HP Unified Functional Testing

Software Version: 11.50

API Testing Tutorial

Document Release Date: December 2012

Software Release Date: December 2012

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:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support web site at:

http://www.hp.com/go/hpsoftwaresupport

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

HP 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 HP 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 HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Table of Contents

Welcome to the Tutorial

Welcome to the API Testing Tutorial for non-GUI testing, a self-paced printable guide, designed to lead you through the process of creating tests for Web services, REST services, and other GUI-less applications.

At the conclusion of this tutorial, you will be ready to design, run, and monitor a simple test on your own application. It is recommended that you proceed through the tutorial in the order in which the information is presented.

Tutorial Lessons

The following lessons are included in this tutorial:

Lesson 1: Introducing HP Unified Functional Testing - API Testing: Familiarizes you with the HP Unified Functional Testing Tutorial for API Testing.

Lesson 2: Build a Simple Test: Introduces you to the basic creation of a test through dragging and dropping activities into the canvas. It also shows you how to data-drive the test with external data.

Lesson 3: Test a Web Service: Shows you how to create a test for a Web service and how to check the results.

Lesson 4: Test REST Services: Shows you how to model and create tests for REST services.

1

Introducing HP Unified Functional Testing - API Testing

HP Unified Functional Testing for API testing contains an extensible framework for the construction and execution of functional tests of headless systems, systems that do not have a user interface. This document describes how to get started with HP Unified Functional Testing and create your first tests. It also introduces the major features in the product and how to incorporate them into your tests.

You create a test by dragging activities onto a canvas. You set parameters, assign values, and run your test. You can check for expected values with the built-in checkpoint mechanism or through custom C# code.

What is SOA?

In recent years, outsourcing and advanced business needs required companies to collaborate and share information. In addition, with the rising number of mergers and acquisitions, businesses struggled to find a way to share data. If, prior to a merger, two businesses maintained their own proprietary computer systems, after the merger, the sharing of data could be very time consuming and costly.

For these issues, several vendors developed technologies to handle B2B (Business-to-Business) communication. Examples of B2B technologies are RMI, COM, CORBA, EDI, and Web services. In addition to allowing the systems to link up to one other, the technologies also handle permissions to allow networking with each other like an Intranet.

Web services are self-contained applications that can run across the Internet on a variety of platforms. They use XML and Simple Object Access Protocol (SOAP) as the base language, making it a developer-friendly solution. Since Web services are based on a set of standardized rules and specifications, they are more portable than other technologies.

SOA (Service Oriented Architecture) is an architectural style that lets multiple software services interact. A service is a unit of work done by a service provider as specified by a consumer. SOA requires that services interact with one another, but without interdependencies. The services are autonomic and loosely coupled, only requiring that they retain an awareness of one other, but no dependencies.

SOA systems are primarily based on Web services. In Web services, a client submits a request and the Web server provides a response using the SOAP protocol. HP Unified Functional Testing for API testing lets you check the behavior of your Web services in an automated manner.

Why should you automate SOA testing?

Automated SOA Testing is a discipline that leverages products and processes to reduce the risks of application upgrades or deployment of new services. At its core, automated testing is about applying production workloads to pre-deployment systems while simultaneously measuring system performance and end-user experience. A well-constructed performance test answers questions such as:

- Does the service respond quickly enough for the intended users?
- Will the server respond with the correct values?
- How will the service handle exceptions and illegal values?
- Is the service stable under expected and unexpected user loads?

By answering these questions, you can design a test more effectively. An effective automated testing process helps you make more informed release decisions, reduces system downtime, and prevents availability problems.

Understanding Terminology

You may encounter the following terms when working with SOA testing:

HTTP	**Hypertext Transfer Protocol**, a communications protocol used to transfer or provide information over the World Wide Web. Users utilize HTTP to publish and retrieve HTML pages.
JMS	**Java Message Service**, a Java-based Message Oriented Middleware API for sending messages between two or more clients.
REST	**Representational State Transfer.** A style of software architecture for distributed systems.
SOAP	**Simple Object Access Protocol**, or **Service Oriented Architecture Protocol**, a protocol for exchanging structured and typed information between peers in a distributed environment using XML sent over HTTP or JMS. SOAP lets you serialize distributed components into XML documents for transport and deserialize them upon reaching their destination. This promotes interoperability between components that are based on different technologies.
Test	The steps that a user performs are described in a **test**. A test emulates the actions of real users using the application.
UDDI	**Universal Description, Discovery and Integration**, a platform-independent, online registry listing businesses worldwide. It is often used as a database for public Web services.
Web services	Standardized, Web-based applications using the XML, SOAP, WSDL and UDDI standards over an Internet protocol. XML is used to tag the data, SOAP is used to transfer the data, WSDLs describe the services and UDDIs list them.

WSDL	**Web services Description Language**, an XML-based language designed to describe a Web service. The WSDL document provides essential information about the Web service, such as its ports and operations, required for implementation.
WS-I	**Web Service Interoperability**, a standard created by the Web Service Interoperability organization to promote compatibility between Web Services.
XML	**Extensible Markup Language** is a general-purpose markup language. It is called extensible because it lets you define your own tags. It enables the sharing of structured data across different information systems, especially over the Internet. XML is used both to serialize data and encode documents.
XSD	**XML Schema Definition** files contain a set of rules that formally define the hierarchical structure of an XML document. An XML document must comply with these rules and constraints in order for parsers and processors to deem it valid. WSDL documents can reference an external schema or it can contain an embedded one.

What is the sample application for this tutorial?

This tutorial is based on a sample application included with this product—the **API Flights**. It is available as both a Web and REST service.

The sample application works with a flight reservation database. You can retrieve flights for specific destinations, create customer orders, update reservations, or delete them.

For details about the service's methods and operations, type help in the Sample Application's command prompt window.

Note: It is recommended to have Microsoft Excel installed on your machine to enable Excel functionality for the data sources.

How do I invoke the application?

As a first step, you will invoke the sample flight application, so that it will be available for your test.

To start the HP Flights service:

1. Make sure you have administrator privileges. These are required by Windows to run the sample HP Flights service.
2. Select **Start** > **(All) Programs** > **HP Software** > **HP Unified Functional Testing** > **Sample Applications** > **Flight API**. A Command window opens indicating that the application is available.
3. If the window issues a message that the default port 24240 is unavailable, edit the **<installation_directory>SampleApplication\HPFlights_Service.exe.config** file in a text editor. In the **appSettings** section, replace the 24240 port key with a valid one.
4. Minimize the sample application's Command window. Do not close the Command window, as this will stop the service.

Where To Go From Here

Now that you have invoked the application, you can begin creating tests for your headless applications using the API Flights application. The following lessons will walk you through the process of creating a test for basic activities, Web, and REST services.

2

Build a Simple Test

This lesson will guide you through the steps of creating tests with simple actions.

This lesson contains the following sections:

- ➤ How do I create a new test?
- ➤ How do I work with the Unified Functional Testing panes?
- ➤ How do I create a test step?
- ➤ How do I data drive the step?
- ➤ How do I connect test steps?
- ➤ How do I map data from multiple sources?

How do I create a new test?

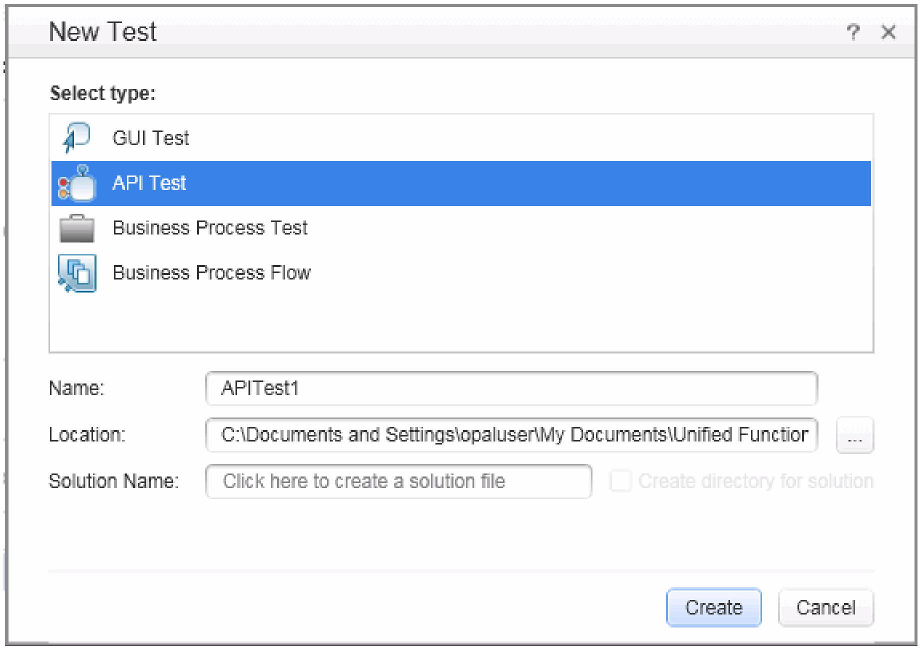
The canvas is the central console in which you build and run your test.

1 Open HP UFT.

Choose **Start** > **(All) Programs** > **HP Software** > **HP Unified Functional Testing** > **Unified Functional Testing**. The Start Page opens.

2 Create a new test.

Click **File** > **New** > **Test**. The New Test dialog box opens. Select the **API Test** type. On machines with an installation of HP LoadRunner, the **API Load Test** type is also visible.



Name: APITest1

Location: C:\Documents and Settings\opaluser\My Documents\Unified Functior

Solution Name: Click here to create a solution file

3 Generate the new solution.

In the **Name** box, replace the default name with **BasicTest**, and click **Create**. An empty test opens, with a canvas showing the **Start**, **Test Flow**, and **End** sections.

The **Test Flow** is the section of the test containing the activities whose functionality you want to test. The **Start** section is ideal for defining items that you want to initialize before the test, such as test variables.

How do I work with the Unified Functional Testing panes?

Most of the panes in the Unified Functional Testing interface, are floating, dockable windows. To show the default panes in their original positions, select **View** > **Reset Window Layout**.

The primary panes are:

- **Solution Explorer pane.** (left) A tree hierarchy of all tests and actions in the current solution, with their references, flow, and events.
- **Toolbox pane.** (left) A collection of built-in and imported activities that can be added as test steps. From this pane, you drag activities into the canvas.
- **Canvas**. (middle) The work area in which you organize the test steps.
- **Data Pane.** (bottom) A tree hierarchy of data sources that can be used with the test—imported Excel and XML files or database tables, or a manually defined table.
- **Output Pane.** (bottom) An informational area providing information about the test run and status.

How do I create a test step?

You create test steps by dragging activities from the **Toolbox** pane into the canvas.

In this section you will create a simple test step to illustrate the use of the Toolbox and the various panes.

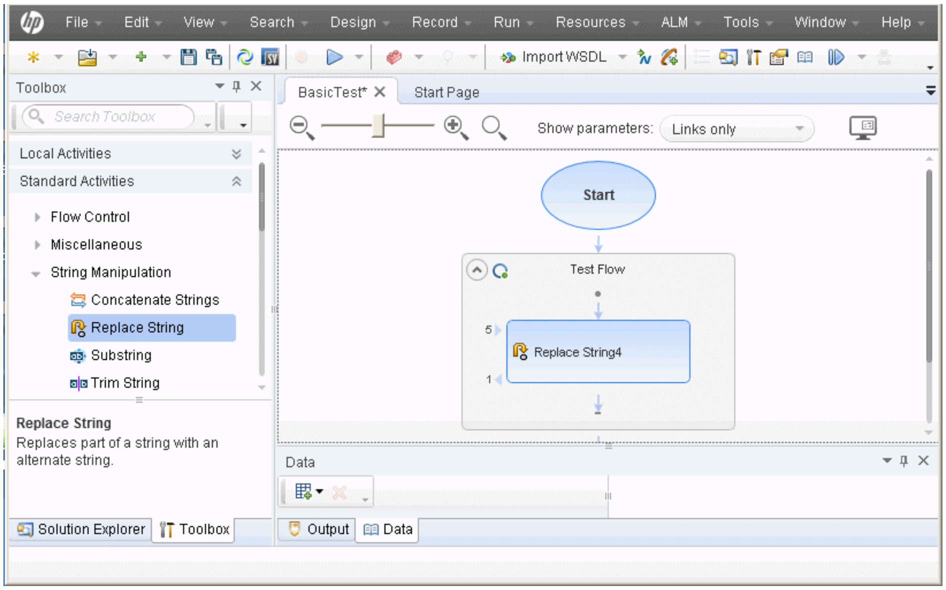
Create a sample Replace String test step:

1 Locate the Replace String activity.

In the left pane, click the **Toolbox** tab to show the **Toolbox** pane. Expand the **String Manipulation** category and select **Replace String**.

2 Create a step.

Drag the **Replace String** activity onto the canvas and drop it in the **Test Flow**. This activity searches for text within a specific string, and replaces it with new text. Alternatively, double-click on the activity in the Toolbox to add it to the canvas.



Replace String
Replaces part of a string with an alternate string.

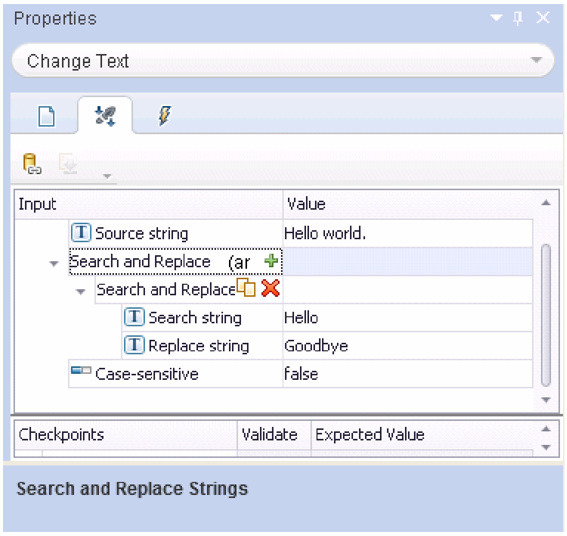
3 Change the step's display name in the General view.

Select **View** > **Properties**. Select the **Replace String** step in the canvas, and in the **Properties** pane, click the **General** tab. In the **Name** row, type **Change Text** and press ENTER. This changes the step name in the canvas.

4 Set the input properties.

In the **Properties** pane, select the **Input/Checkpoints** tab. Enter the following values:

- ➤ **Source string:** Hello world.
- ➤ **Search string:** Hello
- ➤ **Replacement string:** Goodbye
- ➤ **Case-sensitive:** false



Input	Value
Source string	Hello world.
Search and Replace (ar	
Search and Replace	
Search string	Hello
Replace string	Goodbye
Case-sensitive	false

Checkpoints	Validate	Expected Value

Search and Replace Strings

5 Run the test.

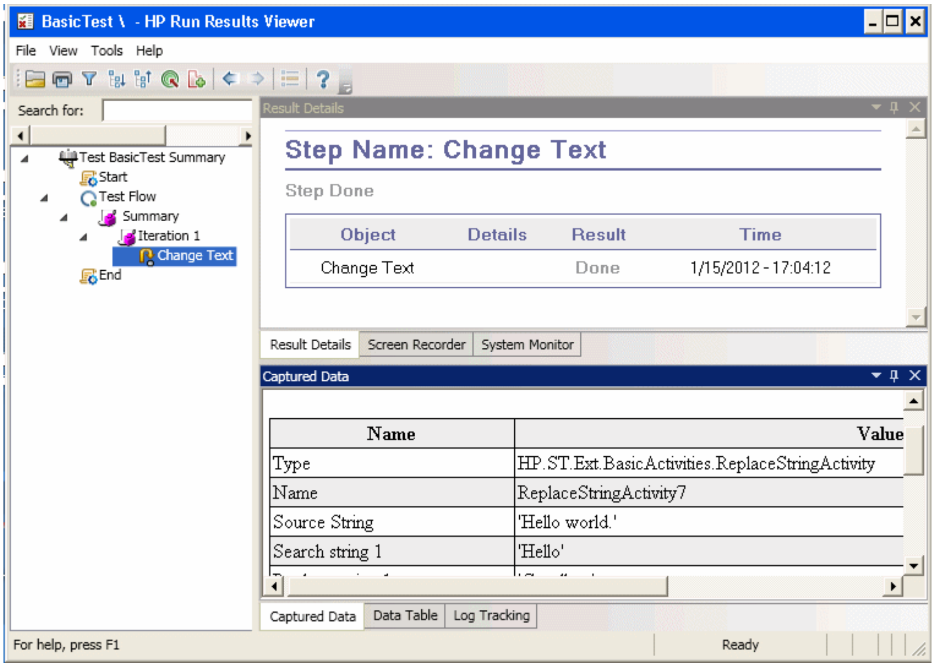
Click the **Run** button or press F5 to open the Run Test dialog box. Click **Options** to expand the dialog box. Select the **Temporary run results folder** option. Click **Run** to compile and run the test.

6 View the results.

The Run Results Viewer opens.

Select **View** > **Expand All** or click the Expand All toolbar button. Click the **Change Text** node. View the source and replacement strings and note the result string, **Goodbye world**. This is in fact the expected string—the test passed.

When you are finished reviewing the results, close the Run Results Viewer.



Step Name: Change Text

Step Done

Object	Details	Result	Time
Change Text		Done	1/15/2012 - 17:04:12

Name	Value
Type	HP.ST.Ext.BasicActivities.ReplaceStringActivity
Name	ReplaceStringActivity7
Source String	'Hello world.'
Search string 1	'Hello'

7 Set a checkpoint.

In the previous step, you manually viewed the output to check if the result matched the expected value. Checkpoints allow you to see whether the action was successful without having to manually check the result. Checkpoints are the means to validate the test—a success or failure is determined by its checkpoints.

Return to the Properties pane (right pane) and ensure that the **Input/Checkpoints** tab is displayed. Click in the lower part of the pane, the **Checkpoints** section, and select the check box in the **Results** row to enable the checkpoint. In the **Expected value** column, type the expected string, **Goodbye world**.

Run the test again. In the Run Results Viewer, expand the nodes, and note the checkmark. This indicates that the checkpoint passed since the result matched the expected value.

When you are finished reviewing the results, close the Run Results Viewer.

How do I connect test steps?

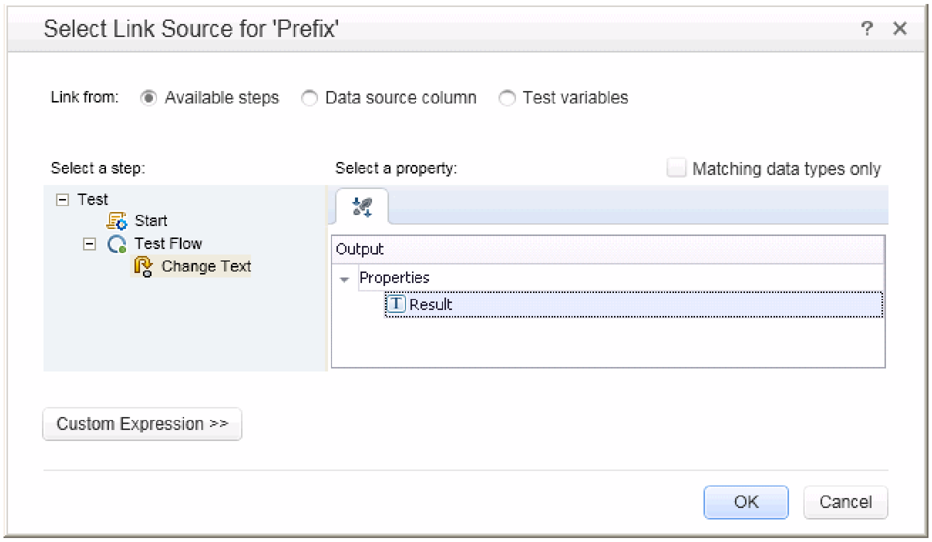
In this section, you will use the output of one step, as input for another.

1 Add a Concatenate String step.

In the **Toolbox** pane, select **Concatenate String** from the **String Manipulation** category. Drag the activity into the canvas and drop it below the Change Text step in the Test Flow. This activity concatenates two strings.

2 Set the prefix.

In the canvas, select the **Concatenate String(x)** step. In the **Properties** pane, click the **Input/Checkpoints** tab. In the upper **Input** section, move the mouse into the **Value** cell of the **Prefix** row. Click the **Link to a data source** button. The Select Link Source dialog box opens.


Select Link Source for 'Prefix'

Link from: [✓ Available steps] Data source column Test variables

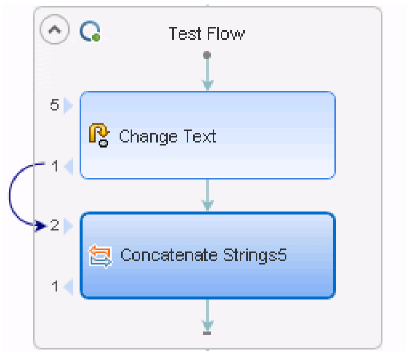
Select a step:
- Test
  - Start
  - Test Flow
    - Change Text

Select a property: Matching data types only

Output
- Properties
  - Result

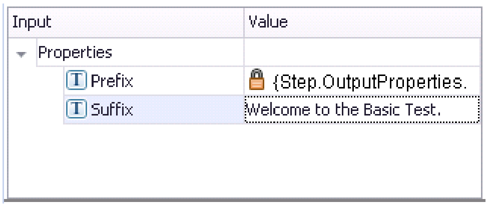
3 Link the steps.

In the Select Link Source dialog box, select the **Available steps** option. Select the **Test Flow** > **ChangeText** node. In the right pane, double-click the **Results** node. The canvas now reflects that data is moving from **Change Text** to **ConcatenateString**.



4 Configure the suffix.

In the **Properties** pane, type the text Welcome to the Basic Test. into the **Suffix** property's **Value** field.



Input	Value
Properties	
Prefix	{Step.OutputProperties.
Suffix	Welcome to the Basic Test.

5 Run the test.

Click the **Run** button or press F5 to run the test.

6 View the report.

Expand the Run Results tree and select the **ConcatenateStringsActivity** node. The report shows the result of the concatenated strings: **Goodbye World.Welcome to the Basic Test.**

When you are finished reviewing the results, close the Run Results Viewer.

How do I map data from multiple sources?

Using the Select Link Source dialog box, you can link to one or more of the following data sources to provide input values: **Available steps**, **Data source column**, and **Test variables**. In the above section, you used the **Available steps** source for one value, and manually typed in the data for another value.

You can create a custom expression to use multiple data sources as a property value. In this section, you will use the Select Link Source dialog box to create an expression for the **Suffix** property that uses both manual entry and automatic values from the **Available steps** option.

1 Set the prefix.

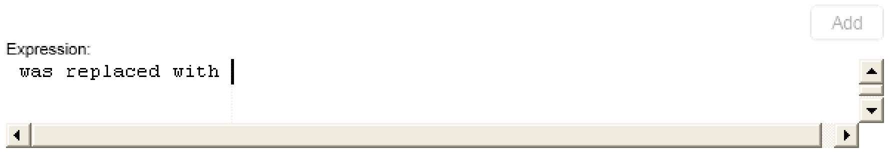
In the canvas, select the **ConcatenateString** step. Open the **Input/Checkpoints** view in the Properties pane. Click in the **Value** cell of the **Prefix** row and click the **X** to clear the contents. Type a new prefix Hello world.

2 Open the Select Link Source dialog box.

Click in the **Value** cell of the **Suffix** row and click **X** to clear its contents. Click the **Link data to source** button. The Select Link Source dialog box opens.

3 Edit the suffix.

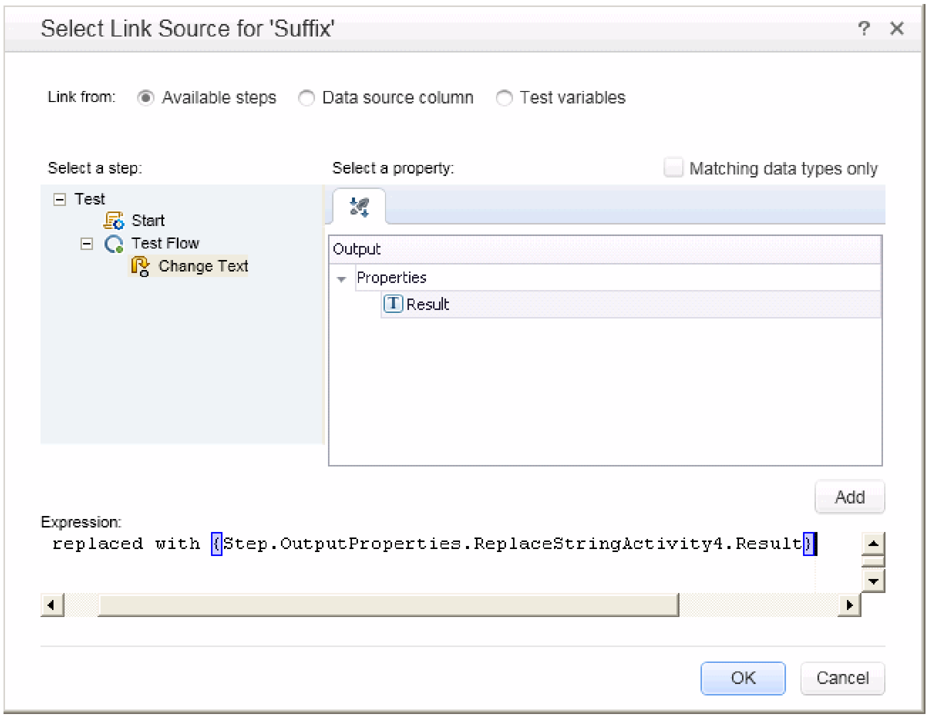
In the Select Link Source dialog box, click the **Custom Expression** button to expand it. In the **Expression** box, type the following:" was replaced with " (adding a space before and after the phrase to improve the readability).



Expression:
```
 was replaced with 
```

4 Add another source.

Select the **Available steps** option and select the **Change Text** node in the left pane. Select the **Result** node in the right pane, and click **Add**. The **Expression** box shows both sources.

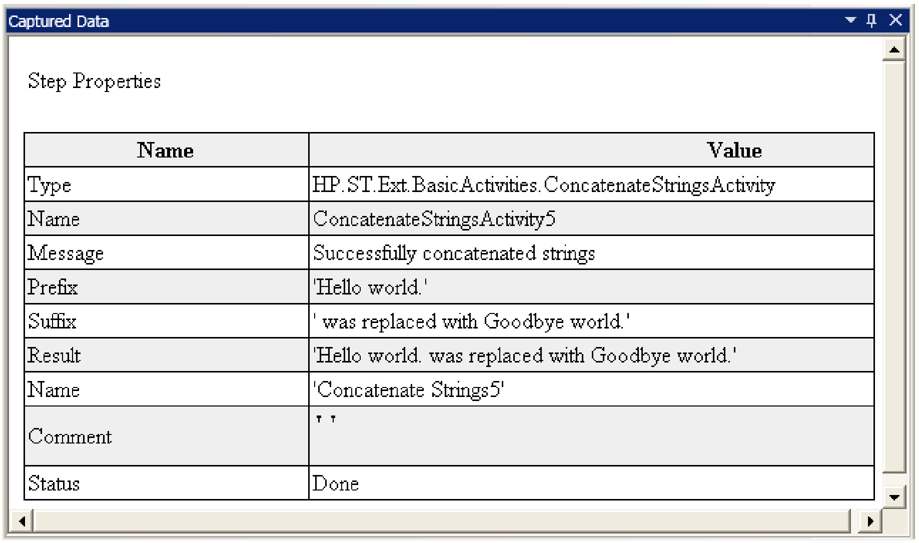


Expression:

```
replaced with {Step.OutputProperties.ReplaceStringActivity4.Result}
```

5 Run the test and view the report.

Click the **Run** button to run the test. Expand the results and select the **ConcatenateString** node. The report shows the result of the concatenated strings.



Step Properties

Name	Value
Type	HP.ST.Ext.BasicActivities.ConcatenateStringsActivity
Name	ConcatenateStringsActivity5
Message	Successfully concatenated strings
Prefix	'Hello world.'
Suffix	' was replaced with Goodbye world.'
Result	'Hello world. was replaced with Goodbye world.'
Name	'Concatenate Strings5'
Comment	' '
Status	Done

6 Close the Run Results Viewer window.

How do I data drive the step?

Data driving is the assigning of data to test steps from a data source, such as an Excel or XML file, database, or local table. The goal of data driving is to run the same business process with different values. It allows you to check your application in different scenarios, by modifying only the data tables.

To data drive test steps

1 Data drive the input arguments.

Select the **Change Text** step in the canvas. Open the **Input/Checkpoint** view in the Properties pane, and click the **Data Drive** button. The Data Driving dialog box opens.

2 Specify a data provider.

In the Data Driving dialog box:

a Set the **Data Provider** type to Excel.

b Enable data driving for **Both Input and Checkpoints**.

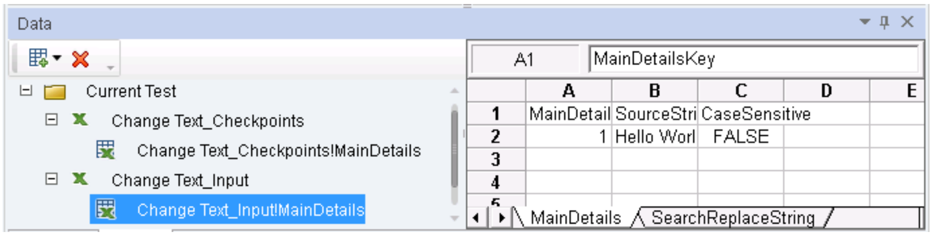
c Clear the **Configure 'Test Flow' as a ForEach loop using the new data source** option, which repeats the Test Flow according to the number of data rows. You will manually set the number of iterations in a later step.

d Click **OK** to close the Data Driving dialog box.

e Accept the popup message. The data driving mechanism replaces the constant values with the new expressions, {DataSource.Change Text_Input!MainDetails.SourceString}.

3 View the Data pane.

Make sure the Data pane is visible. If not, choose **View** > **Data**. Expand the **Change Text_Input** node and select the **Change Text_Input!MainDetails** node. The data pane shows a data table with a column for each input property, and one row of values corresponding to the property, **Hello World.** and **FALSE** (or empty check box for installations without Excel) that you entered earlier.



	A	B	C	D	E
1	MainDetail	SourceStri	CaseSensitive		
2	1	Hello Worl	FALSE		
3					
4					

4 Add new data.

Add two additional rows to the **Change Text_Input!MainDetails** table. Make sure to copy the text exactly, including punctuation where included.

MainDetailsKey	SourceString	CaseSensitive
1	Hello world.	FALSE
2	I like eating broccoli.	TRUE
3	The product version is 11.	FALSE

5 Add new search and replace data.

Click the **Text_Input!SearchReplaceString** node and add two additional rows to the table. Make sure to copy the text exactly, including punctuation where included.

MainDetailsKey	Key	Value	CaseSensitive
1	Hello	Goodbye	FALSE
2	broccoli	ice cream	TRUE
3	11	12	FALSE

6 Add checkpoint values.

Expand the **Change Text_Checkpoints** node and select the **Change Text_Checkpoints!MainDetails** node. Add values to this column as shown below.

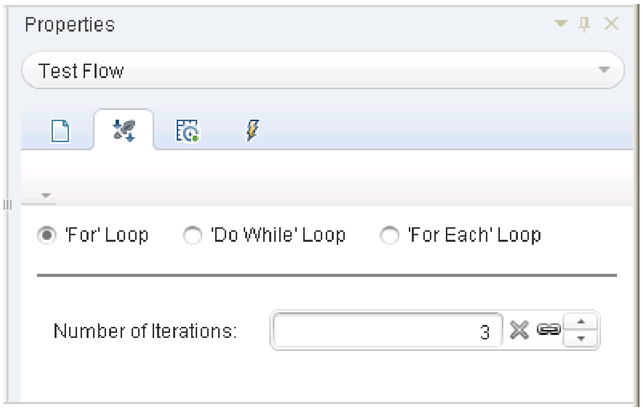
Note: In the third row, we will intentionally insert an exclamation point, !, in order to generate an error.

Result
Goodbye world.
I like eating ice cream.
The product version is 12!

7 Set the number of iterations.

The number of iterations is the number of times to repeat the step. We will set it to 3, corresponding to the number of rows of data in our table.

Return to the canvas and click inside the Test Flow frame—but not within a test step. Open the **Input/Checkpoint** view in the Properties pane. Select **'For' Loop** and set the **Number of Iterations** to **3**.



[✓ 'For' Loop] 'Do While' Loop 'For Each' Loop

Number of Iterations: 3

8 Run the test and view the report.

Click the **Run** button or press F5 to compile and run the test. The test runs three times, using the three lines of data in the table.

After the test run, the Run Results Viewer opens. Expand the **Test Flow** node and drill down to the row with the red **X**, indicating a failed checkpoint. The checkpoint failed because the expected result contained an exclamation point, which was not present in the source string.

9 Correct the error and rerun the test.

In the Data pane, correct the data **Change Text_Checkpoints!MainDetails** node. In the third row of the **Results** column, for the checkpoint, replace the exclamation point with a period.

Run the script again and verify that you have no errors in the report.

Where To Go From Here

Now that you have learned to create simple test steps, you can create steps using Web services. The following lessons will walk you through the process of importing WSDLs and creating Web service tests.

3

Test a Web Service

UFT API Testing allows you to create tests for your WSDL-based Web services.

This lesson contains the following sections:

How do I import a Web service?

A WSDL file defines the operations in a Web service. To use the WSDL file, you import it into your test. This section shows you how to import the sample application's WSDL file.

1 Start the Sample Flight application.

Make sure that the Flight Application service is available, as described in "How do I invoke the application?" on page 13.

2 Create a new solution.

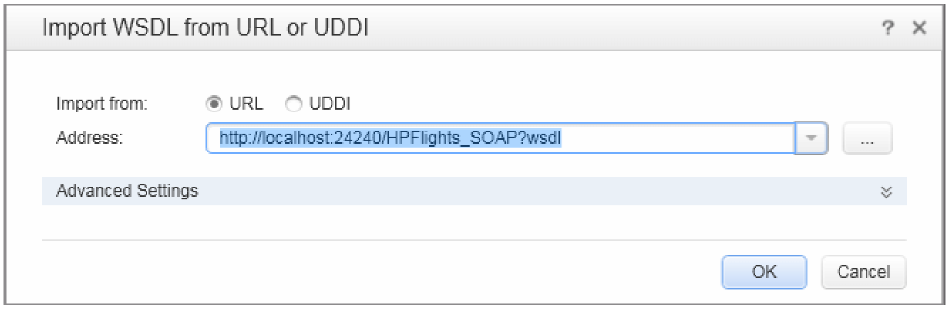
Select **File** > **New** > **Test** and specify the name **WebServiceTest** for a new API Test. Click **Create**.

3 Open the Import Service dialog box.

Select **Import WSDL** > **Import WSDL from URL or UDDI** on the toolbar.

4 Specify an import source.

Select the **URL** option and specify the location **http://localhost:24240/HPFlights_SOAP?wsdl**. Click **OK**.

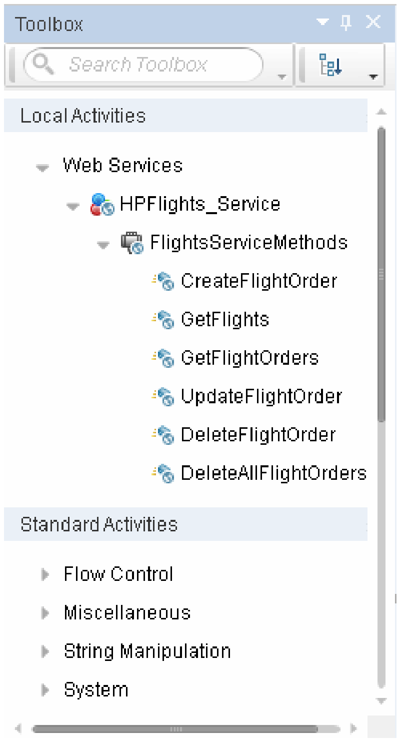


Import from: [✓ URL] UDDI

Address: http://localhost:24240/HPFlights_SOAP?wsdl

5 View the service's operations.

The import created a new branch of Web service operations in the **Toolbox**, under the **Web Services** category. Expand the node to view the operations.



How do I build a Web service test?

In this section, you will create a new flight order using the **HPFlights** Web service.

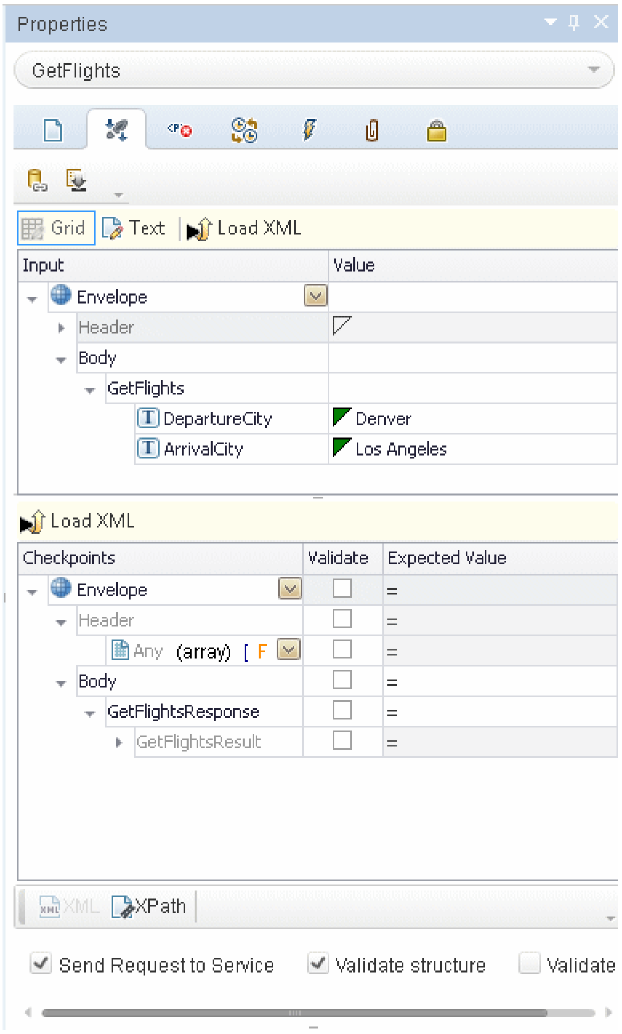
In order to create a flight order, you must first know the available flights. First you will run the **GetFlights** step that retrieves all of the flights to your destination. In the next step, you will use the first flight number returned, as input for the **CreateFlightOrder** step.

1 Create a GetFlights step.

Expand the **Web services** > **HPFlights_Service** node and drag the **GetFlights** activity into the Test Flow.

2 Assign values for DepartureCity and ArrivalCity.

Open the **Input/Checkpoints** view and expand the **Body** > **GetFlights** node. To select a city, click the arrow in the row to open a drop down list. Choose Denver as the **DepartureCity** and Los Angeles for the **ArrivalCity**.



Input	Value
Envelope	
Header	
Body	
GetFlights	
DepartureCity	Denver
ArrivalCity	Los Angeles

Checkpoints	Validate	Expected Value
Envelope	☐	=
Header	☐	=
Any (array) [F	☐	=
Body	☐	=
GetFlightsResponse	☐	=
GetFlightsResult	☐	=

☑ Send Request to Service ☑ Validate structure ☐ Validate

3 Create a CreateFlightOrder step.

Drag the **CreateFlightOrder** activity from the toolbox into the Test Flow, beneath the **GetFlights** step.

4 Set the values for the CreateFlightOrder step.

In the **Input/Checkpoints** view, expand the **Body** > **CreateFlightOrder** > **FlightOrder** node, and set the values for creating a flight order:

- ➤ **Class**—Select a class, such as Business from the dropdown list.
- ➤ **CustomerName**—any value
- ➤ **DepartureDate**—use the dropdown to open a calendar and select a date at least two days in the future.
- ➤ **FlightNumber**—leave blank for now. We will set it in the following steps.
- ➤ **NumberofTickets**—use the scroller to set any value.

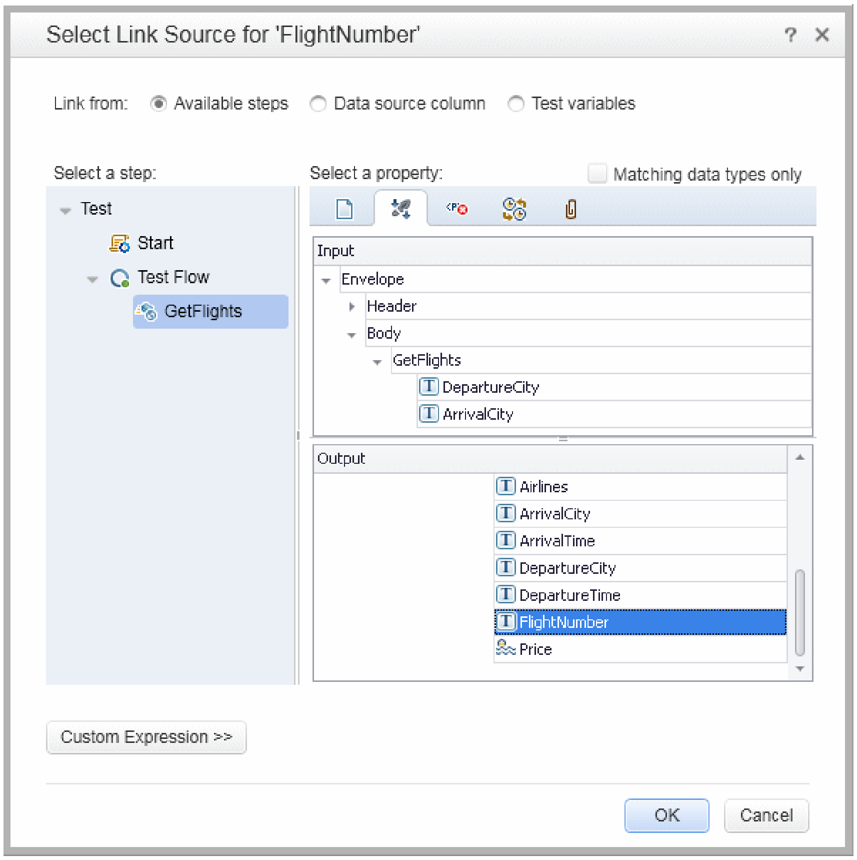
5 Link the output of GetFlights to the CreateFlightOrder step.

a Click the **Link to a data source** icon in the right corner of the **FlightNumber** row. The Select Link Source dialog box opens.

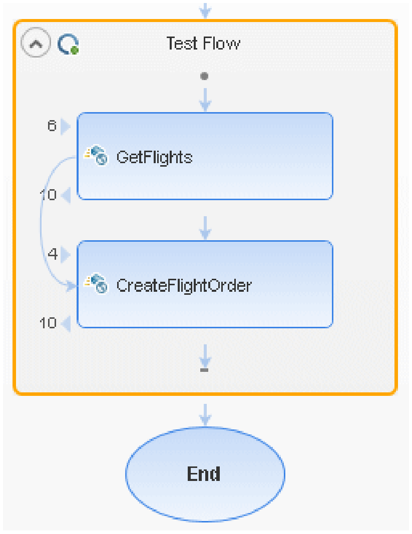
b Select **Available steps** and select the **GetFlights** node.

c In the right pane, select the **Input/Checkpoints** button. In the **Output** section, expand all nodes under the **Body** node. Click the **Add** button in the **Flight (array)** node row to create the **Flight[1]** array.

d Expand the **Flight[1]** array, select the **FlightNumber** element, and click **OK**. The application asks if you want to enclose the target step in a loop. Select **Yes**.



The canvas indicates a connection between the two steps.



6 Reset the number of iterations.

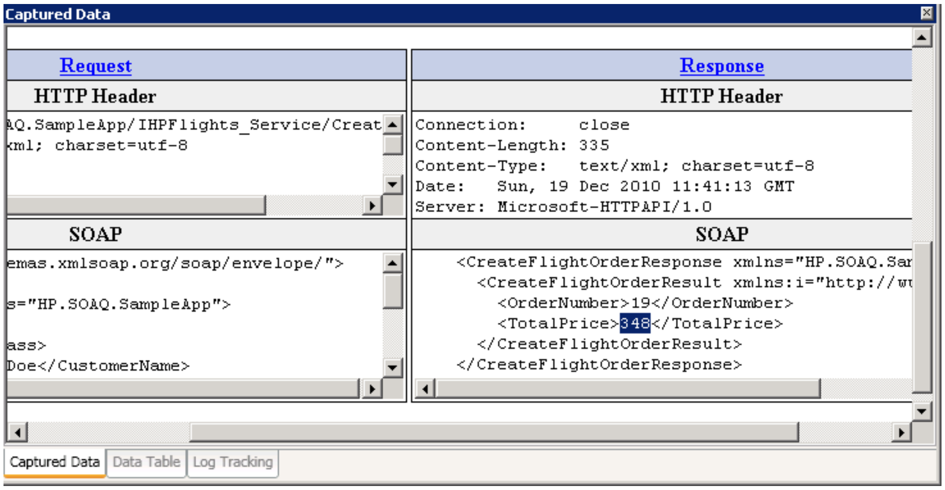
The number of iterations is the number of times to repeat the step. Return to the canvas and click inside the Test Flow frame—not within the test step. Open the Properties pane’s **Input** view. Select **For Loop** and set the **Number of Iterations** to **1**.

7 Run the test.

Click the **Run** button. Observe the log in the **Output** tab. The Run Results Viewer opens automatically.

8 Check the results.

In the left pane, click the parent node and select **Expand All** from the right-click menu. Click the **CreateFlightOrder** node. In the **Captured Data** pane, scroll down to the Web service Call HTTP Snapshot section and look at the Response pane. Note the output of the request—**OrderNumber** and **TotalPrice**. Copy the **TotalPrice** value to the clipboard for use in the next step.



```
AQ.SampleApp/IHPFlights_Service/Creat
xml; charset=utf-8
```

```
Connection:     close
Content-Length: 335
Content-Type:   text/xml; charset=utf-8
Date:   Sun, 19 Dec 2010 11:41:13 GMT
Server: Microsoft-HTTPAPI/1.0
```

```
emas.xmlsoap.org/soap/envelope/">

s="HP.SOAQ.SampleApp">

ass>
Doe</CustomerName>
```

```
    <CreateFlightOrderResponse xmlns="HP.SOAQ.Sar
      <CreateFlightOrderResult xmlns:i="http://wu
        <OrderNumber>19</OrderNumber>
        <TotalPrice>348</TotalPrice>
      </CreateFlightOrderResult>
    </CreateFlightOrderResponse>
```

Tip: Click the **Request** or **Response** links to open the SOAP in a separate browser.

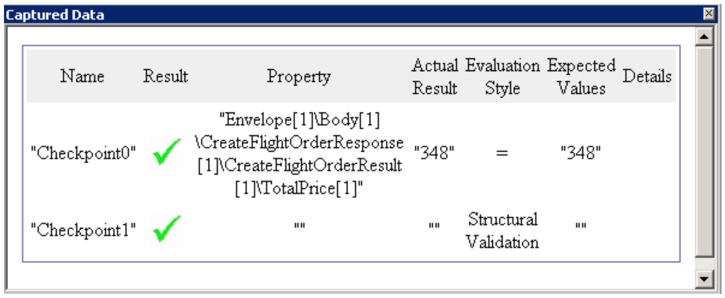
When you are finished viewing the results, close the Run Results Viewer.

9 Set a checkpoint.

Select the **CreatFlightOrder** step in the canvas. Open the Properties pane's **Input/Checkpoints** view, click in the Checkpoints grid, and expand the **CreateFlightOrderResponse** node. Paste the clipboard contents from the previous step into the **TotalPrice** field. Select the check box in the **TotalPrice** row, to include it as a checkpoint.

10 Run the test and view the checkpoint results.

Run the test again and expand the results tree. Select the **Checkpoints** node for **CreateFlightOrder**. The report shows a checkmark and indicates the expected and actual values. If the expected value was not returned by the server, the report indicates a failure.



Captured Data

Name	Result	Property	Actual Result	Evaluation Style	Expected Values	Details
"Checkpoint0"	✓	"Envelope[1]\Body[1]\CreateFlightOrderResponse[1]\CreateFlightOrderResult[1]\TotalPrice[1]"	"348"	=	"348"	
"Checkpoint1"	✓	""	""	Structural Validation	""	

When you are finished viewing the results, close the Run Results Viewer.

How do I integrate data into a test?

In this section you will learn how to integrate data from an existing source, and how to data drive the test. When you data drive a test, the Data pane automatically creates a data table whose values you can edit.

1 Import Sample Data

In the Data pane, in the bottom of the UFTwindow, select **New** > **Excel**. The Add New Excel Data Source dialog box opens.

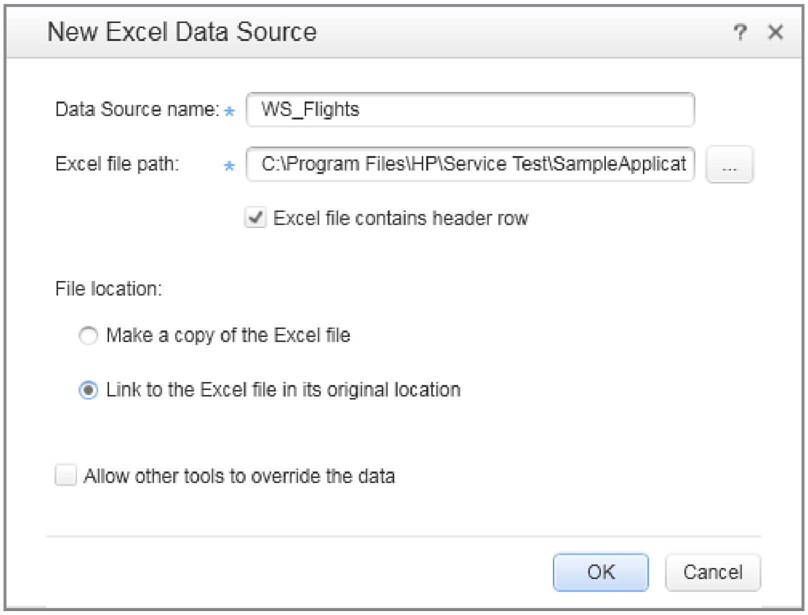
a Browse for the sample application's Excel file, **SampleAppData.xlsx**, in the <*installation directory*>/**SampleApplication** folder. By default, this folder is **C:\Program Files\HP\Unified Functional Testing\SampleApplication**.

b Enable the **Excel file contains header row** option, since the sample file contains a header row.

c Enter WS_Flights as a **Data source name.**

d Select **Referenced Data Source** as the mode of import. This links to the Excel file at its original location, so that if data changes, your data source will be current.

e Click **OK**.


New Excel Data Source

Data Source name: * WS_Flights

Excel file path: * C:\Program Files\HP\Service Test\SampleApplicat

[✓] Excel file contains header row

File location:

Make a copy of the Excel file [✓ Link to the Excel file in its original location]

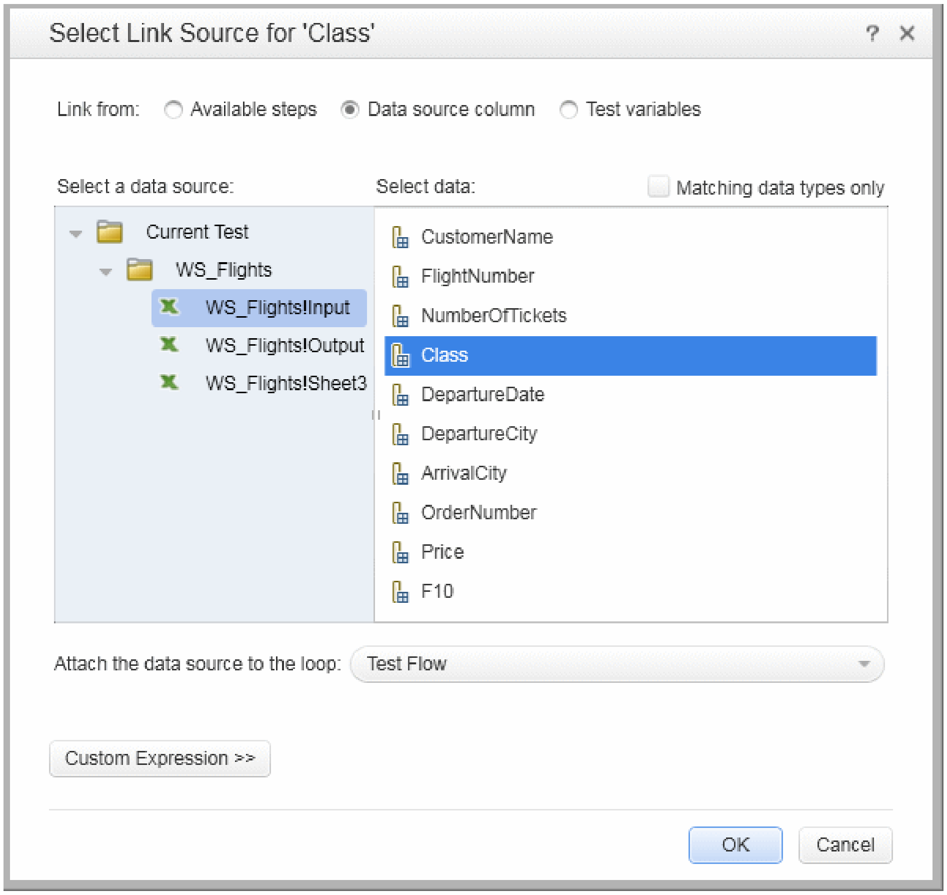
[ ] Allow other tools to override the data

2 Open the Select Link Data dialog box.

Select the **CreateFlightOrder** step in the canvas and open the **Input/Checkpoints** view. In the **Input** section, expand all the nodes and select the **Class** row. Click the **Link to a data source** icon. The Select Link Source dialog box opens.

3 Select a value from the data source.

Select the **Data source column** option.


Select Link Source for 'Class'

Link from: Available steps [✓ Data source column] Test variables

Select a data source:

- Current Test
  - WS_Flights
    - WS_Flights!Input
    - WS_Flights!Output
    - WS_Flights!Sheet3

Select data: Matching data types only

- CustomerName
- FlightNumber
- NumberOfTickets
- Class
- DepartureDate
- DepartureCity
- ArrivalCity
- OrderNumber
- Price
- F10

Attach the data source to the loop: Test Flow

4 Use the sample Excel data.

Select the **WS_Flights!Input** node, and select **Class** in the right pane. Click **OK**. This instructs the test to refer to this column in the sample data during the test run.

Repeat this for the other input parameters: **CustomerName**, **DepartureDate**, **FlightNumber**, and **NumberofTickets**.

5 Disable the checkpoint.

In the **Input/Checkpoint** view, click in the Checkpoints grid. Clear the check box for the **TotalPrice** property, to exclude it as a checkpoint.

6 Set the navigation settings.

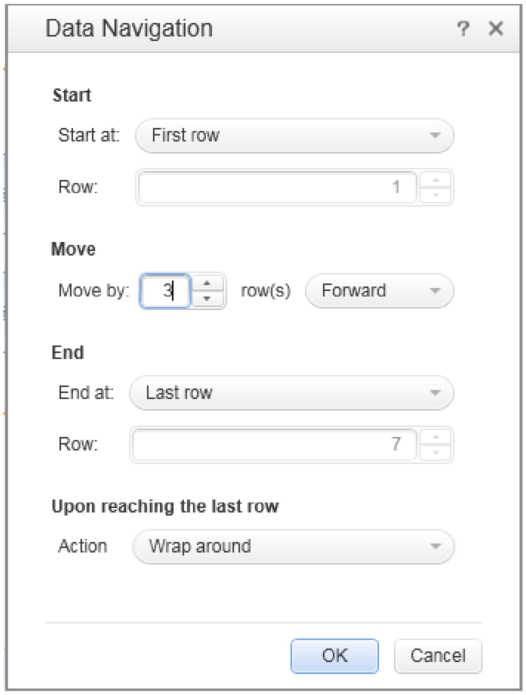
The navigation settings let you indicate how to use the data in your data source. You can specify from which row to begin, how many rows to advance, and in what direction to move for the next set of values. You can also specify what to do when reaching the end of the data table—wrap around or continue using the last line.

a In the canvas, click in the **Test Flow** but not within a step.

b In the Properties pane, click the **Data Sources** view button.

c Select the **WS_Flights!Input** node and click **Edit** to open the Data Navigation dialog box.

d Specify the data navigation details: **Start at**: First row, **Move**: Move by 3 rows Forward, **End at**: Last row, and **Upon reaching the last row**: Wrap around.



Start

Start at: First row

Row: 1

Move

Move by: 3 row(s) Forward

End

End at: Last row

Row: 7

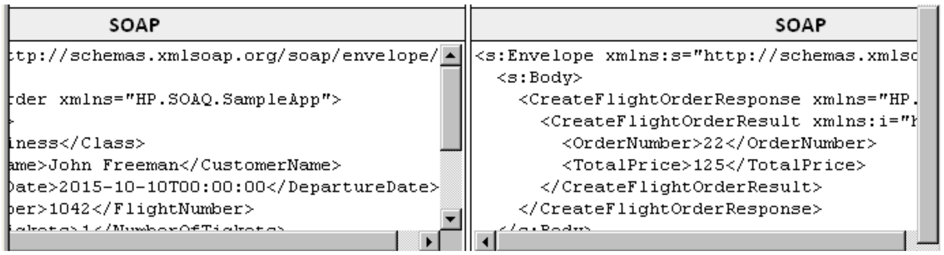
Upon reaching the last row

Action: Wrap around

e Click **OK**.

7 Run the test and view the results.

Click the **Run** button and observe the results in the Output window. The Run Results Viewer opens automatically. Expand the result tree and select the **CreateFlightOrder** step. Scroll down within the **Captured Data** tab and note the data from the Excel file in the SOAP request (left pane), and the result in the SOAP response (right pane).



```
tp://schemas.xmlsoap.org/soap/envelope/

der xmlns="HP.SOAQ.SampleApp">

iness</Class>
ame>John Freeman</CustomerName>
Date>2015-10-10T00:00:00</DepartureDate>
ber>1042</FlightNumber>
```

```
<s:Envelope xmlns:s="http://schemas.xmlsc
  <s:Body>
    <CreateFlightOrderResponse xmlns="HP.
      <CreateFlightOrderResult xmlns:i="h
        <OrderNumber>22</OrderNumber>
        <TotalPrice>125</TotalPrice>
      </CreateFlightOrderResult>
    </CreateFlightOrderResponse>
```

When you are finished viewing the results, close the Run Results Viewer.

How do I use multiple data sources and custom code?

This section describes how to define data using multiple data sources and sending information to the report through a custom code step.

1 Create a new test.

Create a new test called WebServicesCustom and import the HP Flights Services WSDL as described in “How do I import a Web service?” on page 32.

2 Create test steps.

Drag the activities into the canvas in the following order: From the **Web services** folder: **GetFlights** and **CreateFlightOrder**. From the **Miscellaneous** folder, drag in **Custom Code**.

3 Add a data source.

In the Data pane, select **New > Excel**. In the Add New Excel Data Source dialog box:

a Browse for the sample application Excel file. By default, this folder is **C:\Program Files\HP\Unified Functional Testing\SampleApplication**.

b Select the **Excel file contains header row** check box.

c Enter WS_Flights as a **Data source name**.

d Select the **Referenced data source** mode.

4 Assign values for GetFlights.

Select the **GetFlights** step in the canvas and open the **Input/Checkpoints** view in the Properties pane. In the **Input** section, select **DepartureCity**= Denver, and **ArrivalCity**=Los Angeles.

5 Assign values for the CreateFlightOrder.

Select the **CreateFlightOrder** activity in the canvas and open the **Input/Checkpoints** view. Expand the **Body** > **FlightOrder** node and set the input properties as follows:

➤ **Class**. Economy

➤ **CustomerName**. Click the Link to a data source button in the right corner of the **CustomerName** row. The Select Link Source dialog box opens. Select **Data source column**, and expand the tree to show the **WS_Flights!Input** node. In the right pane, select the **CustomerName** parameter. Click **OK**.

➤ **DepartureDate**. A date in the following format YYYY-MM-DDTHH:MM:SS For example, 2015-02-18T00:00:00. Use the drop down arrow to open the calendar. The date must be at least two days ahead of the current date.

➤ **NumberofTickets**. 3

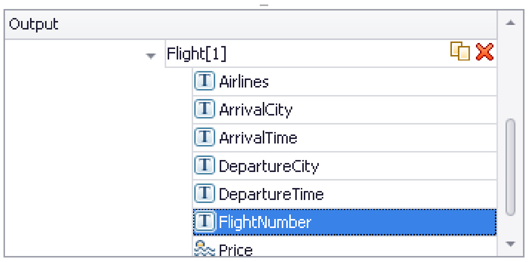
➤ **FlightNumber**. Link from the previous step:

a Click the **Link to a data source** button in the right corner of the **FlightNumber** row.

b In the Link to Source dialog box, select **Available steps**, expand the **Test Flow** branch, and click **GetFlights**.

c In the right pane, select the **Input/Checkpoints** button.

d In the **Output** section, click the **Add** button in the **Flight (array)** node row to create the **Flight[1]** array. Expand the array, select **FlightNumber**, and click **OK.**



6 Create a property for the custom code step.

Select the **Custom Code** activity in the canvas and open the **Input/Checkpoint** view in the Properties pane. Expand the **Add Property** toolbar button and select **Add Input Property**. Create a new **String** type property called **FlightInfo**.

7 Define values for the custom code step.

In this step you will define a value using multiple sources. In this example, you will set a value which is a combination of the **CustomerName**, a constant string, and the **OrderNumber**.

a Click the **Link to a data source** button in the right corner of the **FlightInfo** row. The Select Link Source dialog box opens.

b Click **Custom Expression** to show the Expression area.

c Select the **Data source column** option. In the tree's **WS_Flights!Input** node, select **CustomerName**. Click **Add**.

d In the **Expression** area, type _OrderNumber_ (with the underscores) after the existing expression.

e Select **Available steps** and expand the **Test Flow** branch. Select the **CreateFlightOrder** node. In the right pane, select the **Input/Checkpoints** button. In the lower pane, expand the Output **Body** node, expand the tree, select the **OrderNumber** element, and click **Add**.

The CustomCode input property, **FlightInfo**, has the following value:

```
{DataSource.WS_Flights!Input.CustomerName}_OrderNumber_{Step.OutputPropert
ies.StServiceCallActivity(x).Body.CreateFlightOrderResponse.CreateFlightOrderRes
ult.OrderNumber}
```

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

8 Create an event.

In this step you will create an event handler in order to use the Application Program Interface (API). You can add C# code to this section. Defining events let you adapt your test to your custom requirements, and perform actions that are not built-in to UFT. In this example, you will add code that sends a custom string to the report.

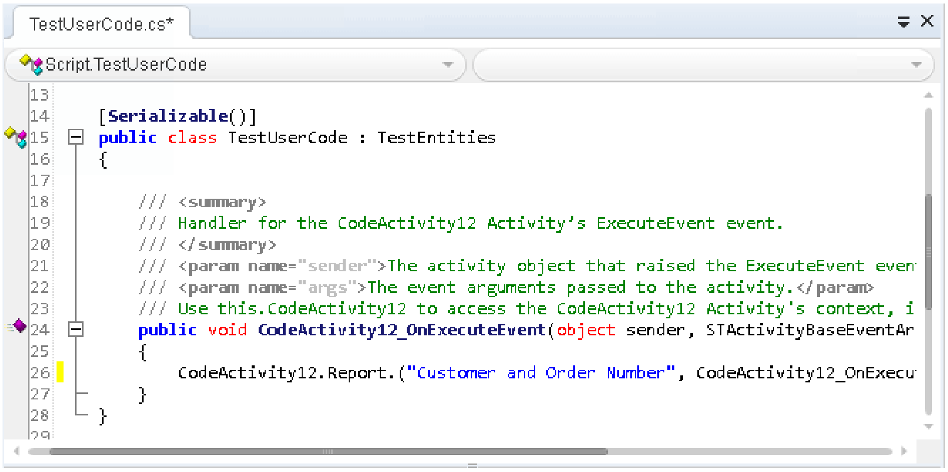
Select the **Custom Code** step in the canvas. In the Properties pane, click the **Events** button. In the **ExecuteEvent** row, click the drop down arrow and select **Create a default handler**. UFT creates an event called **CodeActivity(x)_OnExecuteEvent** and opens a new tab **TestUserCode.cs.**

9 Edit the "Todo" section, using the Index assigned to the activity.

Replace the commented text in the **Todo** section with the following:

```
CodeActivity(x).Report("Customer and Order
Number",CodeActivity(x).Input.FlightInfo);
```

In the following example, the index assigned to the event was 12, so the string is CodeActivity12.Report("Customer and Order Number",CodeActivity12.Input.FlightInfo);



```
    [Serializable()]
    public class TestUserCode : TestEntities
    {

        /// <summary>
        /// Handler for the CodeActivity12 Activity's ExecuteEvent event.
        /// </summary>
        /// <param name="sender">The activity object that raised the ExecuteEvent even
        /// <param name="args">The event arguments passed to the activity.</param>
        /// Use this.CodeActivity12 to access the CodeActivity12 Activity's context, i
        public void CodeActivity12_OnExecuteEvent(object sender, STActivityBaseEventAr
        {
            CodeActivity12.Report.("Customer and Order Number", CodeActivity12_OnExecu
        }
    }
```

1 Run test and check the results.

Drill down in the results to the **Custom Code** step. Note the new entry in the **Captured Data** pane: Customer and Order Number.

Tip: You can also use the **Report Message** activity under the **Miscellaneous** folder, to send text and property values to the report.

Where To Go From Here

Now that you have learned to create a test for a Web service, you can relate this to other types of services and application components. The next lesson will walk you through the process of creating a test for a REST service.

4

Test REST Services

Using UFT, you can model and create tests for REST services.

This lesson contains the following sections:

- ➤ How do I create a REST service activity?
- ➤ How do I run a REST test?
- ➤ How do I assign data to my REST method?
- ➤ How do I check my output?
- ➤ How do I resolve changes in my REST service?

How do I create a REST service activity?

This section describes how to model a REST service activity using the sample application. A best practice for working with REST services is to create a reusable prototype method for the REST service. Once you create this method, you can reuse it in different test steps or separate tests.

1 Start the Sample Flight application.

Make sure that the Flight Application service is running, as described in“How do I invoke the application?” on page 13.

2 Obtain the REST service modelling document.

In the Sample Application command window, type **h**, and press ENTER. A browser opens with the modelling information for the REST service. This file, **index.htm**, is located in the **<installation_folder>\SampleApplication\Help** folder.

3 Save the Request body to a file.

Copy the Request Body for **FlightOrders** > **ReserveOrder (POST)** to the clipboard. Only copy the XML code.

```
<FlightOrderDetails xmlns="HP.SOAQ.SampleApp">
  <Class>Business</Class>
  <CustomerName>John Doe</CustomerName>
  <DepartureDate>2012-12-12</DepartureDate>
  <FlightNumber>1304</FlightNumber>
  <NumberOfTickets>21</NumberOfTickets>
 </FlightOrderDetails>
```

Create a new file in a text editor and paste the contents of the clipboard. Save the file as **body.xml** to any location.

4 Obtain the REST service endpoint URL.

Return to the browser window and copy the URL for the **FlightOrders** > **ReserveOrder (POST)** operation, http://localhost:24240/HPFlights_REST/FlightOrders/, to the clipboard.

5 Create a new test.

- **a** Select **File** > **New** > **Test** and select an **API Test** type.
- **b** Specify the name **RESTServiceTest** for the new test.
- **c** Click **Create**.

6 Add a REST service.

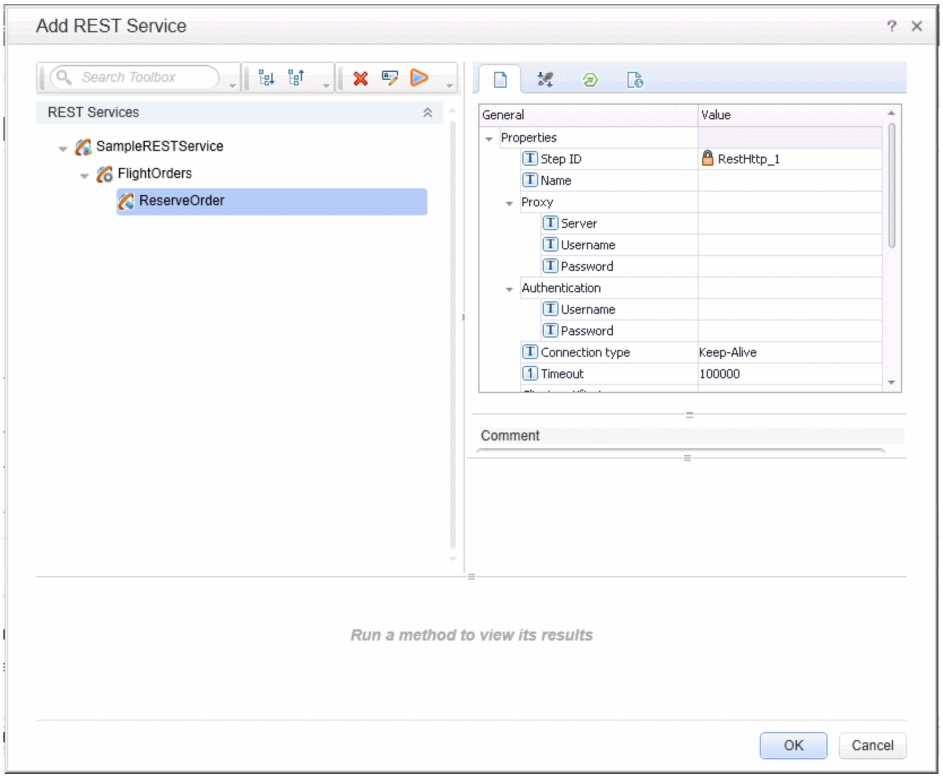
- **a** Click the **Add REST Service** toolbar button. The Add REST Service dialog box opens.
- **b** In the left pane, click on the **New Service** node and rename it to **SampleRESTService**.

7 Add a resource.

Click the toolbar's **Add Resource** button and rename the resource to **FlightOrders**.

8 Add a method to the REST service.

Click the **Add Method** button and rename the method to **ReserveOrder**.



General	Value
Properties	
Step ID	RestHttp_1
Name	
Proxy	
Server	
Username	
Password	
Authentication	
Username	
Password	
Connection type	Keep-Alive
Timeout	100000

9 Configure the method's HTTP properties.

Open the **HTTP Input/Checkpoints** view in the right pane.

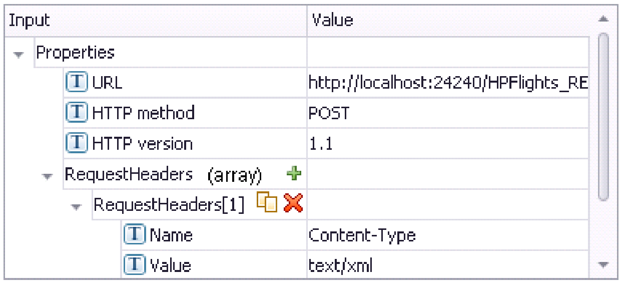
a Paste the contents of the clipboard into the **URL** row.

b Set the **HTTP method** to **POST**.

c Click the **Add** button in the **Request Headers (array)** row to add an array element.

d Expand the **Request Headers** array. Using the method's details from the Help page, set the header name and value as follows:

- ➤ **Name** row—Content-Type
- ➤ **Value** row—text/xml



Input	Value
Properties	
URL	http://localhost:24240/HPFlights_RE
HTTP method	POST
HTTP version	1.1
RequestHeaders (array)	
RequestHeaders[1]	
Name	Content-Type
Value	text/xml

10 Create input properties.

a Open the **Input/Checkpoint** view in the right pane.

b Select **Add** > **Add Input Property**.

c Add a **String** type property called **Class**.

d Add another **String** type property called **Customer_Name**.

e Add another **Date/Time** type property called **Departure_Date**.

f Add an **Int** type property called **Flight_Number**.

g Add another **Int** type property called **Number_of_Tickets**.

11 Create Output properties.

a Select **Add** > **Add Output Property**.

b Add a **Int** type property called **Total_Price**.

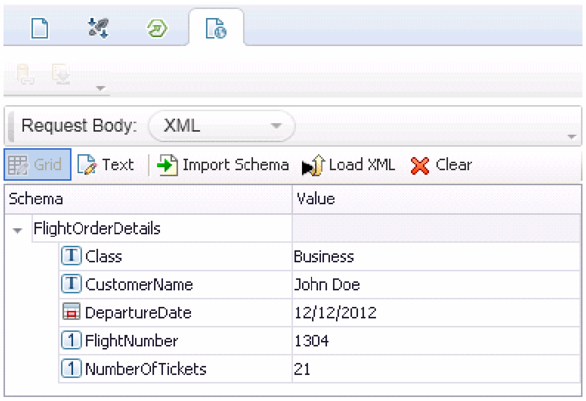
c Add another **Int** type property called **Order_Number**.

12 Import the Request body.

a Open the **HTTP** view in the right pane.

b Select **XML** as the **Body** type.

c Click the **Load XML** button and load the **body.xml** file that you saved earlier.



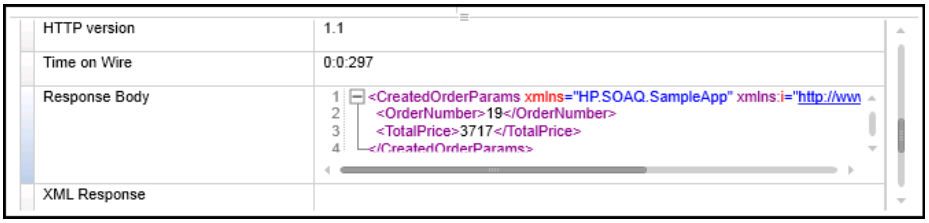
Request Body: XML

Grid | Text | Import Schema | Load XML | Clear

Schema	Value
FlightOrderDetails	
Class	Business
CustomerName	John Doe
DepartureDate	12/12/2012
FlightNumber	1304
NumberOfTickets	21

13 Test the method.

Click the **Run Method** toolbar button to check the validity of the method. Scroll through the results and verify that the response body contains an order number and price.



HTTP version	1.1
Time on Wire	0:0:297

Response Body	`<CreatedOrderParams xmlns="HP.SOAQ.SampleApp" xmlns:i="http://ww`
	`<OrderNumber>19</OrderNumber>`
	`<TotalPrice>3717</TotalPrice>`
	`</CreatedOrderParams>`

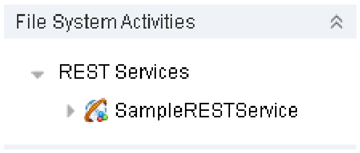
XML Response	

14 Add the method to the Toolbox.

Click **OK** in the Add REST Service dialog box. The REST designer adds the REST service, along with its resource and method to the **Toolbox** pane, under the **Local Activities** category.

15 Share the REST activity to make the activity available for all tests.

In the **Toolbox** pane, select the parent node of the REST service, **SampleRESTService**, and select **Move to** > **File System Activities** from the right-click menu. The REST service activity is now moved to the **File System Activities** section in the **Toolbox** pane.



You have now created a prototype activity for your REST service, complete with input parameters and the HTTP information. You can now drag the activity into tests and run it with little intervention.

How do I run a REST test?

This section describes how to run the REST service method created in the previous section. You will incorporate data into the test, using the sample data file included with the product.

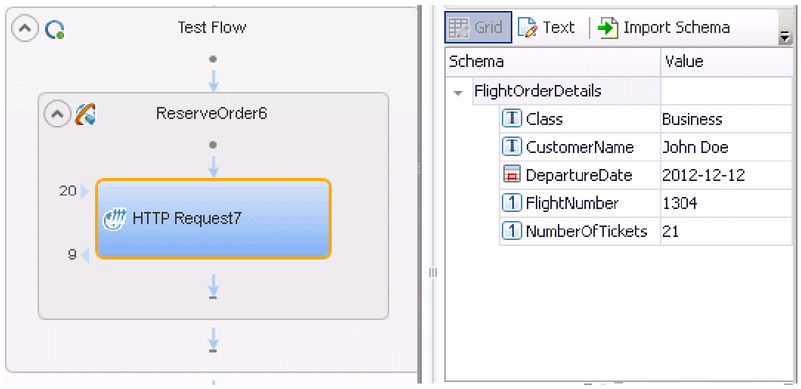
1 Create a test step.

Double-click the **File System Activities** > **REST Services** > **SampleRESTService** > **FlightOrders** > **ReserveOrder** method to add it to the **Test Flow** on the canvas.

2 Expand the internal HTTP Request of the REST method wrapper.

Click the arrow in the left side of the **ReserveOrder** method frame, to view the inner **HTTP Request** frame. Click within the **HTTP Request** frame and open the **HTTP** view in the Properties pane.

The property values are those which we imported earlier from the **body.xml** file. These values were used in the test run that we just performed.



Schema	Value
FlightOrderDetails	
Class	Business
CustomerName	John Doe
DepartureDate	2012-12-12
FlightNumber	1304
NumberOfTickets	21

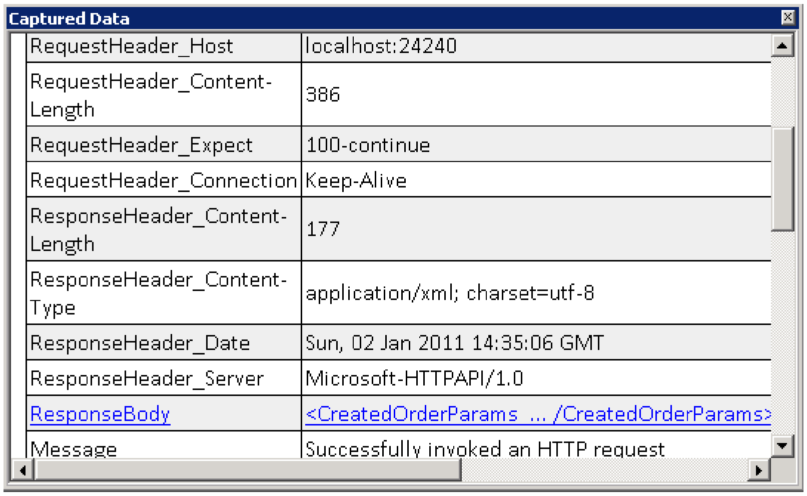
3 Run the test.

Select **Run** > **Run** to run the test.

4 View the report.

In the Run Result Viewer's left pane, click the **Expand All** button or select **Expand All** from the right-click menu and select the **HTTP Request** node.

In the **Captured Data** pane, click the **ResponseBody** link to open the response in a separate browser window.



RequestHeader_Host	localhost:24240
RequestHeader_Content-Length	386
RequestHeader_Expect	100-continue
RequestHeader_Connection	Keep-Alive
ResponseHeader_Content-Length	177
ResponseHeader_Content-Type	application/xml; charset=utf-8
ResponseHeader_Date	Sun, 02 Jan 2011 14:35:06 GMT
ResponseHeader_Server	Microsoft-HTTPAPI/1.0
ResponseBody	<CreatedOrderParams ... /CreatedOrderParams>
Message	Successfully invoked an HTTP request

Verify that the **Response Body** contains values for the **OrderNumber** and **TotalPrice** elements. This corresponds to the operation's description in the REST Service Help page that indicated the following: It creates a new flight order and returns OrderNumber and TotalPrice.

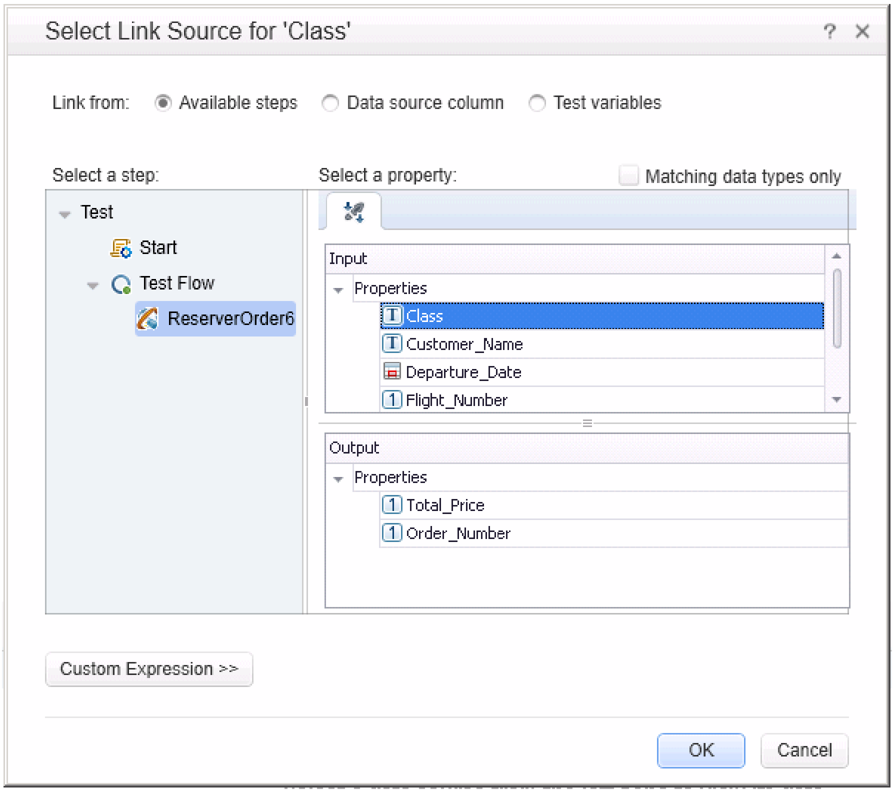
```
- <CreatedOrderParams xmlns="HP.SOAQ.SampleApp"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<OrderNumber>18</OrderNumber>
<TotalPrice>3717</TotalPrice>
</CreatedOrderParams>
```

How do I assign data to my REST method?

In order to assign data to our method, we will link the inner HTTP Request properties to the outer REST wrapper properties. This allows us to control the properties of the request message from the higher level, the REST method level.

To assign data to the REST method step:

1. Select the inner **HTTP Request** frame and open the **HTTP** view in the Properties pane.
2. Click the **Link to a data source** button in the right corner of the **Class** row. The Select Link Source dialog box opens.
3. Select **Available steps** as a data source, select the **ReserveOrder** node in the left pane, and choose the **Class** property in the right pane. Click **OK**.


Select Link Source for 'Class'

Link from: [✓ Available steps] Data source column Test variables

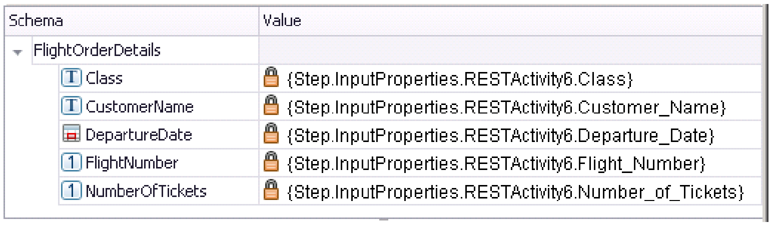
Select a step:
- Test
  - Start
  - Test Flow
    - ReserverOrder6

Select a property: Matching data types only

Input
- Properties
  - Class
  - Customer_Name
  - Departure_Date
  - Flight_Number

Output
- Properties
  - Total_Price
  - Order_Number

4 Repeat the above step for the other input properties: **CustomerName**, **DepartureDate**, **FlightNumber**, and **NumberofTickets**. The resulting HTTP view shows the new links.



Schema	Value
FlightOrderDetails	
Class	{Step.InputProperties.RESTActivity6.Class}
CustomerName	{Step.InputProperties.RESTActivity6.Customer_Name}
DepartureDate	{Step.InputProperties.RESTActivity6.Departure_Date}
FlightNumber	{Step.InputProperties.RESTActivity6.Flight_Number}
NumberOfTickets	{Step.InputProperties.RESTActivity6.Number_of_Tickets}

5 Import sample data.

We will import a data source whose values we will use for the REST method properties.

a In the **Data** pane located in the lower section of the window, expand the Data Source button and select **Excel**. The New Excel Data Source dialog box opens.

b Click the Browse button adjacent to the **Excel file path** field, and browse to the sample application's **SampAppData** Excel file in the <*installation directory*>\SampleApplication folder.

c Select the **Excel file contains header row** check box, because the sample contains a header row.

d In the **Data source name** field, enter REST_Flights.

e Select the **Make a copy of the Excel file** option. This saves a copy of the data file with the test.

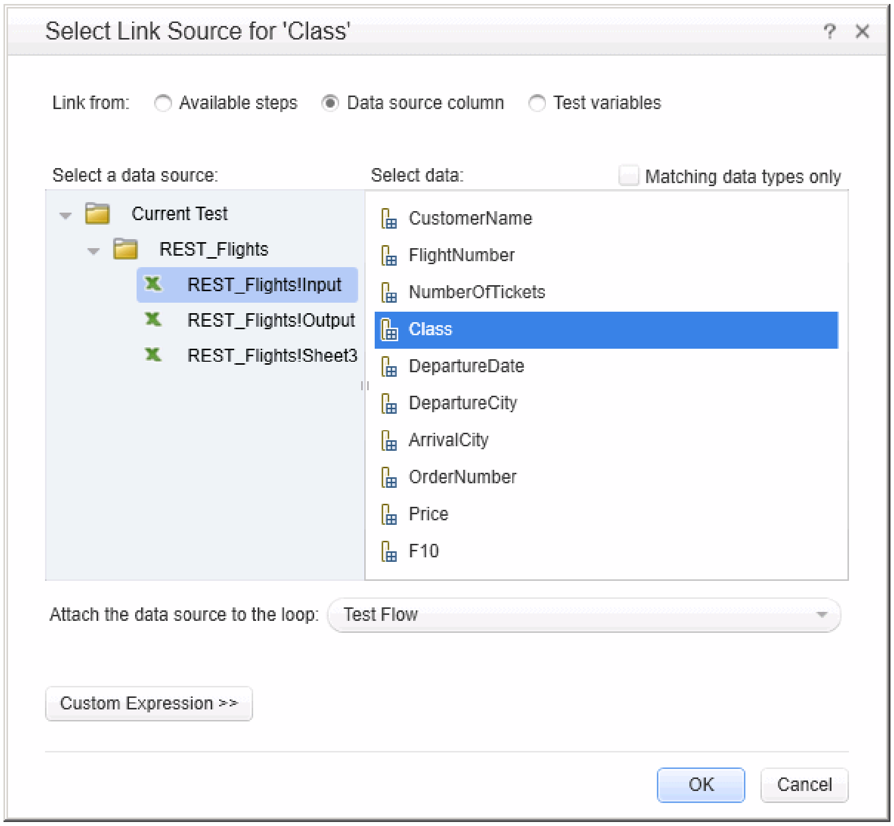
f Click **OK**. The data is imported into the Data pane.

6 Link the REST method properties to the data source.

a Select the outer **ReserveOrder** frame and open the **Input/Checkpoint** view in the right pane.

b Click the **Link to a data source** button in the rightmost corner of the **Class** row. The Select Link Source dialog box opens.

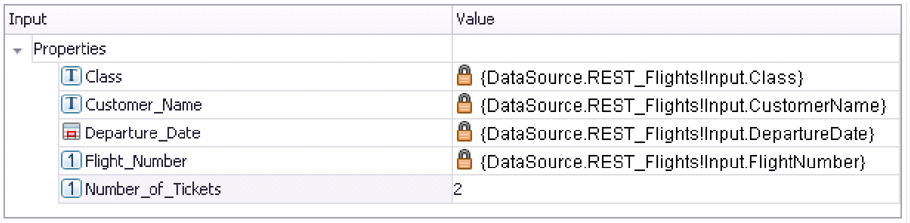
c Select **Data source column** as a data source, select the **ReserveOrder** node in the left pane, and choose the **Class** property in the right pane. Click **OK**.



d Repeat the above step for the other input properties: **Customer_Name**, **Departure_Date**, and **Flight_Number.**

e Enter 2 for the **Number_of_Tickets.**

f The resulting properties view shows the new links.



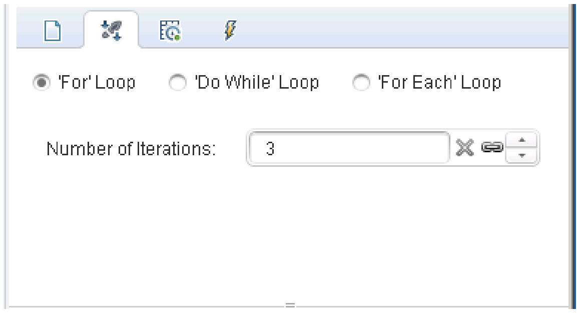
Input	Value
Properties	
Class	{DataSource.REST_Flights!Input.Class}
Customer_Name	{DataSource.REST_Flights!Input.CustomerName}
Departure_Date	{DataSource.REST_Flights!Input.DepartureDate}
Flight_Number	{DataSource.REST_Flights!Input.FlightNumber}
Number_of_Tickets	2

7 Set the iterations.

By setting iterations, we will see how our REST method used multiple sets of data from the data source.

a Select the **Test Flow** frame in the canvas and open the **Input/Checkpoint** view in the right pane.

b Set a **For Loop** with 3 iterations.



[✓ 'For' Loop] 'Do While' Loop 'For Each' Loop

Number of Iterations: 3

8 Run the test.

Select **Run** > **Run** to run the test.

9 Verify that the request used the table data.

Run the test. In the Run Result Viewer's left pane, click the **Expand All** button and select the **HTTP Request** nodes. Click the **Request Body** link in the Captured Data pane. Note that the test used the data from the Data pane for the properties that we assigned: Class, CustomerName, DepartureDate, and FlightNumber.

```
- <FlightOrderDetails xmlns="HP.SOAQ.SampleApp">
<Class>Business</Class>
<CustomerName>John Freeman</CustomerName>
<DepartureDate>2012-12-12</DepartureDate>
<FlightNumber>1304</FlightNumber>
<NumberOfTickets>21</NumberOfTickets>
</FlightOrderDetails>
```

10 Save the response data.

We will save the response data from this run for use in future steps.

a In the Run Result Viewer's left pane, select an **HTTP Request** node. Click the **Response Body** link in the Captured Data pane. A browser window opens with the XML response.

b Save the entire contents of the window to a file Response.xml. Close the Run Results Viewer.

How do I check my output?

In order to verify that the output of our REST method is correct, we will set checkpoints.

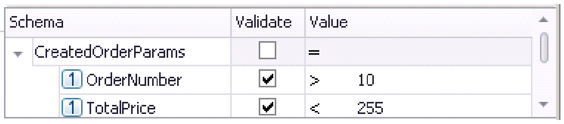
1 Insert a checkpoint.

a Click on the inner frame, **HTTP Request**. Open the **HTTP** view in the Properties pane.

b The lower section of the pane contains the output properties or output schema. We will use these as our checkpoints to check the server response. In the lower pane, select **XML** from the **Body** drop-down.

c Click the **Load XML** button and load the **Response.xml** file that you saved earlier. Select the **Validate** checkbox in the **OrderNumber** and **TotalPrice** rows.

d Set the value of **OrderNumber** to Greater than (>) 10, and the value of **TotalPrice** to Less than (<) 255.



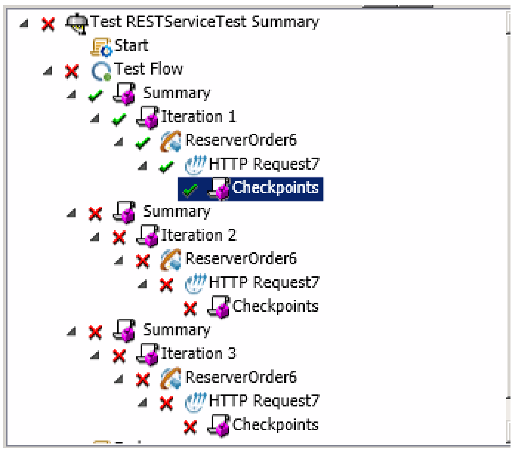
Schema	Validate	Value
CreatedOrderParams	☐	=
OrderNumber	✓	> 10
TotalPrice	✓	< 255

2 Run the test.

Select **Run** > **Run** to run the test.

3 Verify that the checkpoint passed.

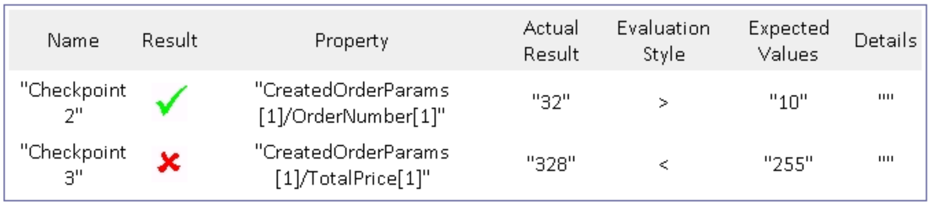
Run the test. In the Run Result Viewer's left pane, click the **Expand All** button Note that some of the checkpoints passed, while others did not.



- ✗ Test RESTServiceTest Summary
  - Start
  - ✗ Test Flow
    - ✓ Summary
      - ✓ Iteration 1
        - ✓ ReserverOrder6
          - ✓ HTTP Request7
            - ✓ Checkpoints
    - ✗ Summary
      - ✗ Iteration 2
        - ✗ ReserverOrder6
          - ✗ HTTP Request7
            - ✗ Checkpoints
    - ✗ Summary
      - ✗ Iteration 3
        - ✗ ReserverOrder6
          - ✗ HTTP Request7
            - ✗ Checkpoints

4 Determine why checkpoints failed.

Select one of the failed checkpoints nodes. In the right pane, view the Captured Data pane and note the Actual Results and Expected Values. In the following example, the **OrderNumber** was valid, but the **TotalPrice** was not valid since it exceeded 255.



Name	Result	Property	Actual Result	Evaluation Style	Expected Values	Details
"Checkpoint 2"	✓	"CreatedOrderParams [1]/OrderNumber[1]"	"32"	>	"10"	""
"Checkpoint 3"	✗	"CreatedOrderParams [1]/TotalPrice[1]"	"328"	<	"255"	""

5 Save the test.

Close the Run Results Viewer and save the test.

How do I resolve changes in my REST service?

Initially, we created a prototype REST service method **ReserveOrder**, with specific properties, such as the URL and property names. If these properties changed after you created a test, your test step will no longer match the prototype. The Resolve Conflict wizard detects changes in the method's properties and helps you resolve them.

1 In the Toolbox pane, select the **ReserveOrder** node and choose **Edit Service** from the right-click menu. The Edit REST Service dialog box opens.

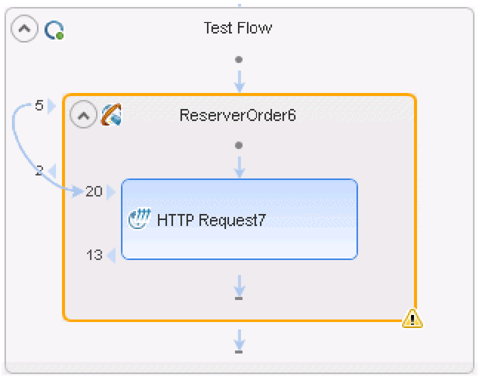
2 Select the **ReserveOrder** method and open the **Custom Input/Checkpoint** view in the **Properties** pane.

3 Select the **Flight_Number** property, and click the **Edit Property** toolbar button. In the Edit Property dialog box, rename the property to **Flight_Number_1**. Click **OK**.

4 Select the **Number_of_Tickets** property, and click the **Delete Property** toolbar button. Confirm the warning and click **OK**.

5 Click **OK** in the Edit REST Service dialog box.

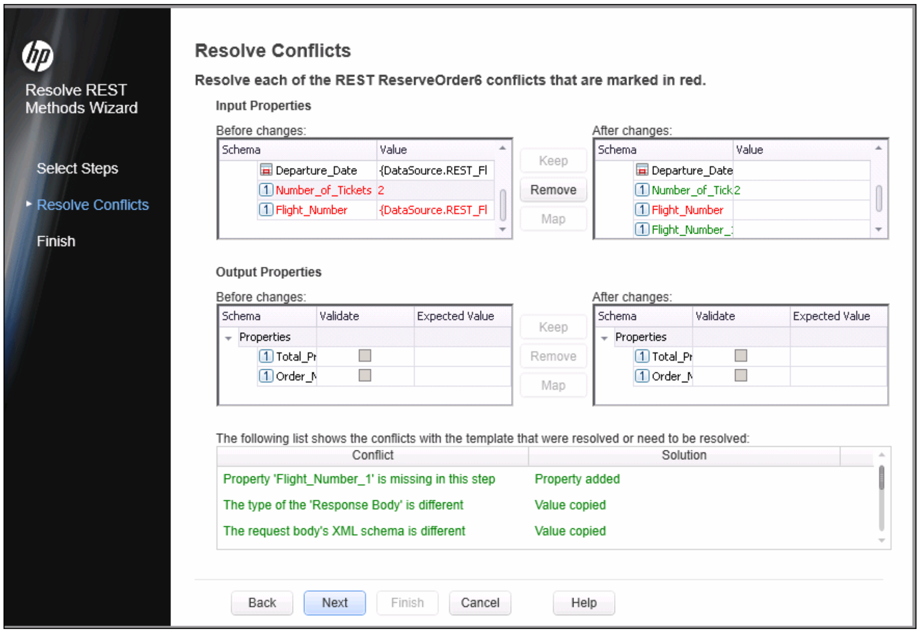
6 View the canvas. Note the alert icon in the bottom right corner of the REST method frame.



7 Click on the drop-down arrow adjacent to the alert icon, and select the text message: This step should be resolved. Resolve step.The Resolve REST Methods Wizard opens.

8 The wizard's first screen shows the problematic steps. If multiple steps were affected, you could choose which steps to resolve and which to ignore. Click **Next**.

9 In the Resolve Conflicts screen, select the **Number_of_Tickets** property in the left pane. Click **Keep**. This instructs the existing step to keep the property, even though it was removed from the method's prototype.



Resolve Conflicts

Resolve each of the REST ReserveOrder6 conflicts that are marked in red.

Input Properties

Before changes:

Schema	Value
Departure_Date	{DataSource.REST_Fl
Number_of_Tickets	2
Flight_Number	{DataSource.REST_Fl

After changes:

Schema	Value
Departure_Date	
Number_of_Tick	2
Flight_Number	
Flight_Number_:	

Output Properties

Before changes:

Schema	Validate	Expected Value
Properties		
Total_Pr	☐	
Order_N	☐	

After changes:

Schema	Validate	Expected Value
Properties		
Total_Pr	☐	
Order_N	☐	

The following list shows the conflicts with the template that were resolved or need to be resolved:

Conflict	Solution
Property 'Flight_Number_1' is missing in this step	Property added
The type of the 'Response Body' is different	Value copied
The request body's XML schema is different	Value copied

10 In the **After changes** pane, select the new property **Flight_Number_1** property and click **Remove**.

Scroll through the lower section of the wizard screen to see a log of all of the conflicts and their resolutions.

11 Click **Next.** Click **Finish** to close the wizard and return to your test.

Where To Go From Here

Now that you have learned to create tests with standard activities, Web services, and REST services, you can create your own tests for your GUI-less applications.