation application.

1. **Start UFT and open the flight reservation application solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Add a new API test to the solution.**
   a. In the toolbar, select the **Add** drop-down arrow 

and select **Add New Test**. The Add New Test dialog box opens.
   b. In the Add New Test dialog box, select **API Test**.
   c. In the **Name** field, name the test `Book Flights Web Service`.
   d. In the **Location** field, click the **Browse** button and navigate to the **C:\%HOMEPATH%\Unified Functional Testing** folder.
   e. Click **Add** to create the test and add it to the solution.

      The Book Flights Web Service test is added as a separate node in the Flight Reservation Application solution and opens as a separate tab in the document pane.

      > **Note:** The solution is saved automatically.

Now that you have created the test, you are ready to begin working with the Web service and its methods. Continue to "Exercise 6b: Import a Web service" below to learn how to import the Web service into your test.

Exercise 6b: Import a Web service

Before testing your Web service, you must import the service description and its methods into UFT. Typically, service descriptions are stored in a WSDL (Web

Service Description Language) file. This file defines the metadata for the service as well as the service's operations/methods. UFT then reads this WSDL file and creates the service's methods as activities in the Toolbox pane.

In this exercise, you will import the flight reservation application's service WSDL file into UFT.

1. **Start the Flights API application.**

   Start the HP Flights Service API application, as described in "Explore the flight reservation application" on page 23.

   > **Note:** Make sure that this application remains open when you are working with the tutorial, as UFT must be able to access it when editing and running the test.

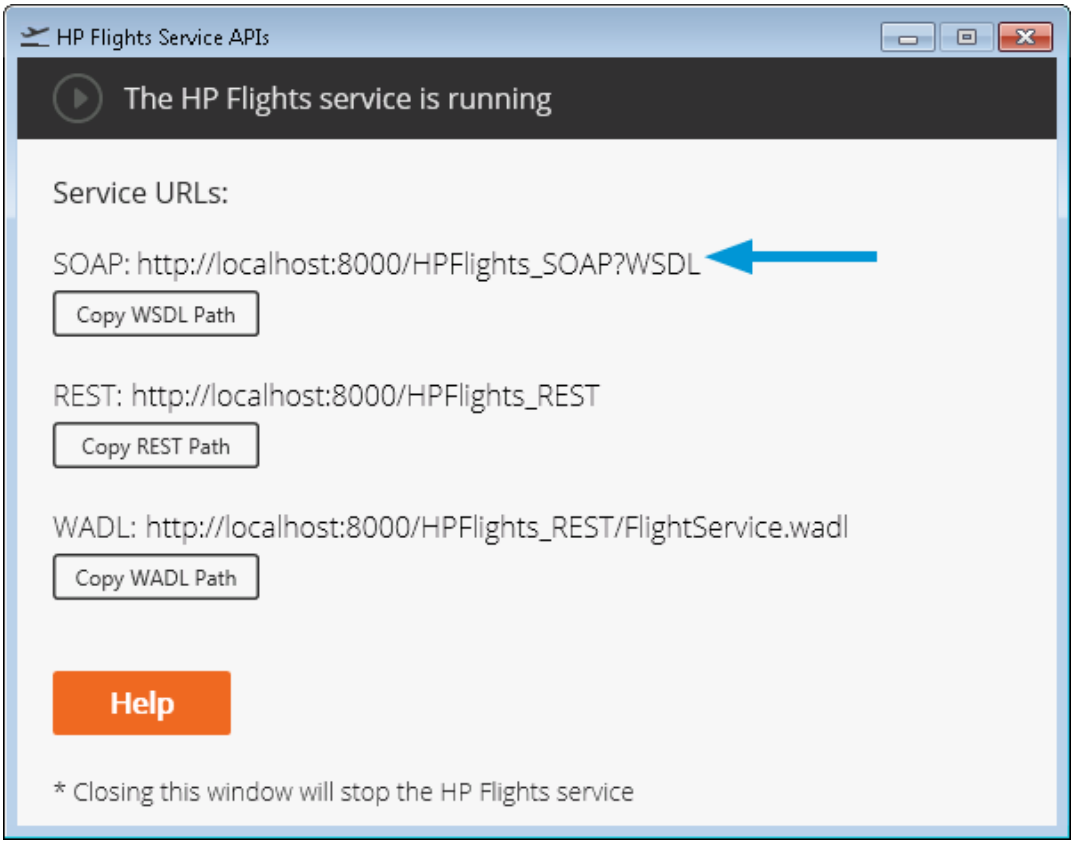
2. **Import the WSDL file.**

   a. In UFT, in the toolbar, press the **Import WSDL** button and select **Import WSDL from URL or UDDI**. The Import WSDL from URL or UDDI dialog box opens.

      > **Note:** If you had a copy of your WSDL file saved locally or in an ALM project, you can also import the file directly into UFT.

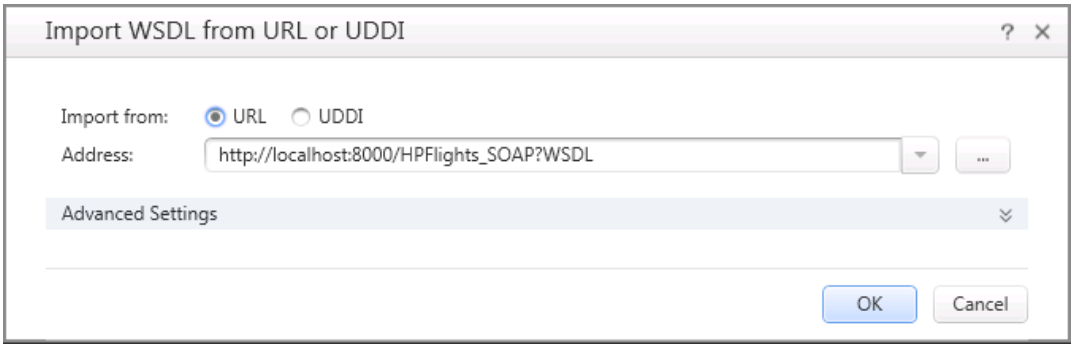
   b. In the Import WSDL from URL or UDDI dialog box, select the **URL** option.

c. In the HP Flights Service API application window, locate the URL for the SOAP-based service:



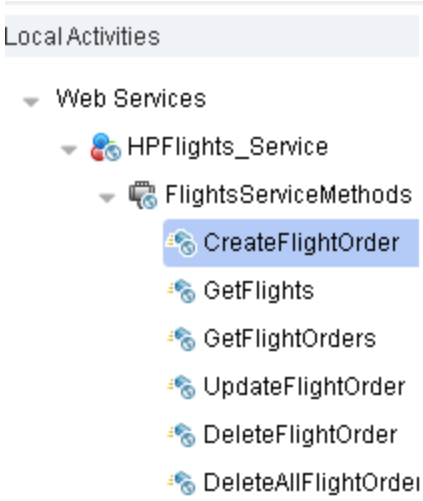
d. In the HP Flights Service API application window, click the **Copy WSDL Path** button. This saves the URL of the WSDL file so you can copy it into the Import WSDL from URL or UDDI dialog box.

e. In UFT, in the Import WSDL from URL or UDDI dialog box, in the **Address** field, paste the URL copied from the application window:



f. Click **OK** to import the service into UFT.

The service is imported into UFT and its methods are displayed in the Toolbox pane under the Local Activities section:



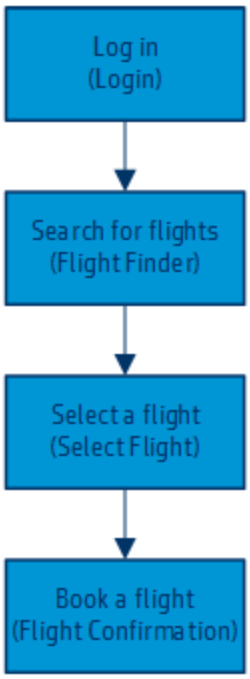
Now that you have imported the service and its methods into your test, you are ready to create a test of your Web service. Continue to "Exercise 6c: Build and parameterize a Web service test" below to build and parameterize a test of the Web service.

Exercise 6c: Build and parameterize a Web service test

In "Exercise 6b: Import a Web service" on page 192, you imported a WSDL file containing the details of your Web service. After you imported the service, UFT displayed the methods in the Toolbox pane. From the Toolbox pane, you can use any of these methods to create a test.

In this lesson, you will create a Web service test and parameterize it to see how Web service tests are created using the UFT API testing interface.

In the Book Flights GUI test that you created in "Lesson 3: Add steps to a test" on page 46, the order of the application windows was as follows:



When you create an API test of the same application, you want to make the steps match the application's flow as closely as possible. In the list of methods imported from the WSDL file, you have the following:

- **CreateFlightOrder**
- **GetFlights**
- **GetFlightOrders**
- **UpdateFlightOrder**
- **DeleteFlightOrder**
- **DeleteAllFlightOrders**

In order to match the flow of the user interface, you need to create API test steps that find the flight, and then create a flight order based on the customer input.

In this exercise, you will create two test steps: **GetFlights** and **CreateFlightOrder**.

1. **Create the test steps.**
   a. In the Toolbox pane, in the **Local Activities** section, expand the Web Services and then the **HP_Flights Services** and the **FlightsServiceMethods** nodes. The Toolbox pane displays the full list of available methods (six in all).
   b. From the list of FlightsServiceMethods, drag the **GetFlights** method to the canvas.

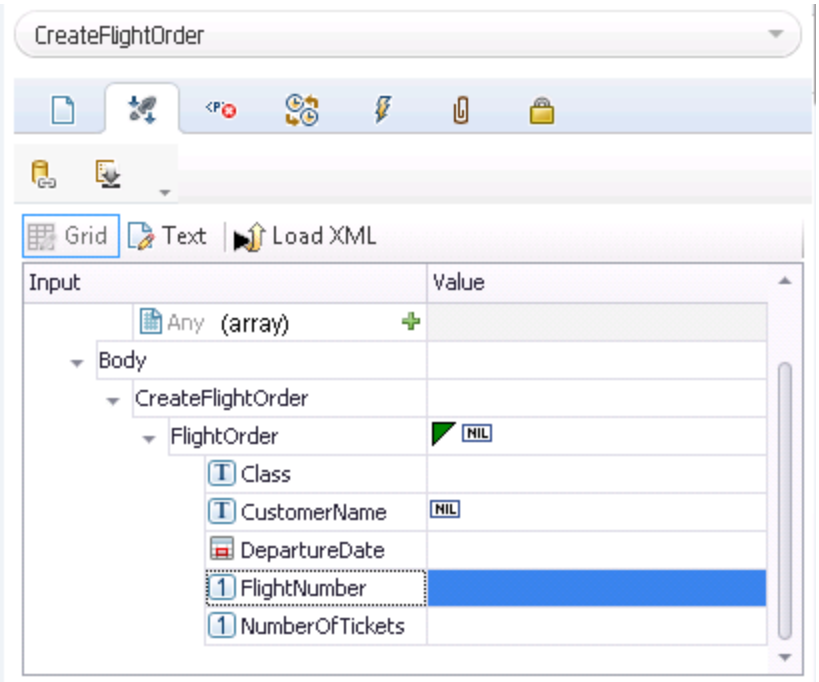
A new step block is added to the canvas, called GetFlights. The **Input/Checkpoints** tab opens in the Properties pane.

c. In the Toolbox pane, from the **FlightsServiceMethods** node again, drag the **CreateFlightOrder** method to the canvas.

2. **Link the FlightNumber property of the CreateFlightOrder step to the output of the GetFlights step.**

a. In the canvas, select the **CreateFlightOrder** step. The **Input/Checkpoints** tab opens in the Properties pane.

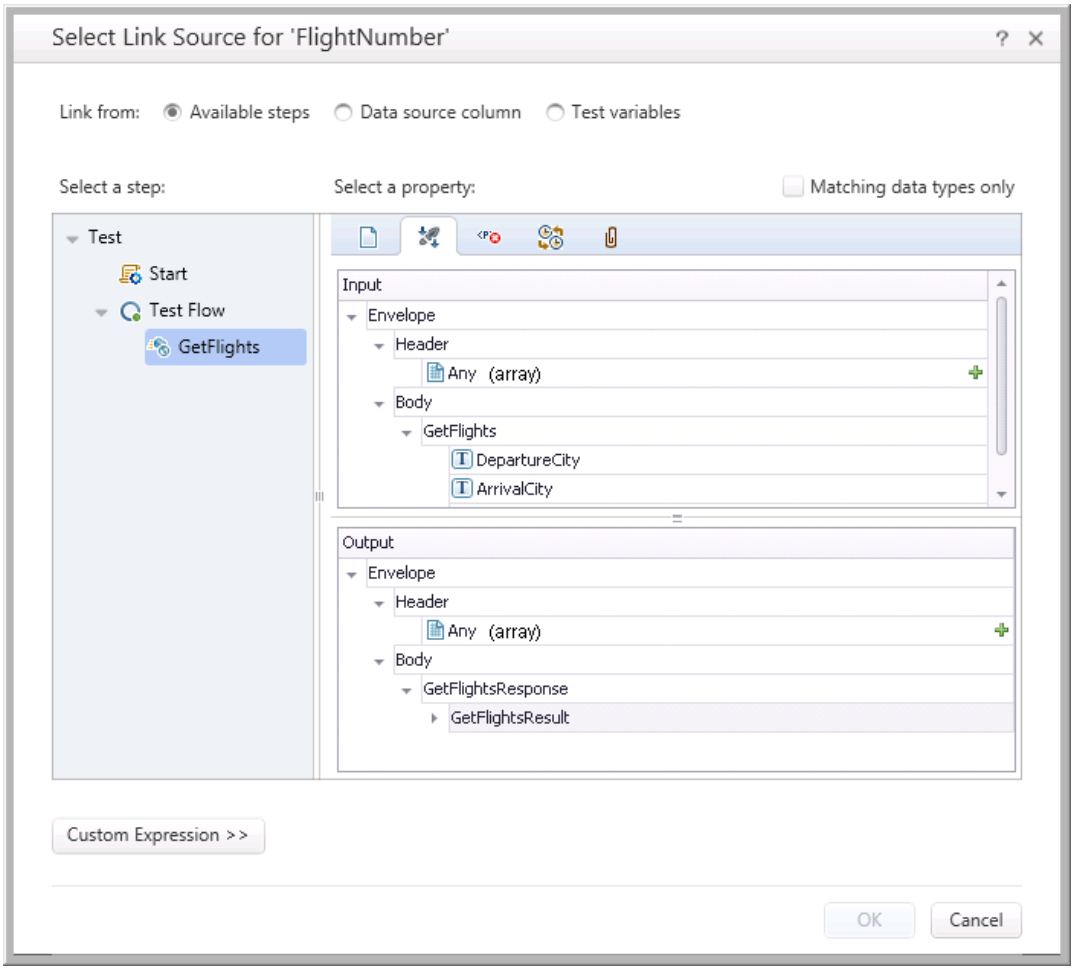
b. In the Input/Checkpoints tab, in the **Input** section, select the **FlightNumber** row.



c. In the **Value** column of the FlightNumber row, click the **Link to a data source** button . The Select Link Source dialog box opens.

d. In the Select Link Source dialog box, select the **Available steps** option. The **Select a step:** pane (left side) is updated with a list of available steps.

e. In the Select a step: pane, select the **GetFlights** step. The **Select a property:** pane (right side) is updated with the step properties:



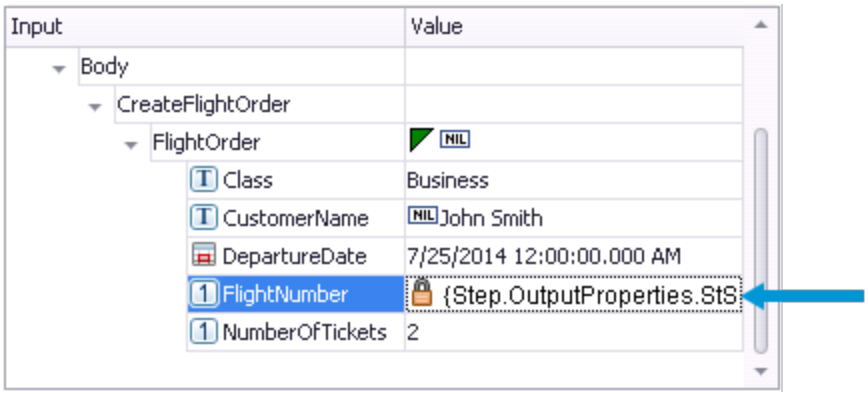
f. In the Select a property: pane, in the **Input/Checkpoints** tab, in the **Output** section, expand the **GetFlightsResult** node.

g. Under the GetFlightsResult node, in the **Flight (array)** row, click the **Add** button. A new output array is added to the output properties.

h. Under the Flight (array) row, expand the **Flight[1]** array. A list of all the output properties for the GetFlights step is displayed.

i. In the list of output properties, select the **FlightNumber** property and click **OK**. When prompted if you want to enclose the target step in a loop, select **No**.

UFT updates the **Value** column for the FlightNumber property to reflect the link:

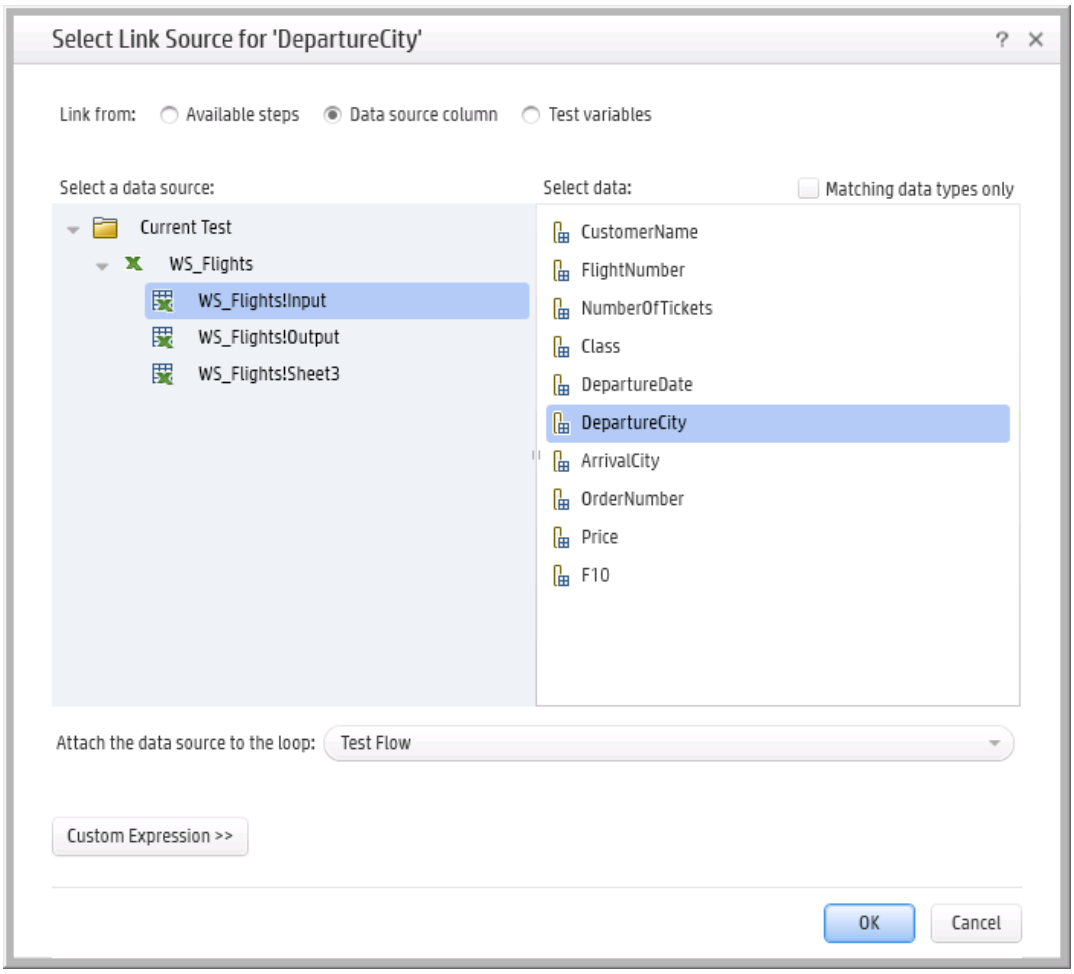


3. **Add a data source to use in your test.**
   a. If necessary, select **View > Data** to display the Data pane.
   b. In the Data pane, click the **New Data source** button and select **Excel**. The New Excel Data Source dialog box opens.
   c. In the New Data Source dialog box, in the **Excel file path** field, click the **Browse** button.
   d. In the Open dialog box, navigate to the application Excel file, saved in **<UFT installation directory>\samples\Flights Application** and click **OK**.
   e. In the New Excel Data Source dialog box, name the file **WS_Flights**.
   f. Select the **Link to the Excel file in its original location** option.
   g. Click **OK** to save the data source information and add the Excel data to your test.

4. **Link the input properties of the test steps to the data source.**
   a. In the canvas, select the **GetFlights** step. The **Input/Checkpoints** tab opens in the Properties pane.
   b. In the Input/Checkpoints tab, in the **Input** section, expand the **GetFlights** node.
   c. In the GetFlights node, select the **DepartureCity** row.
   d. In the **Value** column of the **DepartureCity** row, click the **Link to a data source** button. The Select Link Source dialog box opens.
   e. In the Select Link Souce dialog box, select the **Data source column** option. The list of data sources (in this case just the Excel file) is displayed in the **Select a data source** pane (left side).

f. In the **Select a data source** pane, select the **WS_Flights!Input** node. The list of all available data parameters (columns) is displayed in the **Select data** pane (right side).

g. In the Select data pane, choose the **DepartureCity** column:



h. Click **OK** to link the property to this column in the data table.

UFT updates the Value column with a statement showing the link to the data source.

i. Repeat the process above for the step's other properties:

- **ArrivalCity**
- **FlightDate**

j. Repeat the same process for the CreateFlightOrder step's properties:
   - **Class**
   - **CustomerName**

     > **Note:** In the **Value** column for this property, there is a blue box with **NIL** written inside. You need to click this box and remove the **NIL** (the box turns white) before linking the property.

   - **DepartureDate**
   - **NumberofTickets**

5. **Set the navigation settings for the data source.**
   a. In the canvas, select the **Test Flow** (but not a step in the test flow).
   b. In the Properties pane, select the **Data Sources** tab .
   c. In the Data Sources tab, in the list of associated data sources, select the **WS_Flights!Input** entry in the table and click **Edit**. The Data Navigation dialog box opens.
   d. In the Data Navigation dialog box, specify the data naviagation details:

Start at:	First row
Move by:	3 rows Forward
End at:	Last row
Upon reaching the last row:	Wrap around

   e. Click **OK** to assign the data navigation properties and close the dialog box.

Now that you have created a test for the flight reservation application's Web service, you can run the test and see how UFT runs and reports run results for the Web service. Continue with "Exercise 6d: Run a Web service test" below to learn more.

Exercise 6d: Run a Web service test

In "Exercise 6c: Build and parameterize a Web service test" on page 195, you created a Web service test from the imported methods and then parameterized one of the steps. In this lesson, you will run the test to see how UFT reports the run results when testing a Web service application.

1. **Start UFT and open the Book Flights solution.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.
      The Flight Reservation Application solution opens, containing the **Book Flights** test you created in "Lesson 1: Create a GUI test and actions" on page 29.

2. **Start the Flights API application.**
   If necessary, start the HP Flights Service API application, as described in "Explore the flight reservation application" on page 23.

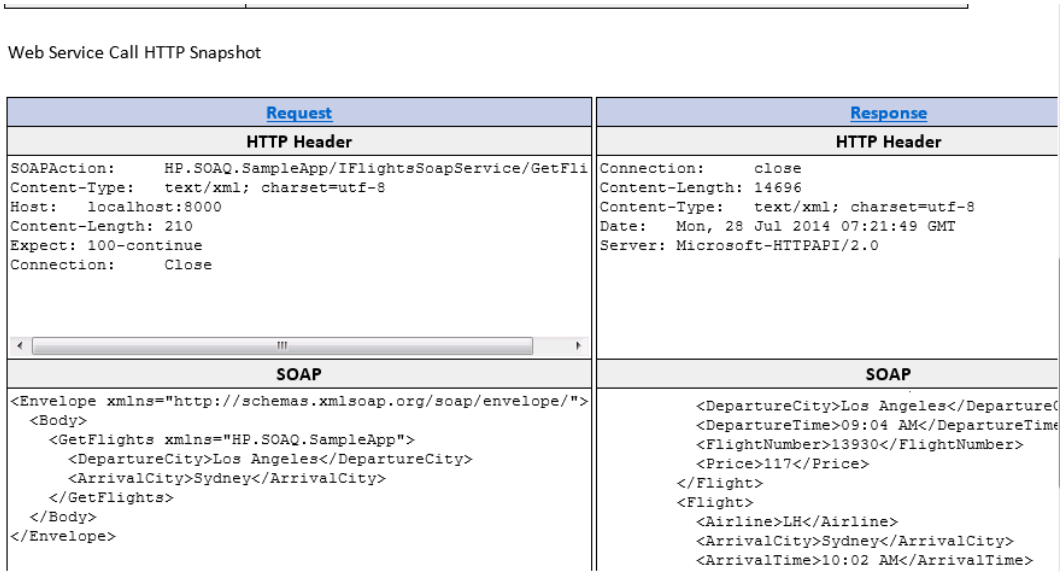
   > **Note:** Make sure that this application remains open when you are working with the tutorial, as UFT must be able to access it when editing and running the test.

3. **Set the number of iterations for the test.**
   a. In the canvas, select the **Test Flow** (but not a step in the Test Flow). The **Input** tab opens in the Properties pane.
   b. In the Input tab, select the **'For' Loop** option.
   c. In the **Number of Iterations** field, enter **4**.

4. **Run the test.**
   a. In the toolbar, click the **Run** button . The Run dialog box opens.
   b. In the Result Locations tab in the Run dialog box, ensure that the **Temporary run results folder** is still selected.
   c. Click **Run** to compile and run the test.
      After the test run is complete, the run results open.

5. **View the run results.**
   a. In the Test Flow, under any of the nodes for the iterations, select the **GetFlights** node. The step summary details are displayed.

b. In the captured, scroll down until the **Web Service Call HTTP Snapshot** area is visible:

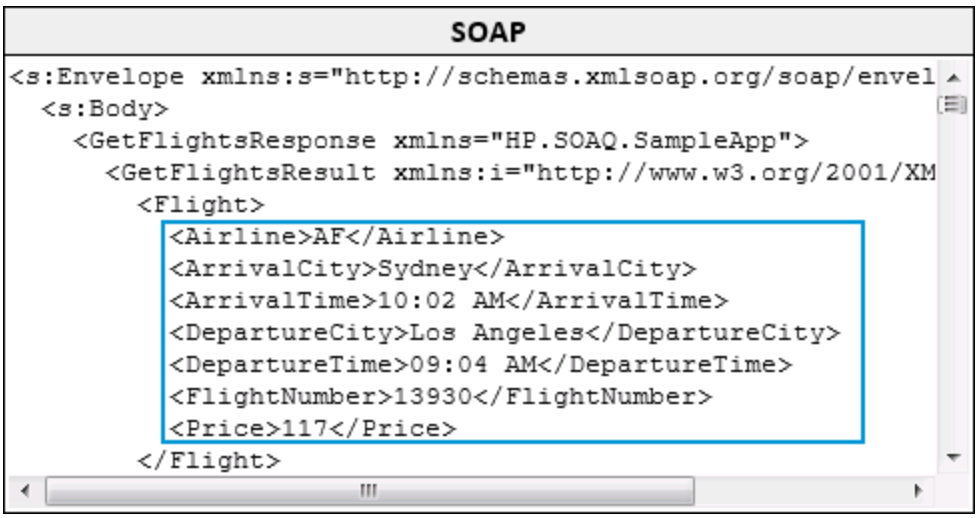


Note that in this area, UFT provides the HTTP Request and Response information for the Web Service call.

In the **SOAP** window for the **Request**, you can see the input properties sent for the GetFlights step:

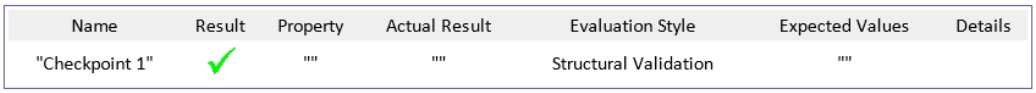
```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <GetFlights xmlns="HP.SOAQ.SampleApp">
      <DepartureCity>Los Angeles</DepartureCity>
      <ArrivalCity>Sydney</ArrivalCity>
    </GetFlights>
  </Body>
</Envelope>
```

Likewise, in the **Response** area, you can see the step's output properties:



c. Under the **GetFlights** node, select the **Checkpoints** node. The run results display a summary of the checkpoint.

d. In the step details for the checkpoint, UFT displays the result (whether the checkpoint passed or failed, the actual and expected values, and the type of checkpoint:



Name	Result	Property	Actual Result	Evaluation Style	Expected Values	Details
"Checkpoint 1"	✓	""	""	Structural Validation	""	

6. **Save the test.**

   Select **File > Save**.

Lesson 7: Create and Run API tests of REST services

In addition to testing Web services in UFT, you can use API testing to also test your REST-based services or REST-based service layers of your application. You create a prototype model of the service in UFT, and then use the created methods to structure your tests.

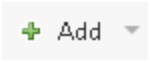
This lesson will teach you the basic steps in creating REST service prototype models and creating tests using these method models.

This lesson includes the following:

Exercise 7a: Create a REST service test

In "Lesson 6: Create and run API tests of Web services" on page 191, you created a Web service test for the API side of the flight reservation application. In this exercise, you will add a test for the REST service component of the API side of the flight reservation application.

1. **Start UFT and open the flight reservation application solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.
      The Flight Reservation Application solution opens.

2. **Add a new API test to the solution.**
   a. In the toolbar, select the **Add** drop-down arrow 

and select **Add New Test**. The Add New Test dialog box opens.
   b. In the Add New Test dialog box, select **API Test**.
   c. In the **Name** field, name the test `Book Flights REST Service`.
   d. In the **Location** field, click the **Browse** button and navigate to the **C:\%HOMEPATH%\Unified Functional Testing** folder.
   e. Click **Add** to create the test and add it to the solution.
      The Book Flights REST Service test is added as a separate node in the Flight Reservation Application solution and opens as a separate tab in the document pane.

      > **Note:** The solution is saved automatically.

Now that you have created the test, you are ready to begin working with the REST service model and its methods. Continue to "Exercise 7b: Create a REST service structure" below to learn how to create the REST service model in your test.

Exercise 7b: Create a REST service structure

Before you can use a REST Service activity in your tests, you must create a model of the necessary methods and their properties inside of UFT. UFT then takes the information for the service and methods and uses them as test steps to test the real service's performance.

1. **Start UFT and open the Book Flights solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.
   The Flight Reservation Application solution opens, containing the **Book Flights REST Service** test you created in "Exercise 7a: Create a REST service test" on the previous page.

2. **Start the Flights API application.**
   Make sure that the HP Flights Service APIs application is running, as described in "Explore the flight reservation application" on page 23.

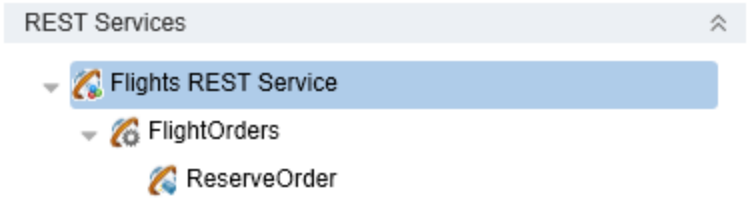
3. **Open the REST Service method properties help document.**
   In the HP Flights Service APIs window, click the **HELP** button. A browser window opens with the method information.

4. **Create a REST service model.**
   a. In the toolbar, click the **Add REST Service** button. The Add REST Service dialog box opens.
   b. In the Add REST Service dialog box, change the **New Service** name to `Flights REST Service`.

5. **Add a resource to the REST service model.**
   a. In the Add REST Service dialog box, in the toolbar, click the **Add Resource** button  . A sub-node is added to the Flights REST Service node.
   b. Change the name of the resource to **FlightOrders**.

6. **Add a method to the REST service model.**
   a. In the Add REST Service dialog box, in the toolbar, click the **Add Method** button  . A sub-node is added to the FlightOrders resource.
   b. Change the name of the resource to ReserveOrder.

Now that you have added a Service, Resource, and Method, you should have a three level hierarchy:



7. **Configure the REST Service method model URL.**

   In order for the REST Service model methods to accurately test your application, you must provide the URL of the application's service. This URL is provided at the **Service**, **Resource**, and **Method** levels.

   a. In the Add REST Service dialog box, select the **Flights REST Service** node.

      The **General** tab opens in the right pane.

   b. In the General tab, in the **Value** column for the **URL** property, enter **http://localhost:8000**.

   c. In the left pane, select the **Flight Orders** row. The **General** pane again opens in the right pane. Note that the URL you added in the General pane when the Flights REST Service was selected is displayed.

   d. In the General tab, in the **Value** column for the **Relative URL** property, enter **HPFlights_REST**.

      After you add this portion of the URL, UFT adds the **Relative URL** value to the **URL** value to make the concatenated URL: **http://localhost:8000/HPFlights_REST**.

   e. In the left pane, select the **ReserveOrder** node. The **General** pane opens in the right pane.

   f. In the right pane, select the **HTTP Input/Checkpoints** tab.

   g. In the **Value** column for the **Relative URL** property, enter **/FlightOrders**.

      UFT concatenates this part of the URL with the URL passed from the **Flights REST Service** and **Flight Orders** levels.

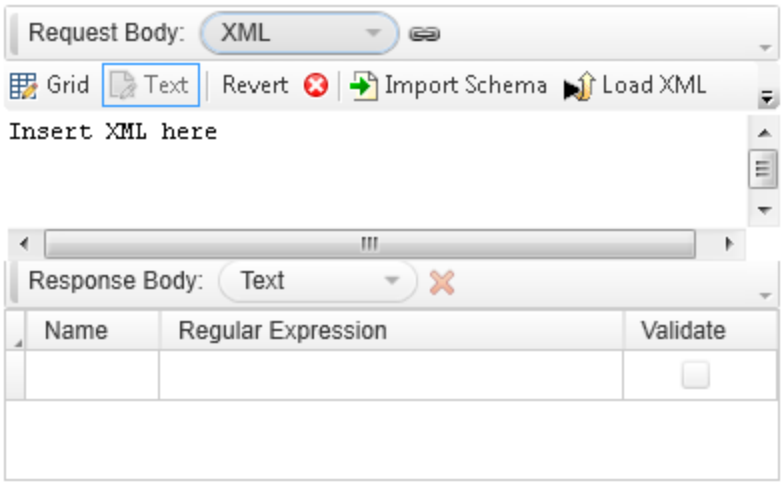
8. **Configure the additional HTTP properties for the ReserveOrders method model.**

   a. In the Add REST Service dialog box, select the **ReserveOrder** node. The General pane opens in the right pane.

   b. In the right pane, select the **HTTP Input/Checkpoints** tab.

c. In the **Value** column for the **HTTP method** property, set the HTTP type to **POST**.

9. **Add response information for the ReserveOrder method model.**
   a. In the Add REST Service dialog box, select the **ReserveOrder** node. The **General** tab opens in the right pane.
   b. In the right pane, select the **HTTP** tab.
   c. In the **Request** body section, in the **Request** drop-down list, select **XML**. A text editor opens to enable you to enter your XML:

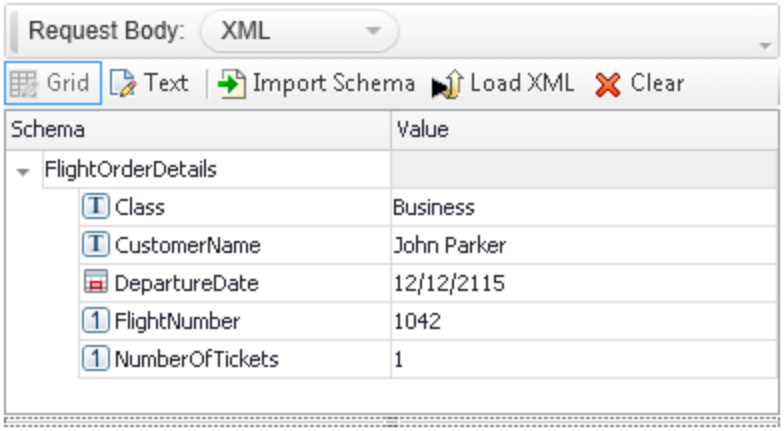


   d. Under the Request body type, ensure that **XML** is selected.
   e. In the text editor area, enter the following XML:

```
<FlightOrderDetails xmlns="HP.SOAQ.SampleApp">
  <Class>Business</Class>
  <CustomerName>John Parker</CustomerName>
  <DepartureDate>2115-05-27</DepartureDate>
  <FlightNumber>1042</FlightNumber>
  <NumberOfTickets>1</NumberOfTickets>
  </FlightOrderDetails>
```

> **Note:** You can also save this XML in a file and enter the XML by clicking the **Load XML** button.

If you click the Grid button, you can also see the properties you entered in the XML displayed in grid form:

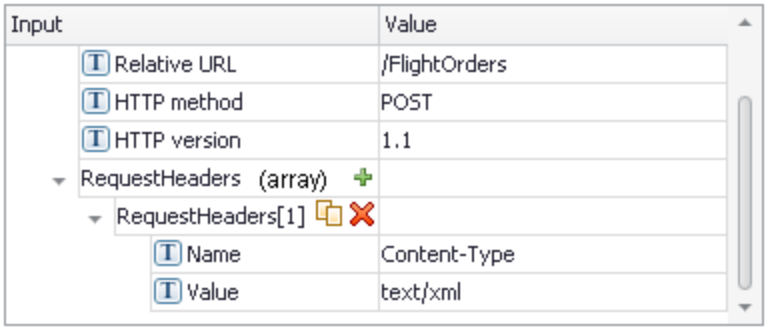


f. In the right pane, select the **HTTP Input/Checkpoints** tab again.

g. In the Input section of the HTTP Input/Checkpoints tab, expand the **Request Headers** node, and then the **Request Headers [1]** node.

Note the settings for the response:

- **Name:** Content - Type
- **Value:** text/xml



10. **Create output properties for the ReserveOrder method model.**

a. In the Add REST Service dialog box, select the **ReserveOrder** node. The **General** tab opens in the right pane.

b. In the right pane, select the **Custom Input/Checkpoints** tab.

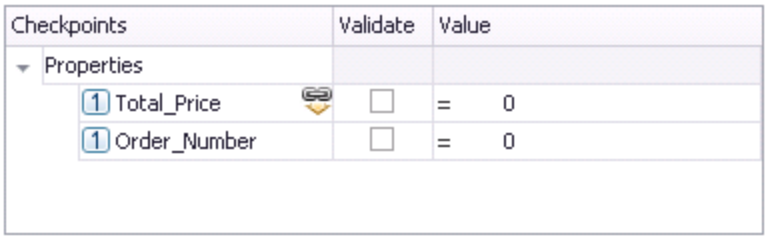
c. In the Custom Input/Checkpoints tab, click the **Add** button and select **Add Output Property**. The Add Output Property dialog box opens.

d. In the Add Output Property dialog box, in the **Name** field, enter **Total_Price**.

e. In the Type drop-down menu, select **Int** and click **OK** to add the output property. The new output property is added in the **Output** section of the Custom Input/Checkpoints tab.

f. Using the same process, add another output property named **Order_Number** of type **Int**.

The Custom Input/Checkpoints tab for the ReserveOrder method now displays all the output properties you created:

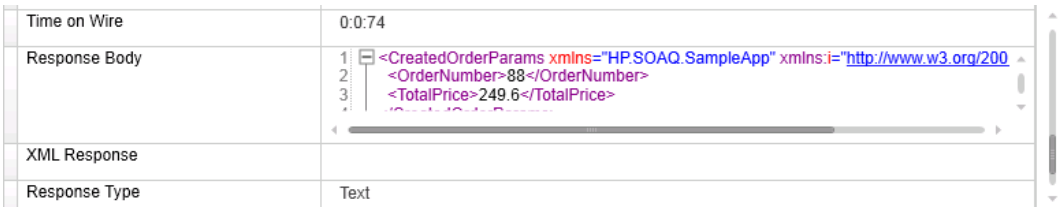


11. **Test the ReserveOrder method model.**

a. In the Add Rest Service dialog box, select the **ReserveOrder** method node.

b. In the toolbar, click the **Run Method** button .

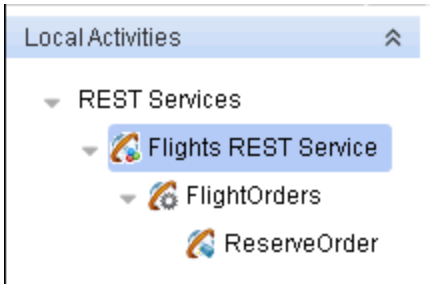
UFT runs the method and provides the results in the bottom pane of the Add REST Service dialog box:



12. **Add the service model and its methods to the Toolbox.**

In the Add REST Service dialog box, click the **OK** button.

UFT adds the REST Service model, its resources, and methods under the Local Activities node of the Toolbox pane:



From the Toolbox pane, you can drag the method to the canvas and edit the step properties.

Now that you have created the prototype model of your REST service, you are ready to create tests using the methods. Continue with "Exercise 7c: Create a Test Using REST Service Methods" on the next page to use the methods in a test.

Exercise 7c: Create a Test Using REST Service Methods

In "Exercise 7b: Create a REST service structure" on page 206, you created the prototype model of your REST Service, including methods and their properties, to use in creating test steps.

In this exercise, you will create a test using the REST Service model method in the test flow.

1. **Start UFT and open the Book Flights solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the **Book Flights REST Service** test you created in "Exercise 7a: Create a REST service test" on page 206.

2. **Import a data source to use in your test.**
   a. If necessary, select **View > Data** to display the Data pane.
   b. In the Data pane, click the **New Data source** button  and select **Excel**. The New Excel Data Source dialog box opens.
   c. In the New Data Source dialog box, in the **Excel file path** field, click the **Browse** button.
   d. In the Open dialog box, navigate to the application Excel file, saved in **<UFT installation directory>\samples\Flights Application** and click **OK**.
   e. In the New Excel Data Source dialog box, name the file **Flights_REST**.
   f. Select the **Link to the Excel file in its original location** option.
   g. Click **OK** to save the data source information and add the Excel data to your test.

3. **Create a step to test the ReserveOrder method.**
   a. In the toolbar, click the **Toolbox** button  . The Toolbox pane opens.
   b. In the Toolbox pane, in the **Local Activities** section, expand the nodes under the **Flights REST Servic**e node.
   c. Under the Local Activities node, drag the **ReserveOrder** step to the canvas.

      UFT adds a new block to the Test Flow, with the method name (**ReserveOrder**).

4. **Link the method's HTTP Request properties to the data source.**

a. In the canvas, select the **ReserveOrder** step.

b. In the Properties pane, select the **HTTP** tab .

c. In the HTTP tab, in the **Request** section, click the **Grid** button.

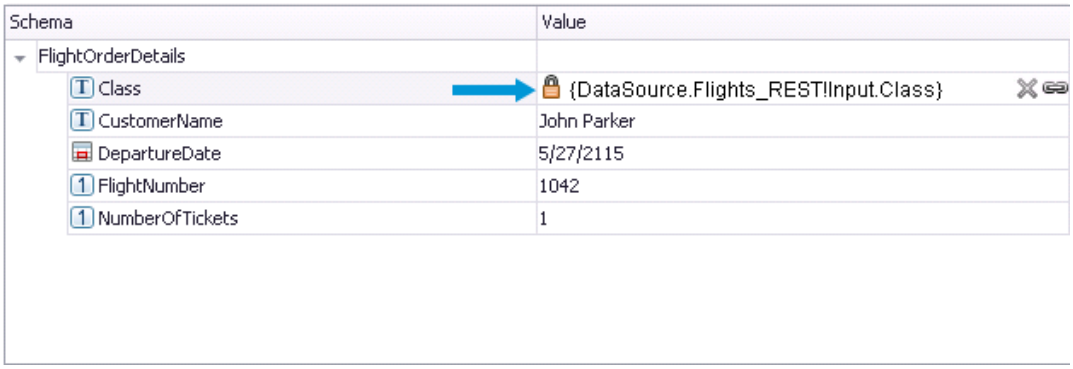
d. In the Value column of the Class property, click the **Link to a data source** button . The Select Link Source dialog box opens.

e. In the Select Link Source dialog box, select the **Data source column** option. The **Select a data source:** pane (left pane) displays a list of all available data sources.

f. In the Select a data source pane, select the **Flights_REST!Input** node. The **Select data:** pane (right pane) displays a list of all data columns/parameters.

g. In the Select data: pane, select the **Class** node and click **OK**.

UFT updates the value of the Class property to reflect the link to the data source:



h. Repeat the same process for the other HTTP Request properties:

- **CustomerName**
- **DepartureDate**
- **FlightNumber**
- **NumberofTickets**

5. **Set a checkpoint for the ReserveOrder step.**

a. In the canvas, select the **ReserveOrder** step again.

b. In the Properties pane, select the **HTTP** tab .

c. In a text editor, paste the following XML:

```
<?xml version="1.0"?>
<CreatedOrderParams xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="HP.SOAQ.SampleApp">
       <OrderNumber>93</OrderNumber>
       <TotalPrice>374.400024</TotalPrice>
```

```
</CreatedOrderParams>
```

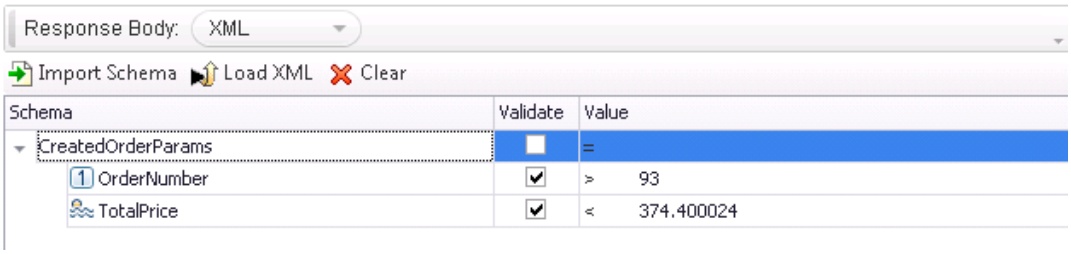
d. Save the file in the text editor as **response.xml** in a directory of your choice.

e. In UFT, in the Properties pane, in the **Response** section of the **HTTP** tab, from the drop-down list, choose **XML**.

f. In the Response Body section, click the **Load XML** button.

g. In the Open dialog box, navigate to the **response.xml** file you saved in the previous step and click **Open**.

UFT loads the XML schema from the **response.xml** file to the **Response** body section of the HTTP tab:



h. In the **Value** column for the **OrderNumber** property, click the drop-down arrow and select **>**.

i. Enter **10** for the value.

j. In the **Value** column for the **TotalPrice** property, repeat the same process and enter **<** and **500** for the value.

Now that you have created the test steps, and provided input and checkpoint properties, you are ready to run the test and view the run results. Continue with "Exercise 7d: Run a REST service Test" below to learn more.

Exercise 7d: Run a REST service Test

In "Exercise 7c: Create a Test Using REST Service Methods" on page 212, you created a test using the REST Service model methods. In this lesson, you will learn how to run the test and how to view the run results.

1. **Start UFT and open the Book Flights solution.**

   a. Open UFT as described in "Create a solution" on page 26.

   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

   The Flight Reservation Application solution opens, containing the **Book Flights REST Service** test you created in "Exercise 7a: Create a REST service test" on page 206.

2. **Set the number of iterations for the test.**

   Because your test has a data source with multiple rows, you must specify the number of iterations to run.

   a. In the canvas, select the **Test Flow**. The **Input** tab opens in the Properties pane.
   b. In the Input tab, select the **'For' Loop** option.
   c. In the **Number of Iterations** field, enter 8.

3. **Set the data navigation properties for the data source.**

   a. In the canvas, select the Test Flow. The **Input/Checkpoints** tab opens in the Properties pane.
   b. In the Properties pane, select the **Data Sources** tab. A list of all data sources associated with the test flow is displayed
   c. In the list of data sources, select the **Flights_REST!Input** data source and click **Edit**. The Data Navigation dialog box opens.
   d. In the Data Navigation dialog box, configure the following data navigation properties:

Start at:	First row
Move by:	1 rows Forward
End at:	Last row
Upon reaching the last row	Wrap around

4. **Run the test.**

   a. Ensure that the HP Flights Service APIs application is open.
   b. In the toolbar, click the **Run** button.

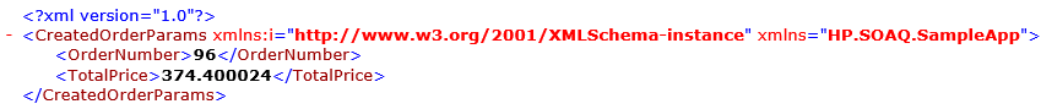
      UFT runs the test steps, providing the property values from the data source. The test run log is displayed in the Output pane.

      After the test run is over, the run results open.

5. **Analyze the run results.**

   a. In the Run Results Tree pane (left pane), expand the **Action: Book Flights REST Service** node.
   b. In the run results tree, select the **ReserveOrder** node. The step details are displayed.

c. In the captured data, note the Request and Response information. You can click on the links in the **Request Body** and **Response Body** cells to open the XML response and request information in a browser window:



d. Below the ReserveOrder node, select the **Checkpoints** node.

In the captured data, note the status of the checkpoints. In this case the checkpoints passed because the actual values were within the limits of the expected values:

Name	Result	Property	Actual Result	Evaluation Style	Expected Values	Details
"Checkpoint 1"	✓	"CreatedOrderParams [1]/OrderNumber[1]"	"96"	>	"10"	""
"Checkpoint 2"	✓	"CreatedOrderParams [1]/TotalPrice[1]"	"374.400024"	<	"500"	""

Exercise 7e: Resolve a REST service conflict

In "Exercise 7b: Create a REST service structure" on page 206, you created a prototype REST service model with the method ReserveOrder. This method had specific properties, such as the URL and property names. If the service model's properties changed after you created a test, your test would no longer match the model. As a result, UFT has a Resolve Conflict wizard to enable you to resolve changes in the method's properties.

In this exercise, you will use the Resolve Conflict wizard to help resolve these differences.

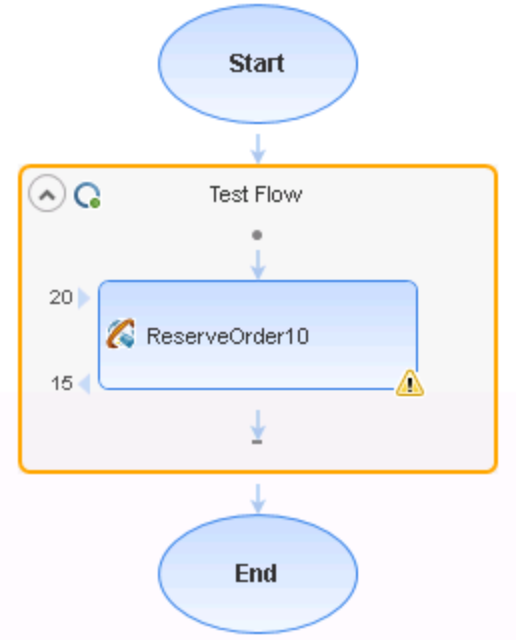
1. **Start UFT and open the Book Flights solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

   The Flight Reservation Application solution opens, containing the **Book Flights REST Service** test you created in "Exercise 7a: Create a REST service test" on page 206.

2. **Edit the service model properties.**
   a. In the toolbar, click the **Toolbox** button  .

b. In the Toolbox, in the **Local Activities** section, expand the nodes under the **REST Services** node.
c. Right-click the **Flights REST Service** node and select **Edit Service**. The Edit REST Service dialog box opens.
d. In the Edit REST Service dialog box, select the **ReserveOrder** node. The **General** tab opens in the right pane.
e. In the right pane, select the **Custom Input/Checkpoints** tab.
f. In the Custom Input/Checkpoints tab, in the **Checkpoints** section, select the **Total_Price** property and click the **Edit Property** button. The Edit Property dialog box opens.
g. In the Edit Property dialog box, change the name of the property to **TotalPrice** and click **OK**. The property name is modified in the Checkpoints section.
h. Repeat the same process to change the **Order_Number** property to **OrderNumber**.
i. In the Edit REST Service dialog box, click **OK** to save the changes to your service.

   UFT saves the changes to the REST Service model. However, in the ReserveOrder step in the canvas, an alert is displayed:

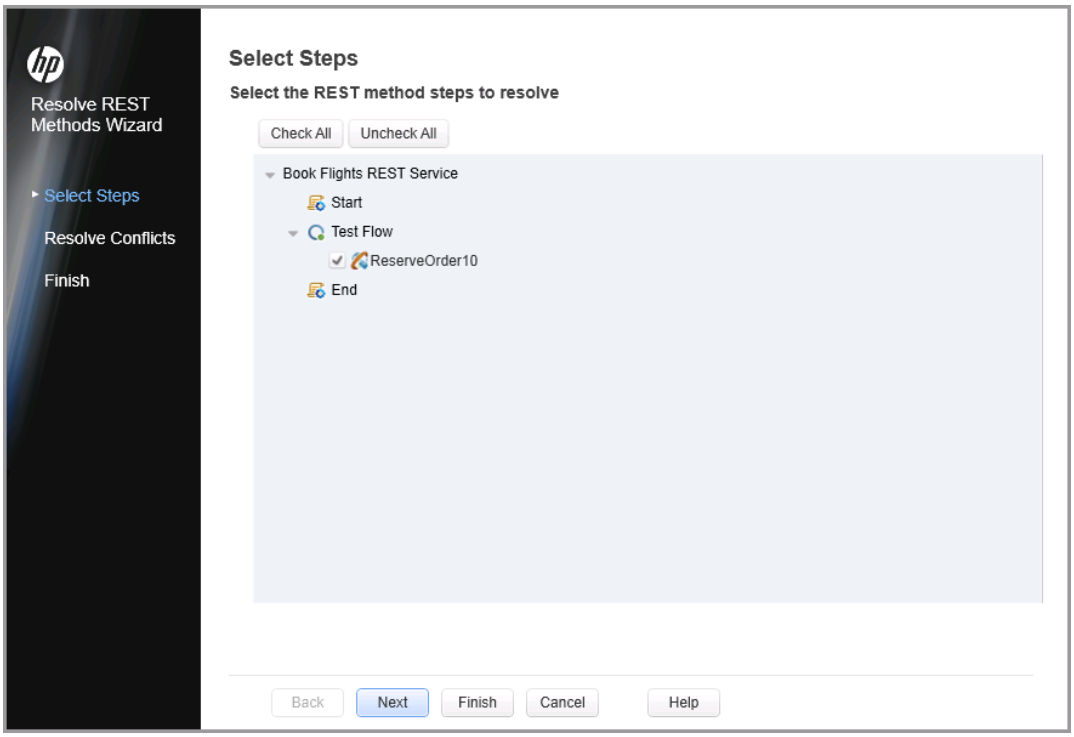


   This indicates that there is a conflict between the service model and the step in the test, that must be resolved.

3. **Resolve the conflicts.**

a. In the canvas, click on the Alert icon. UFT displays a message: `This step should be resolved. Resolve step.`

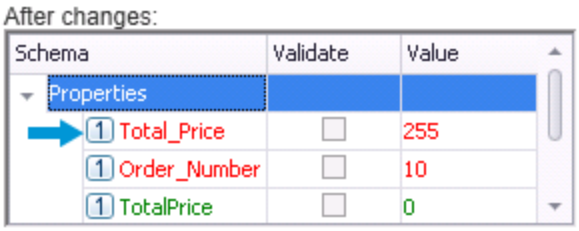
b. Click on the alert text. The Resolve REST Methods wizard opens:



c. In the Select Steps screen (first screen), select the checkbox for the **ReserveOrder** step and click **Next**. The Resolve Conflicts screen opens.

> **Note:** When using this wizard, the Select Steps screen displays all the steps in which there are conflicts. As a result, you can resolve multiple step conflicts at once.

d. In the Resolve Conflicts screen, in the **Output Properties** section (bottom area), in the **After changes** box, select the **Total_Price** property (colored red):



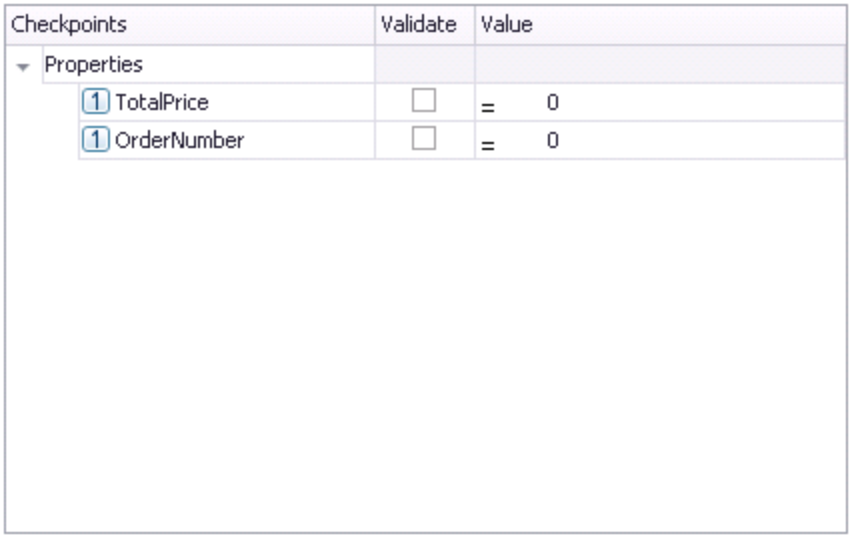
e. Click **Remove**.

f. Repeat the same process to remove the **Order_Number** property (also colored red).

g. Click **Next** to continue. The Finish screen opens.

h. The Finish screen displays the status of the conflicts in your service model. In this case, there are no existing conflicts.

   Click **Finish** to exit the wizard. UFT updates your test with the changes selected in the Resolve REST Methods wizard. In this exercise, the output properties for the **ReserveOrder** step are now updated in the Properties pane:



4. **Save the test.**

   Select **File > Save**.

Lesson 8: Create and run API tests of Web Application Services (WADLs)

In "Lesson 3: Create API test steps using standard activities" on page 157, you learned how to create a test using standard API activities.

However, there will also be times where standard activities do not match the processes your application performs. In these cases, you will need to use custom activities that you import into UFT. One of the other types of service activities are Web application services. In UFT, you import the service description and its methods into UFT, which then makes them available for use in your tests.

In this lesson, you will learn how to import and use a Web application service description into UFT.

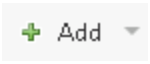
This lesson includes the following:

Exercise 8a: Create a test for a Web Application Service

In "Create a solution" on page 26, you created a solution for the flight reservation application, to which you added GUI tests of the application's user interface, and API tests of the application's Web services and REST services.

In this exercise, you will create a test for the Web Application service of the flight reservation application.

1. **Start UFT and open the flight reservation application solution.**
   a. Open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solution** area, click **Flight Reservation Application**.
   The Flight Reservation Application solution opens in the Solution Explorer.

2. **Add a new API test to the solution.**
   a. In the toolbar, select the **Add** drop-down arrow 

and select **New Test**.
   b. In the Add New Test dialog box, select **API Test**.
   c. In the **Name** field, name the test `Flights WADL`.
   d. In the **Location** field, click the **Browse** button and navigate to the **C:\%HOMEPATH%\Unified Functional Testing** folder.
   e. Click **Add** to create the test and add it to the solution.
   The Book Flights WADL test is added as a separate node in the Flight Reservation Application solution and opens as a separate tab in the document pane.

> **Note:** The solution is saved automatically.

Now that you have a created a test, you are ready to begin working with the Web Application service model and its methods. Continue to "Exercise 8b: Import a Web Application service model" below to learn how to import the service description into UFT.

Exercise 8b: Import a Web Application service model

Before you test your Web Application service, you must import the service description (including its structure of the resources and its methods). Web Application service descriptions are stored in a WADL (Web Application Description Language) file. UFT reads this file, and then creates a hierarchy of

service, resources, and methods (similar to a REST service hierarchy). Once you have imported the service description, you can use the methods to create a test.

In this exercise, you will import the flight reservation application's service WADL file into UFT.

1. **Start the Flights API application.**

   Start the HP Flights Service API application, as described in "Explore the flight reservation application" on page 23.

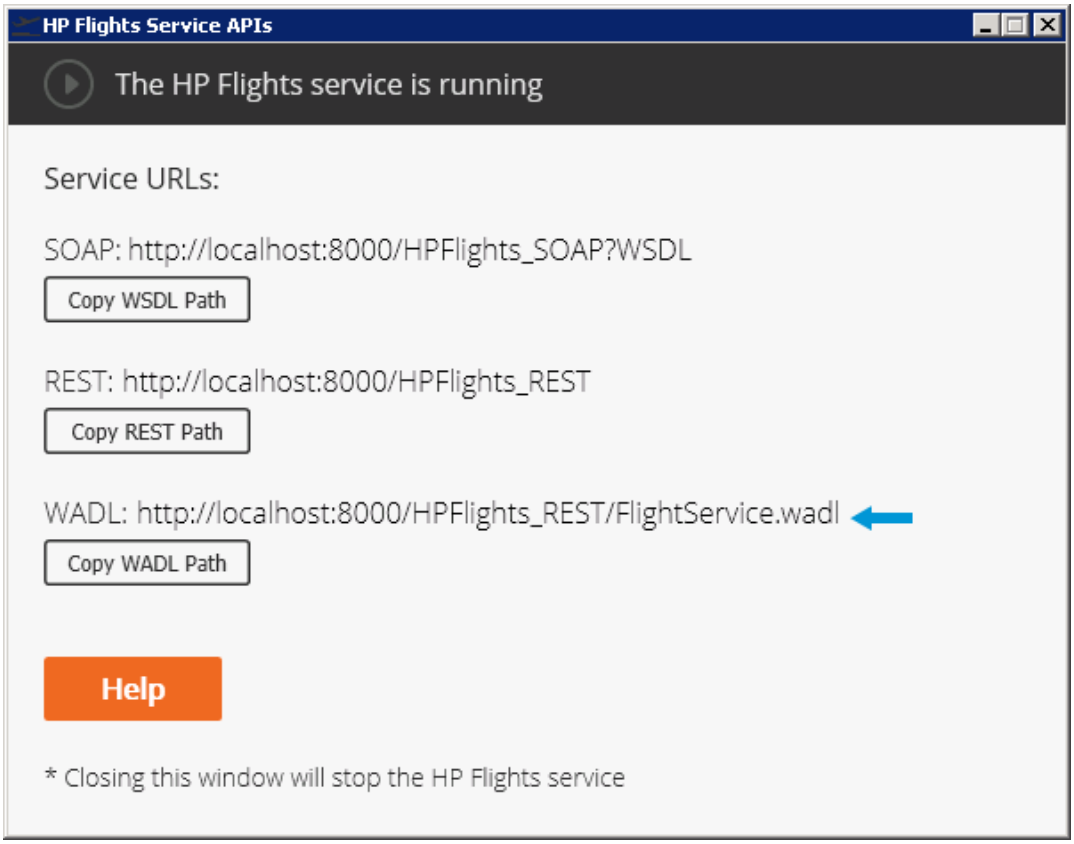
   > **Note:** Make sure that this application remains open when you are working with the tutorial, as UFT must be able to access it when editing and running the test.

2. **Import the WADL file.**

   a. In UFT, in the toolbar, click the **ADD REST Service** drop-down arrow and select **Import WADL from URL**. The Import WADL from URL dialog box opens.

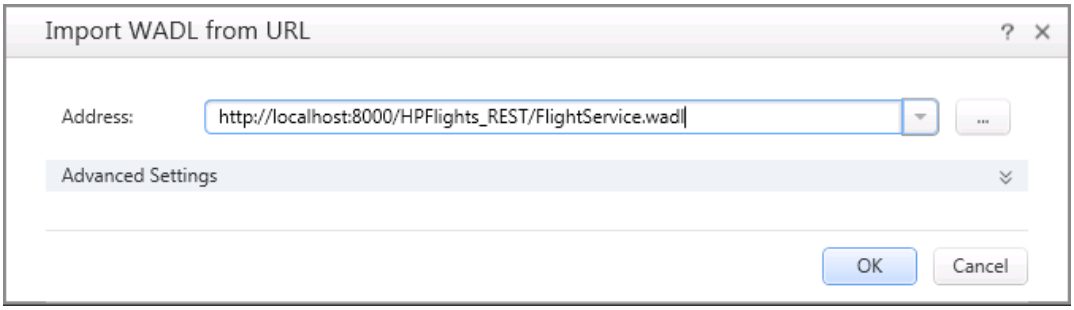
   > **Note:** If you have a copy of the WADL file saved locally, you can import the WADL file into UFT.

b. In the HP Flights Service API application window, locate the URL for the WADL service:



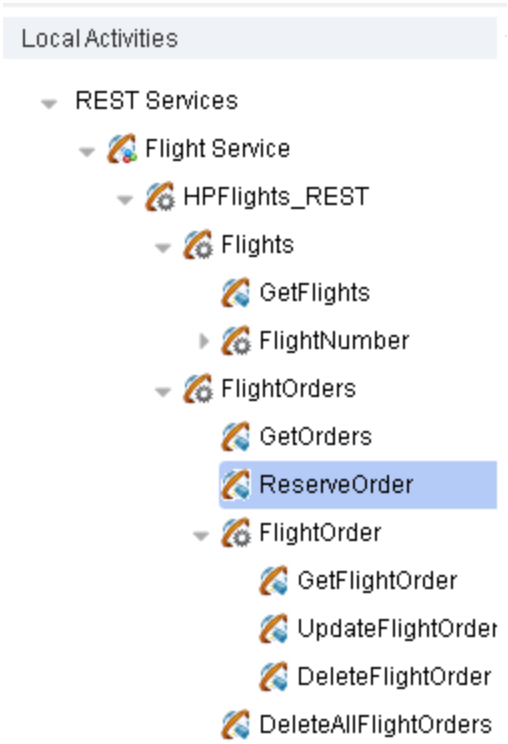
c. In the HP Flights Service API application window, click the **Copy WADL Path** button. This saves the URL of the WADL file so you can copy it into the Import WADL from URL dialog box.

d. In UFT, in the Import WADL from URL dialog box, in the **Address** field, paste the URL copied from the application window:



e. Click **OK** to import the service description into UFT.

The service description is imported into UFT and its resources and methods hierarchy is displayed in the Toolbox pane in the Local Activities section:



Now that you have imported the service description and its methods into UFT, you are ready to create a test using the methods. Continue to "Exercise 8c: Edit the Web Application service methods" below to learn how to edit the service description methods in UFT.

Exercise 8c: Edit the Web Application service methods

In "Exercise 8b: Import a Web Application service model" on page 221, you imported the Web application service description and methods into UFT to create a hierarchy of services, resources, and methods.

The WADL that you import contains a detailed description of the service's resources and methods, including the URL of the service and its methods, and parameters for various methods. Once you import the WADL file into UFT, you cannot change these properties.

However, you can add additional information to your service description, such as additional parameters for the methods, and request and response information for the methods. This additional information is also saved as part of the service model and is used as the prototype information when you drag a method to the canvas.

In this lesson, you will learn how to edit the service model's properties in UFT. You will be adding information to the GetFlights method and ReserveOrder methods, which you will use in the next exercise to create a test.

1. **Start UFT and open the flight reservation application solution.**
   a. If necessary, open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solution** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens in the Solution Explorer, containing the Flights WADL test you created in "Exercise 8a: Create a test for a Web Application Service" on page 221.

2. **Open the Web Application service model description.**

   When you edit a Web Application service description, you use the same dialog box and tools that you use to create and edit a REST service model description.

   In the Toolbox pane, under the **Local Activities** section, right click the Flight Service node and select Edit Service. The Edit REST Service dialog box opens.

3. **Edit the response information for the GetFlights method.**
   a. In the Edit REST Service dialog box, under the **Flights** node, select the **GetFlights** node. The right pane is updated with the information for the method.
   b. In the right pane, select the **HTTP** tab  .
   c. In a text editor, copy the following XML:

```
<ArrayOfFlight xmlns="HP.SOAQ.SampleApp"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
 <Flight>
   <Airlines>AA</Airlines>
   <ArrivalCity>Denver</ArrivalCity>
   <ArrivalTime>01:23 PM</ArrivalTime>
   <DepartureCity>London</DepartureCity>
   <DepartureTime>06:12 AM</DepartureTime>
   <FlightNumber>20279</FlightNumber>
   <Price>112.2</Price>
 </Flight>
</ArrayOfFlight>
```

Note: In this case, you must leave values as part of the XML, as this

> instructs UFT on the type of value that each parameter must use.

d. Save the file as **response.xml** in a directory of your choice.

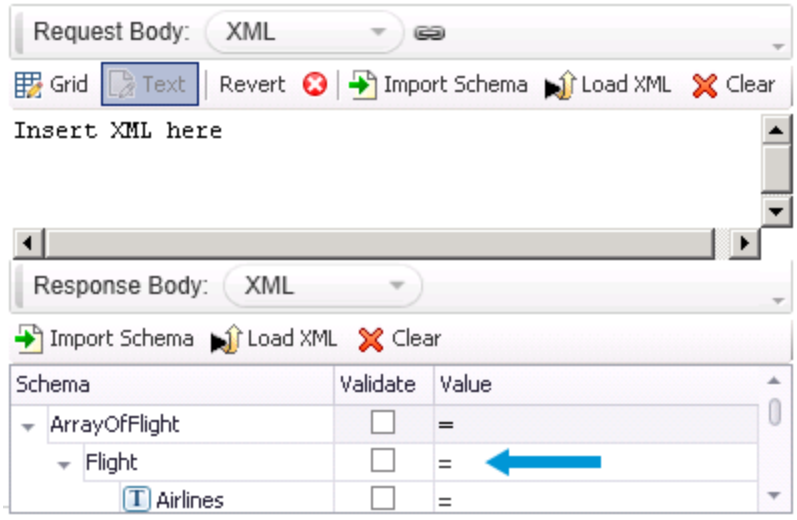
e. In UFT, in the Edit REST Service dialog box, with the GetFlights node selected, in the HTTP tab, in the **Response** section, from the drop-down list, select **XML**

f. In the Response Body section, click the **Load XML** button. The Open dialog box opens.

g. In the Open dialog box, navigate to the directory in which you saved the **response.xml** file and select the **response.xml** file.

h. Click **Open** to add the XML schema to the service description.

UFT adds the response properties based on the XML in the Response grid:



4. **Add request information for the ReserveOrders method.**

a. In the Edit REST Service dialog box, under the **FlightOrders** node, select the **ReserveOrder** node. The right pane is updated with the information for the ReserveOrder method.

b. In the right pane, select the **HTTP** tab .

c. In the HTTP tab, in the **Request** section, select **XML** from the **Request Body** drop-down list. The text editor opens with the message `Insert XML here`.

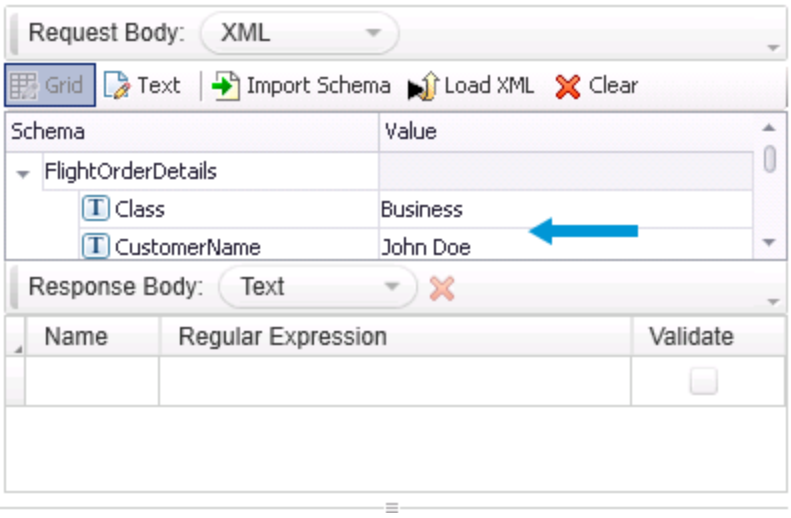
d. In the text entry area, paste the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<FlightOrderDetails xmlns="HP.SOAQ.SampleApp">
   <Class>Business</Class>
```

```
<CustomerName>John Doe</CustomerName>
<DepartureDate>2012-12-12</DepartureDate>
<FlightNumber>1304</FlightNumber>
<NumberOfTickets>21</NumberOfTickets>
</FlightOrderDetails>
```

> **Note:** In this case, you must leave values as part of the XML, as this instructs UFT on the type of value that each parameter must use.

If you click the **Grid** button, you can see the parameters for the XML request displayed in the Properties grid:



e. Click **OK** to save the changes.

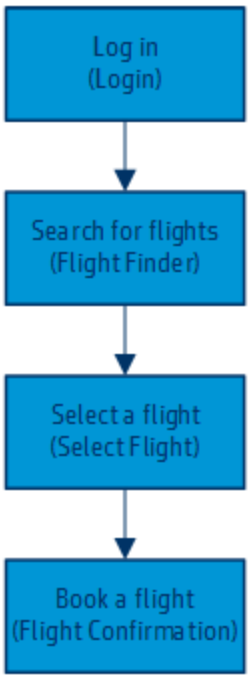
   UFT saves the changes to the service model and updates the properties of the methods already imported in the Toolbox pane.

Now that you have edited your Web Application methods to included additional information, you are ready to use them in a test. Continue to "Exercise 8d: Build a test with Web Application service methods" below to build a test using the methods imported from your WADL.

Exercise 8d: Build a test with Web Application service methods

In "Exercise 8b: Import a Web Application service model" on page 221, you imported a WADL file that contained descriptions of the methods used in your Web Application service. In "Exercise 8c: Edit the Web Application service methods" on page 224, you added additional property information to these methods which was not defined in the WADL file.

In the Book Flights GUI test that you created in "Lesson 3: Add steps to a test" on page 46, the order of the application windows was as follows:



When you create an API test of the same application, you want to make the steps match the application's flow as closely as possible. In the list of methods imported from the WADL file, you have the some of the following:

- **GetFlights**
- **GetFlightOrders**
- **UpdateFlightOrder**
- **DeleteFlightOrder**
- **DeleteAllFlightOrders**
- **ReserveOrder**

In order to match the flow of the user interface, you need to create API test steps that find the flight, and then create a flight order based on the customer input.

In this exercise, you will create two test steps: **GetFlights** and **ReserveOrder**.

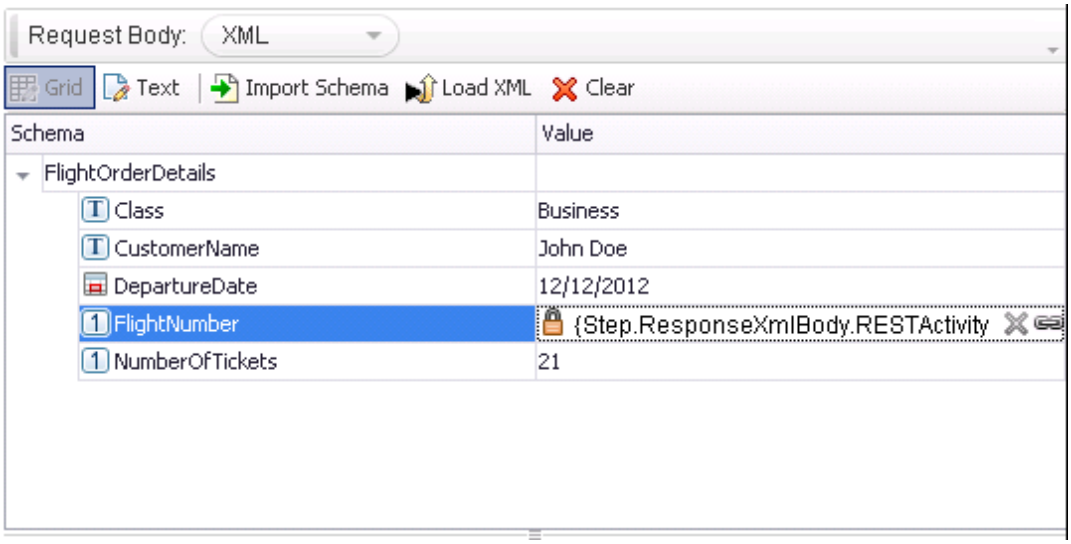
1. **Start UFT and open the flight reservation application solution.**
   a. If necessary, open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solution** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens in the Solution Explorer, containing the Flights WADL test you created in "Exercise 8a: Create a test for a Web Application Service" on page 221.

2. **Create the test steps.**
   a. In the Toolbox pane, in the **Local Activities** section, expand the **Flights** node (located under the **Flight Service** and **HPFlights_REST** nodes).
   b. Under the Flights node, drag a **GetFlights** method to the canvas. UFT adds a block in the Test Flow called **GetFlights**.
   c. Under the FlightOrders node, drag a **ReserveOrder** node to the canvas. UFT adds another block in the Test Flow (under the GetFlights block) called **ReserveOrder**.

3. **Link the FlightNumber property of the ReserveOrder step to the output of the GetFlights step.**
   a. In the canvas, select the **ReserveOrder** step. The **Input/Checkpoints** tab  opens in the Properties pane.
   b. In the Properties pane, select the **HTTP** tab  .
   c. In the **Request Body** section of the HTTP tab, in the **Value** cell of the **FlightNumber** property, click the **Link to data source** button . The Select Link Source dialog box opens.
   d. In the Select Link Source dialog box, select the **Available steps** option. The **Select a step:** pane (left pane) is updated with the list of available steps.
   e. In the Select a step: pane, select the **GetFlights** step. The **Select a property:** pane (right pane) is updated with the list of available properties.
   f. In the Select a property: pane, select the **HTTP** tab  . The list of HTTP properties is displayed.
   g. In the list of properties, select the **FlightNumber** property and click **OK**.

UFT updates the value of the FlightNumber property in the ReserveOrder step to reflect the link to the output of the GetFlights step:



4. **Add a data source to use in your test.**
   a. If necessary, select **View > Data** to display the Data pane.
   b. In the Data pane, click the **New Data source** button and select **Excel**. The New Excel Data Source dialog box opens.
   c. In the New Data Source dialog box, in the **Excel file path** field, click the **Browse** button.
   d. In the Open dialog box, navigate to the application Excel file, saved in **<UFT installation directory>\samples\Flights Application** and click **OK**.
   e. In the New Excel Data Source dialog box, name the file **WADL_Flights**.
   f. Select the **Link to the Excel file in its original location** option.
   g. Click **OK** to save the data source information and add the Excel data to your test.

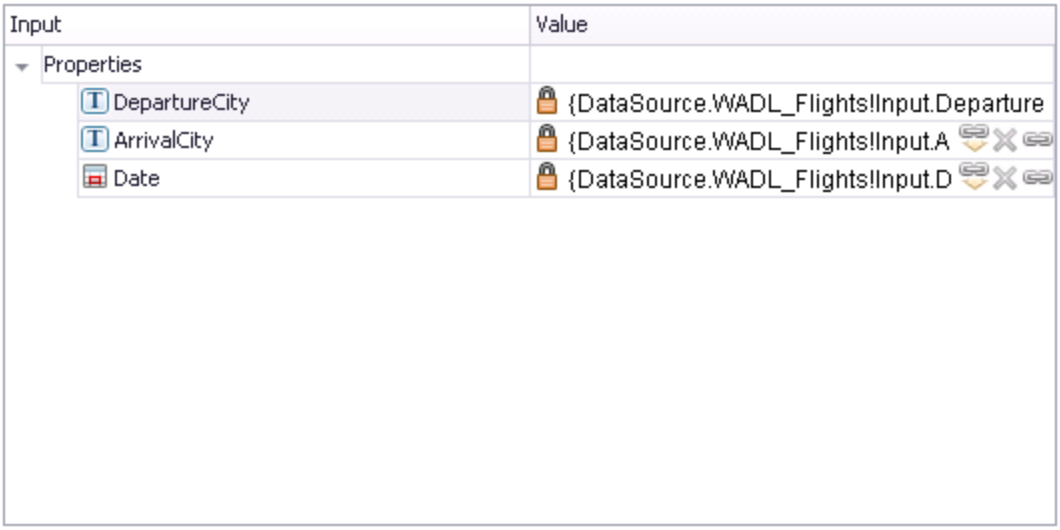
5. **Link the input properties of the GetFlights step to the data source.**
   a. In the canvas, select the **GetFlights** step. The **Input/Checkpoints** tab opens in the Properties pane.
   b. In the Input/Checkpoints tab, in the **Value** cell of the **DepartureCity** property, click the **Link to data source** button. The Select Link Source dialog box opens.
   c. In the Select Link Source dialog box, select the **Data source column** option. The **Select a data source:** pane (left pane) displays a list of all the available data sources.

d. In the Select a data source: pane, select the **WADL_Flights!Input** node. The **Select data:** pane (right pane) displays a list of all available data parameters (columns).

e. In the Select data: pane, select the **DepartureCity** parameter and click **OK**. UFT updates the Value column for the DepartureCity property in the Properties pane to reflect the link to the data source.

f. Repeat the same process to link the **ArrivalCity** and **Date** properties to the data source.

UFT displays the values of the GetFlights properties to display the link to the data source:



6. **Link the HTTP request properties for the ReserveOrder step to the data source.**

a. In the canvas, select the ReserveOrder step. The **Input/Checkpoints** tab opens in the Properties pane.

b. In the Properties pane, select the **HTTP** tab . The list of HTTP request and response properties is displayed.

c. In the **Value** cell for the **Class** property, click the **Link to a data source** button. . The Select Link Source dialog box opens.

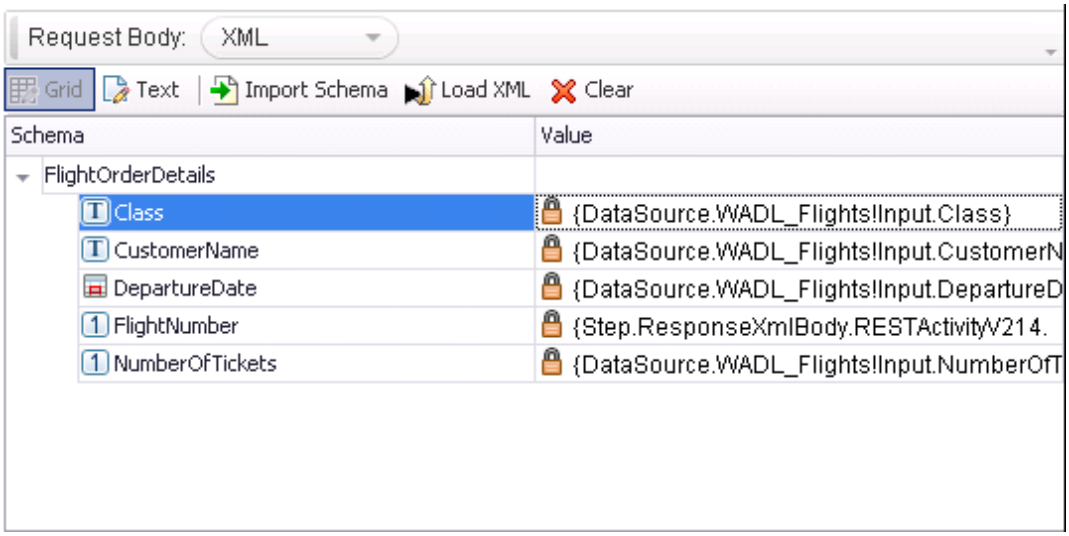
d. In the Select Link Source dialog box, select the **Data source column** option. The **Select a data source:** (left pane) displays a list of all available data sources.

e. In the Select a data source: pane, select the **WADL_Flights!Input** node. The **Select data:** pane (right pane) displays a list of all available data parameters (columns).

f. In the Select data: pane, select the Class column and click **OK**. UFT updates the value of the Class property to reflect the link to the data source.

g. Repeat the same process for the other HTTP Request properties:
   - **CustomerName**
   - **DepartureDate**
   - **NumberOfTickets**

   After you have linked all the properties (except the FlightNumber property) to the data source, the Properties pane updates the values to reflect the link:



7. **Save the test.**

   In the toolbar, click **Save**  .

Now that you have created a test using the methods imported from your WADL file, you are ready to run the test and view the run results. Continue to "Exercise 8e: Run a Web Application service test" below to run the test.

Exercise 8e: Run a Web Application service test

In "Exercise 8d: Build a test with Web Application service methods" on page 227, you created a test using the methods for your Web Application service model that you imported and edited in previous exercises.

In this exercise, you will run the test you created to see the results.

1. **Start UFT and open the flight reservation application solution.**
   a. If necessary, open UFT as described in "Create a solution" on page 26.
   b. On the Start Page, in the **Recent Solution** area, click **Flight Reservation Application**.

The Flight Reservation Application solution opens in the Solution Explorer, containing the Flights WADL test you created in "Exercise 8a: Create a test for a Web Application Service" on page 221.

2. **Set the number of iterations for the test.**

   a. In the canvas, select the Test Flow. The **Input** tab opens in the Properties pane.

   b. In the Input pane, select the **'For' Loop** option.

   c. In the **Number of iterations** field, enter **8**.

3. **Set the data navigation properties for the data source.**

   a. In the canvas, select the **Test Flow**. The **Input/Checkpoints** tab opens in the Properties pane.

   b. In the Properties pane, select the **Data Sources** tab. A list of all data sources associated with the test flow is displayed

   c. In the list of data sources, select the **WADL_Flights!Input** data source and click **Edit**. The Data Navigation dialog box opens.

   d. In the Data Navigation dialog box, configure the following data navigation properties:

Start at:	First row
Move by:	1 rows Forward
End at:	Last row
Upon reaching the last row	Wrap around

4. **Run the test.**

   a. Ensure that the HP Flights Service APIs application is open.

   b. In the toolbar, click the **Run** button.

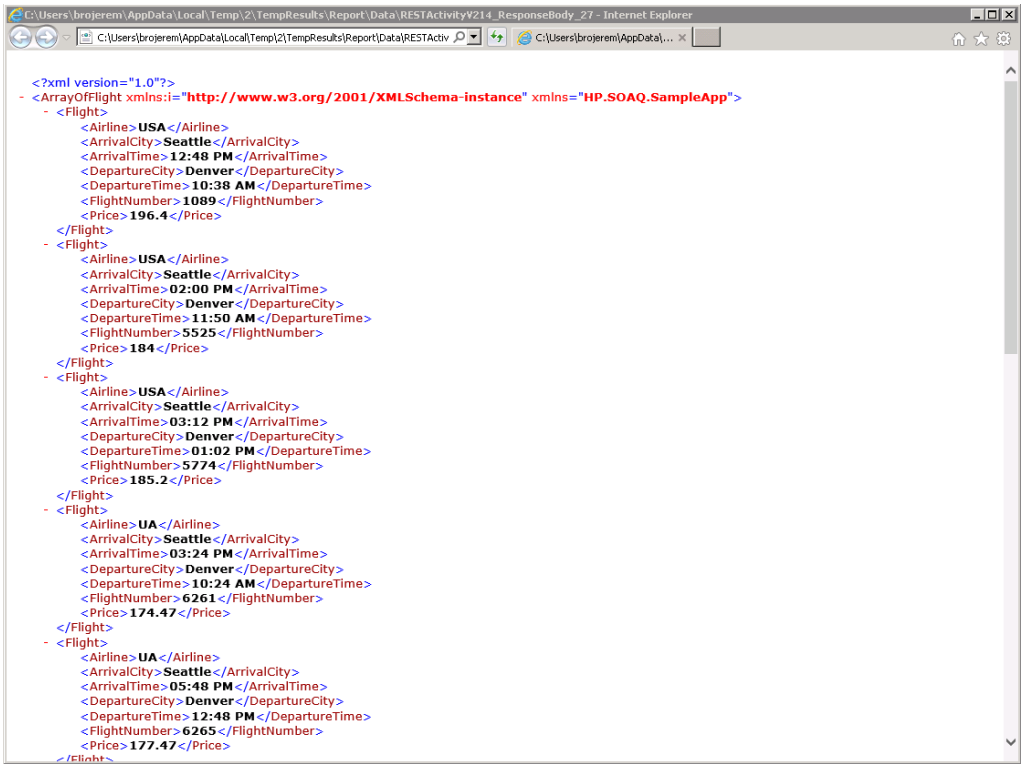
   UFT runs the test steps, providing the property values from the data source. The test run log is displayed in the Output pane.

   After the test run is over, the run results open.

5. **Analyze the run results.**

   a. In the Test Flow, find the **ReserveOrder** step. The run results display a summary of the step.

b. In the captured data, note the Request and Response information. You can click on the links in the **Request Body** and **Response Body** cells to open the XML response and request information in a browser window:



Part 5: Creating and Running GUI and API Tests in a Single Test

> **Note:** You must perform the tutorial lessons included in "Create and running automated GUI tests" on page 28 and "Create and run automated API tests" on page 150 before doing this part of the tutorial.

When you are testing your application, in order to perform a comprehensive test, you must test both the user interface (the GUI) and the service layer (the API). One of the challenges in doing this is maintaining and running separate tests for each part of your application.

However, in UFT, although you still must create and maintain both GUI and API tests of your application, you can run unified tests which test both the GUI and the API of your application in a single unified test run. You simply call an API test from a GUI test, and UFT runs both layers of the application within a single test run. Then, after the test run is complete, the run results display a unified view, reporting the performance of both the GUI and API layers in a single report.

In this part, you will learn how to create and run tests which include both GUI and API tests in a single test run.

This section includes the following:

Lesson 1: Create a test to run GUI and API tests together

In this lesson, you will create a separate test in order to run a unified test with GUI and API tests together.

1. **Start UFT and open the Book Flights test.**
   a. Open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the **Book Flights** GUI tests you created in "Create and running automated GUI tests" on page 28 and the API tests you created in "Create and run automated API tests" on page 150.

2. **Save the Book Flights test as Flight Reservation Application.**
   a. In the Solution Explorer, right-click the **Book Flights** node and select **Save As**. The Save As dialog box opens.
   b. In the Save Test As dialog box, browse to **C:\%HOMEPATH%\My Documents\Unified Functional Testing** and save the test as `Flight Reservation Application`.

      In the Solution Explorer, the Book Flights test is replaced by the new Flight Reservation Application test. The Book Flights test is still saved separately in the file system.

3. **Add the Book Flights test back into the solution.**
   a. Select **File > Add > Existing Test**. The Add Test to Solution dialog box opens.
   b. In the Add Test to Solution dialog box, browse to the **C:\%HOMEPATH%\My Documents\Unified Functional Testing** directory, and select the **Book Flights** test.
   c. Click **Add** to return the Book Flights test to the solution.

      The Book Flights test is again displayed in the Solution Explorer.

Now that you have a separate test for running GUI and API tests together, you are ready to build the test to include both types of tests. Continue to "Lesson 2: Call the API test from a GUI test" on the next page to learn how to build the test to include both types of tests.

Lesson 2: Call the API test from a GUI test

In order to run an API test from a GUI test, you must first call the API test. In this lesson, you will learn how to add calls to an API test from a GUI test in order to run both tests in a single, unified test run.

1. **Start UFT and open the Book Flights test.**
   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the Flight Reservation Application test you created in "Lesson 1: Create a test to run GUI and API tests together" on the previous page.

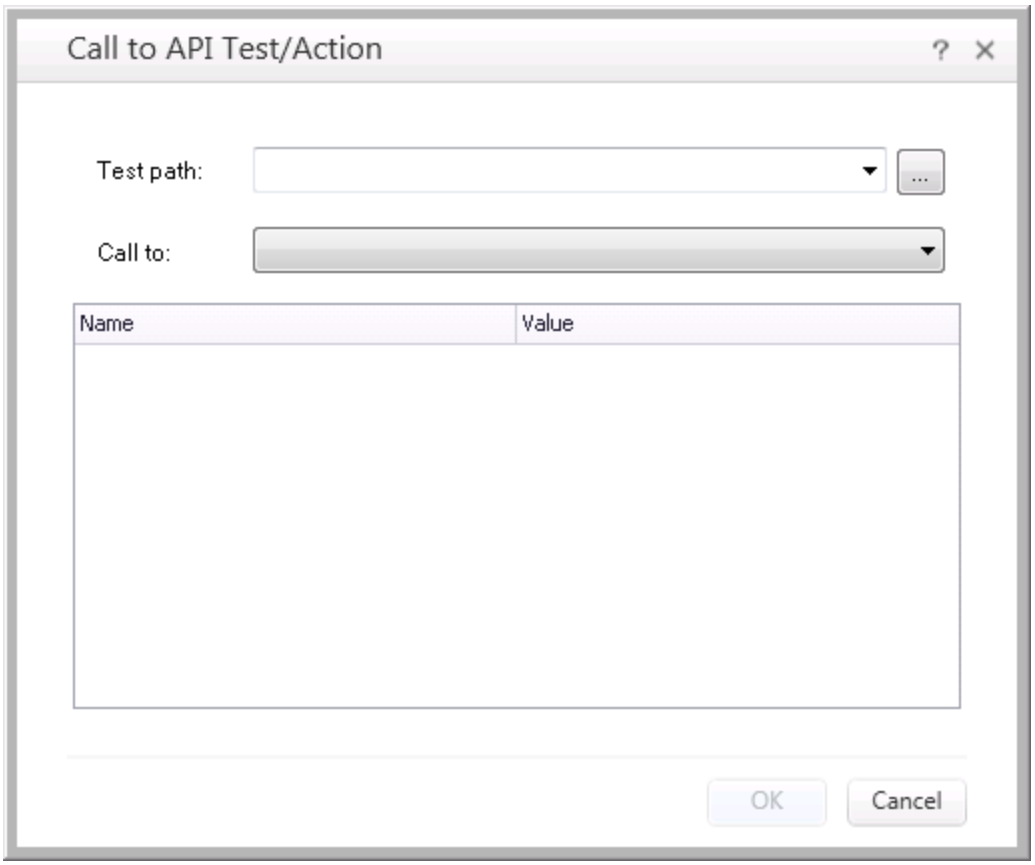
2. **Create a new action for the API test call.**

   When you call an API test, UFT runs the called API test in its entirety. You can put the API test call as a separate step in an action, or as its own independent action.

   a. In the toolbar, click the **Call to New Action** drop -down arrow  and select **Call to New Action**. The Insert Call to New Action dialog box opens.
   b. In the Insert Call to New Action dialog box, name the action **API Test Call**.
   c. Ensure that the **At the end of the test** option is selected. Keep the other options as default.
   d. Click **OK** to add the action to the test.
   e. In the canvas, right-click the **API Test Call** action and select **Move Up**. The API Test Call action is moved above the **Flight Confirmation** action in the Test Flow.

3. **Add a call to the API test.**
   a. In the canvas, double-click the **API Test Call** action block. The action opens as a separate tab in the document pane.

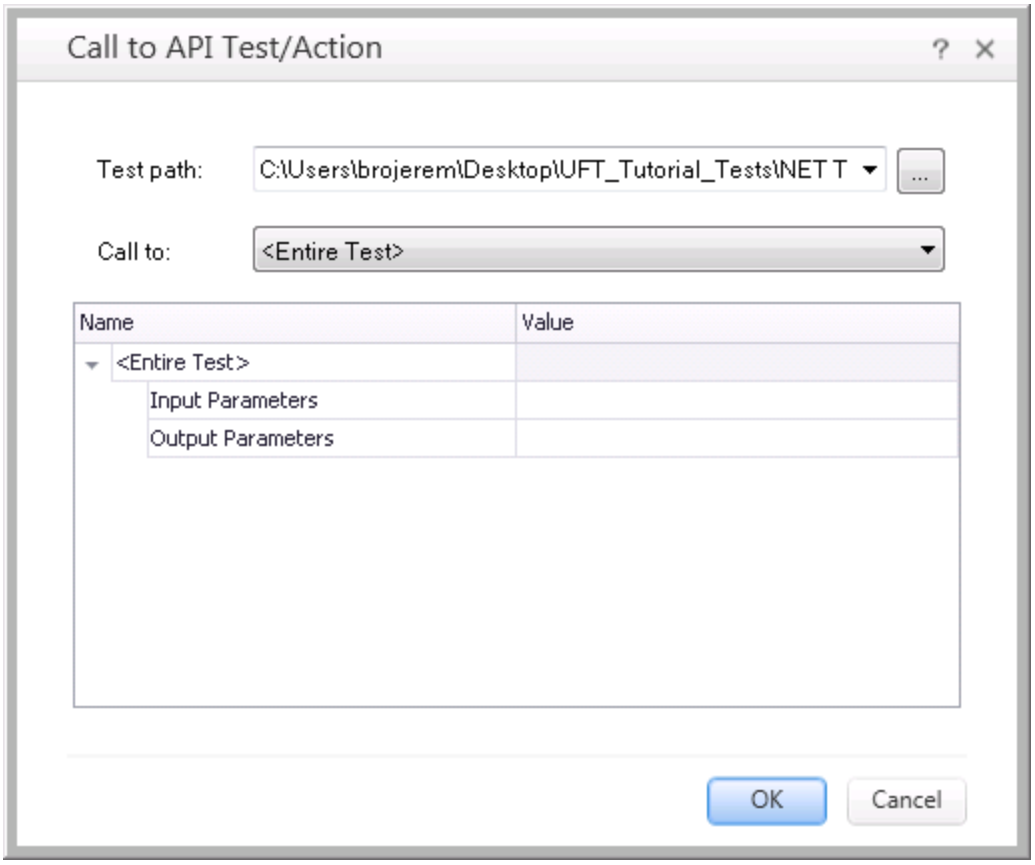
b. In the toolbar, click the **Call to New Action** drop-down arrow and select **Call to Existing API Test/Action**. The Call to API Test/Action dialog box opens:



c. In the Call to API Test/Action dialog box, in the **Test Path** field, click the **Browse** button. The Open Test dialog box opens.

d. In the Open Test dialog box, navigate to the folder with the **Flight Reservation Application** solution tests, stored at **C:\%HOMEPATH%\Unified Functional Testing**.

e. In this directory, select the **Book Flights Web Service** test and click **Open**. UFT adds the name of the test and its parameters to the Call to API Test/Action dialog box:



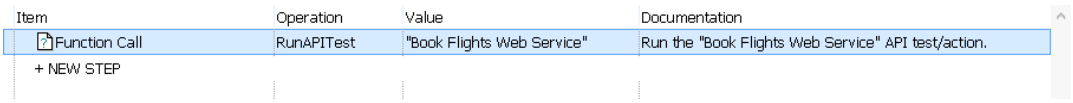
> **Note:** If your test had output parameters, you can use this dialog box to specify the place in which to store the API test output parameter. In this case, the API test has no output parameters, so you will not perform this step.

f. Click **OK** to add the call as a step in your test.

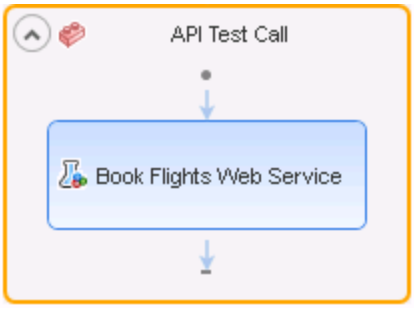
UFT adds a step for the API test call in the API Test Call action. In the editor, the step is displayed as follows:

```
RunAPITest "Book Flights Web Service"
```

In the Keyword View, the step looks like this:



UFT also adds a visual indicator of the action call to the API Test Call action in the canvas:



4. **Save the test.**

   Select **File > Save**.

Now that you have created the test step to call the API test, you are ready to run the test. Continue to "Lesson 3: Run a GUI test that calls an API test" on the next page to run the test and view the run results.

Lesson 3: Run a GUI test that calls an API test

In "Lesson 2: Call the API test from a GUI test" on page 237, you added a call to an API test to an existing GUI test, which enables you to run unified tests of the flight reservation application in one test run.

In this lesson, you will run the test and see the results.

1. **Start UFT and open the Book Flights test.**
   a. If necessary, open UFT as described in "Create a solution" on page 26. Make sure that the WPF Add-in is loaded.
   b. On the Start Page, in the **Recent Solutions** area, click **Flight Reservation Application**.

      The Flight Reservation Application solution opens, containing the Flight Reservation Application test you created in "Lesson 1: Create a test to run GUI and API tests together" on page 236.

2. **Update the number of iterations for the Book Flights Web Service test.**

   Remember that when you created the Book Flights Web Service test, you ran the test with multiple iterations. However, in the Flight Reservation Application test (which is calling the Book Flights Web Service test), there is only one iteration of the test. Therefore, you should modify the API test to run the same number of iterations.
   a. In the Solution Explorer, expand the nodes under the **Book Flights Web Service** node.
   b. Under the Book Flights Web Service node, double-click the **Flow** node. The Book Flights Web Service Test Flow opens as a separate tab in the document pane.
   c. In the canvas, select the **Test Flow**. The **Input** tab  opens in the Properties pane.
   d. In the Input tab, in the **Number of Iterations** field, change the number to **1**.
   e. Select **File > Save** to save the modified settings.

3. **Set the run settings for the Flight Reservation Application test.**
   a. In the document pane, select the **Flight Reservation Application** tab.

      If the Flight Reservation Application tab is closed, double-click on the **Flight Reservation Application** node in the Solution Explorer to open it.
   b. Select **Record > Record and Run Settings**. The Record and Run Settings dialog box opens.

c. In the **Windows Applications** tab, ensure that the **Record and run only on:** and **Applications specified below** options are selected. The application details should be saved from when you ran the original Book Flights test in "Lesson 4: Run and analyze GUI tests" on page 73.

d. Click **OK** to close the dialog box.

4. **Run the Flight Reservation Application test.**

a. Before running the test, ensure that the **HP MyFlight Sample Application** (the user interface for the flight reservation application) window is closed.

b. Ensure that the **HP Flights Service APIs** window is open.

c. In the toolbar, click the Run button . The Run dialog box opens.

d. In the Run dialog box, in the **Results location** tab, select the **New run results folder** option. Keep the default folder name.

e. Click **Run** to start the test run.

UFT opens the HP MyFlight Sample Application window and performs the steps on the application objects as created in the GUI test. When the test comes to the API test call, the GUI test pauses and the API test comes into focus.

While the API test runs, you can view the progress of the API test run in the Output pane.

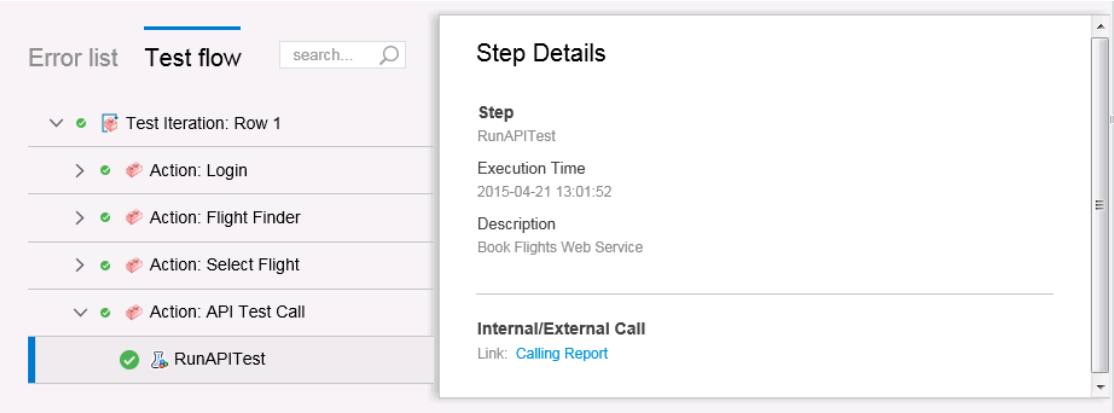
Aft er the API test run is finished, the remainder of the GUI test runs.

When the complete test run is complete, the run results open and displays the test results.

5. **View the run results.**

In the run results, display the **Test Flow**.

Note that there are separate nodes for each of the actions in the GUI test. However, you can also see the API test run results as part of the run results:



If you select a node in the API test results, you can view the step details in step summary

6. **Close the run results.**

   After viewing the run results, close the tab containing the run results.

Where do you go from here?

Now that you have learned how to use UFT, including creating automated GUI tests, automated API tests, and tests running both GUI and API tests, you are ready to use UFT to test your own application.

We suggest that you use the following procedure when testing your own application.

1. **Analyze your application.**
   - Determine the development environment. This enables you to load the relevant UFT add-ins and provide support for the objects in your application.
   - Determine the business processes that users will perform. Plan your tests and actions accordingly.
   - Decide how to organize your test and which operations to include. Consider the goals of the test, and confirm that your application and UFT are set to match the needs of your test.

   At this stage, you can begin creating the skeletal tests and actions to use when testing your application.

2. **Prepare your testing infrastructure.**

   Decide how to store the objects in your test. You can store the objects for each action in its corresponding local object repository, or you can store the objects for each action in one or more common (shared) object repositories. You can also use the same shared object repository for multiple actions.

   - **If you are new to testing**, you may want to use a local object repository for each action. This is the default setting, and all objects are automatically added to the local repository of each action.
   - **If you are familiar with testing**, it is often most efficient to work with shared object repositories, which can be used for one or more actions. Object information is kept in one central location, and when the objects in your application change, you can update them in that one location for multiple actions, in multiple tests.

   Although not discussed in this tutorial, you can also export test objects from a local object repository to a shared object repository, and you can merge object repositories.

   You may also want to create function libraries to enhance UFT functionality.

   For details, see the *HP Unified Functional Testing User Guide*.

3. **Build your test.**

   While you create your test steps, follow the steps you expect users to perform as they navigate within your application.

4. **Enhance your test.**

   - Add checkpoints to search for specific values of a page, object, text string, or table cell.
   - Replace fixed values in your test with parameter to check how your application performs the same operations with multiple sets of data.

   You can further enhance your test with programming, conditional, and loop statements, which add logic to your test. For details, see the *HP Unified Functional Testing User Guide*.

5. **Debug your test.**

   Debug your test to check that it operates smoothly and without interruption. For details, see the *HP Unified Functional Testing User Guide*.

6. **Run your test.**

   Run your test on your application to check that the application functions as expected.

7. **Analyze the run results.**

   Examine the results of your test to pinpoint defects in your application. (Refer to the appropriate sections of this tutorial to understand what to look for in the run results for checkpoints or parameters.)

Accessing UFT in Windows 8.X or Higher Operating Systems

Note: By default, the Start and Apps screens on Windows 8.x or higher are set to open Internet Explorer in Metro Mode. However, if User Account Control is turned off on your computer, Windows 8 will not open Internet Explorer in Metro mode. Therefore, if you try to open an HTML shortcut from the Start or Apps screen, such as the UFT Help or Readme file, an error will be displayed.

To solve this, you can change the default behavior of Internet Explorer so that it never opens in Metro mode. In the **Internet Properties** dialog box > **Programs** tab, select **Always in Internet Explorer on the desktop** for the **Choose how you open links** option. For more details, see http://support.microsoft.com/kb/2736601 and http://blogs.msdn.com/b/ie/archive/2012/03/26/launch-options-for-internet-explorer-10-on-windows-8.aspx.

Send Us Feedback

Let us know how we can improve your experience with the Tutorial.

Send your email to: docteam@hpe.com


Java™
COMPATIBLE