



OpenSpan Elective Training

Service Enabling OpenSpan Projects

- **CHAPTER 1:** Using Desktop Service Enablement
- **CHAPTER 2:** Creating a Service Enablement Project with Single Input and Result
- **CHAPTER 3:** Creating a Service Enablement Project with Multiple Inputs and Results
- **CHAPTER 4:** Service Client Component
- **Appendix A:** Running Services on a User Workstation

© Copyright 2010 OpenSpan, Inc. All Rights Reserved

No part of this publication may be reproduced or distributed in any form or by any means, electronic or otherwise, now known or hereafter developed, including, but not limited to, the Internet, without explicit prior written consent from OpenSpan, Inc. Requests for permission to reproduce or distribute to individuals not employed by OpenSpan any part of, or all of, this publication should be mailed to:

OpenSpan, Inc.
4501 North Point Parkway
Suite 140
Alpharetta, Georgia 30022
www.openspan.com

OpenSpan[®] is a registered trademark of OpenSpan Inc., a Georgia Corporation.

SuperTrace[®] is a registered trademark of Green Hills Software, a Delaware Corporation.


Microsoft[®], Visual Studio[®], MSDN[®], and Windows[®] are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

CONTENTS

Chapter 1: Using Desktop Service Enablement	5
Prerequisites	5
Solutions Used in this Training Module	5
Background Information	6
Overview	7
Chapter 2: Creating a Service Enablement Project with Single Input and Result	9
Add a Web Adapter.....	9
Interrogate Google web application	9
Modify Match Rules	10
Add a Windows Form.....	11
Create Automation	11
Changing the Automation to Activate Service Enablement.....	12
Add Entry Point Value	12
Add Exit Point and Result.....	13
Creating and Configuring the Service Enablement Project Item	14
Add Service Project Item.....	14
Select Service Contract	15
Add and Configure Service Binding.....	15
Configure Service Port	17
Publish WSDL	18
Test Service	18
View WSDL.....	19
Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results.....	21
Create Solution	21
Create Automation	21
Add Service Project Item.....	23
Add and Configure Service Binding.....	23
Configure Service Contract	24
Test the Service.....	24
Chapter 4: Service Client Component.....	27
Creating a Service Client Component with Multiple Bindings	27
Appendix A: Running Services on a User Workstation	33
HTTP Binding.....	33
General Information	35

CONVENTIONS

You can save time using this training guide by understanding how screen elements, input data, and definitions are shown.

Convention	Meaning
Black bold characters	Names of program elements that require emphasis, such as command buttons, menus, and dialog boxes, are shown in black bold text.
Blue Bold Characters	Text that you are supposed to type or data selections, such as from drop-lists, appear in blue boldface characters.
<u>Remember</u>	<u>Definitions of terms and important concepts that bear remembering.</u>
	Next to the Tip icon, you can find best practices and shortcuts to use OpenSpan Studio more effectively.

CONTENTS

This page intentionally left blank.

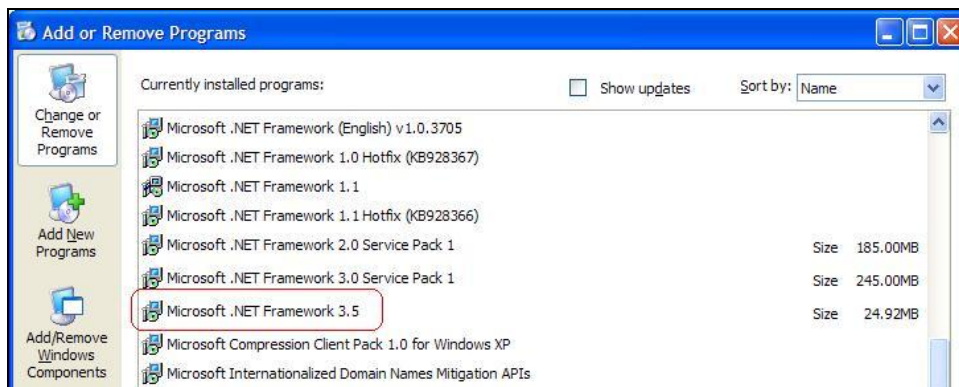
Chapter 1: Using Desktop Service Enablement

This guide describes how to use the Service Enablement feature available in the OpenSpan Studio Enterprise edition.

Prerequisites

This document assumes that you have a working knowledge of OpenSpan Studio and Web Service functionality.

The workstation running OpenSpan Studio Service Enablement must have Microsoft .Net Framework 3.5 installed. To confirm that .Net Framework 3.5 SP1 is installed, use the **Add or Remove Programs** Control Panel applet as show below:



If running a Service Enabled project on a workstation for which the user does not have Admin rights, you must prepare the workstation. Refer to **Appendix A: Running Services on a User Workstation** for details.

Solutions Used in this Training Module

Practice solutions you will use while working through the step-by-step exercises presented in this training guide can be [downloaded here](#). Finished solutions are available from the same download location as a reference for checking your work after completing the exercises on your own.

This document uses the following solutions:

- Service Enablement Example.zip
- Service Enablement Strings.zip

Background Information

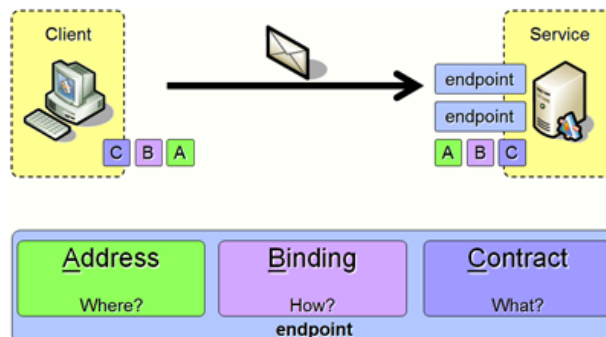
OpenSpan Studio is a development platform built on the Microsoft .NET Framework. OpenSpan's **Desktop Service Enablement** functionality uses the **Microsoft Windows Communication Foundation (WCF)** exposed in the .NET Framework **3.5 SP1**. WCF is implemented primarily as a **set of classes** on top of the **.NET Framework CLR**. OpenSpan Studio is integrated with WCF to enable developers of object-oriented applications to quickly build and publish service-oriented automations and integrations.



A **WCF Service** is a program that exposes a collection of **Endpoints**. Each Endpoint is a portal for communicating with the world. A **ServiceDescription** holds the collection of ServiceEndpoints each describing an Endpoint that the Service exposes. From this description, ServiceHost creates a runtime that contains an EndpointListener for each ServiceEndpoint in the ServiceDescription. The Endpoint's Address, Binding, and Contract correspond to the EndpointListener's listening address, message filtering and dispatch, and channel stack, respectively.

A **WCF Client** is a program that exchanges messages with one or more Endpoints. A **ChannelDescription** holds the one ServiceEndpoint with which the Client communicates. From this ChannelDescription, ChannelFactory creates the channel stack that can communicate with the Service's Endpoint.

A **ServiceEndpoint** has an **A****ddress**, a **B****inding**, and a **C****ontract**.



The Endpoint's **Address** specifies **where** the Endpoint resides as a **Network Address**. An EndpointAddress is basically a **URI**, an **Identity**, and a **collection of optional headers**. *In OpenSpan Studio, the Endpoint Address is specified by the ServiceAddress property.*

The Endpoint's **Binding** specifies **how** the Endpoint communicates with the world including things like **transport protocol** (e.g., TCP, HTTP), **encoding** (e.g., text, binary), and **security** requirements (e.g., SSL, SOAP message security). A Binding has a **Name**, a **Namespace**, and a collection of composable **Binding Elements**. The Binding's name and namespace uniquely identify it in the Service's metadata.

The Endpoint's **Contract** specifies **what** the Endpoint **communicates**, and is essentially a **collection of messages** organized in **operations** that have basic Message Exchange Patterns (MEPs) such as one-way, duplex, and request/reply. *The Contract relates to the automation in OpenSpan Studio solutions.*

Chapter 1: Using Desktop Service Enablement

Similar to Service Bindings, each Service Contract has a **Name** and a **Namespace** that uniquely identifies it in the Service's metadata.

For more information about WCF, please refer to the following MSDN articles:

- <http://msdn2.microsoft.com/en-us/library/ms731082.aspx>
- <http://msdn2.microsoft.com/en-us/library/ms731079.aspx>
- <http://msdn2.microsoft.com/en-us/library/ms733128.aspx>
- <http://msdn2.microsoft.com/en-us/library/aa480210.aspx>

Overview

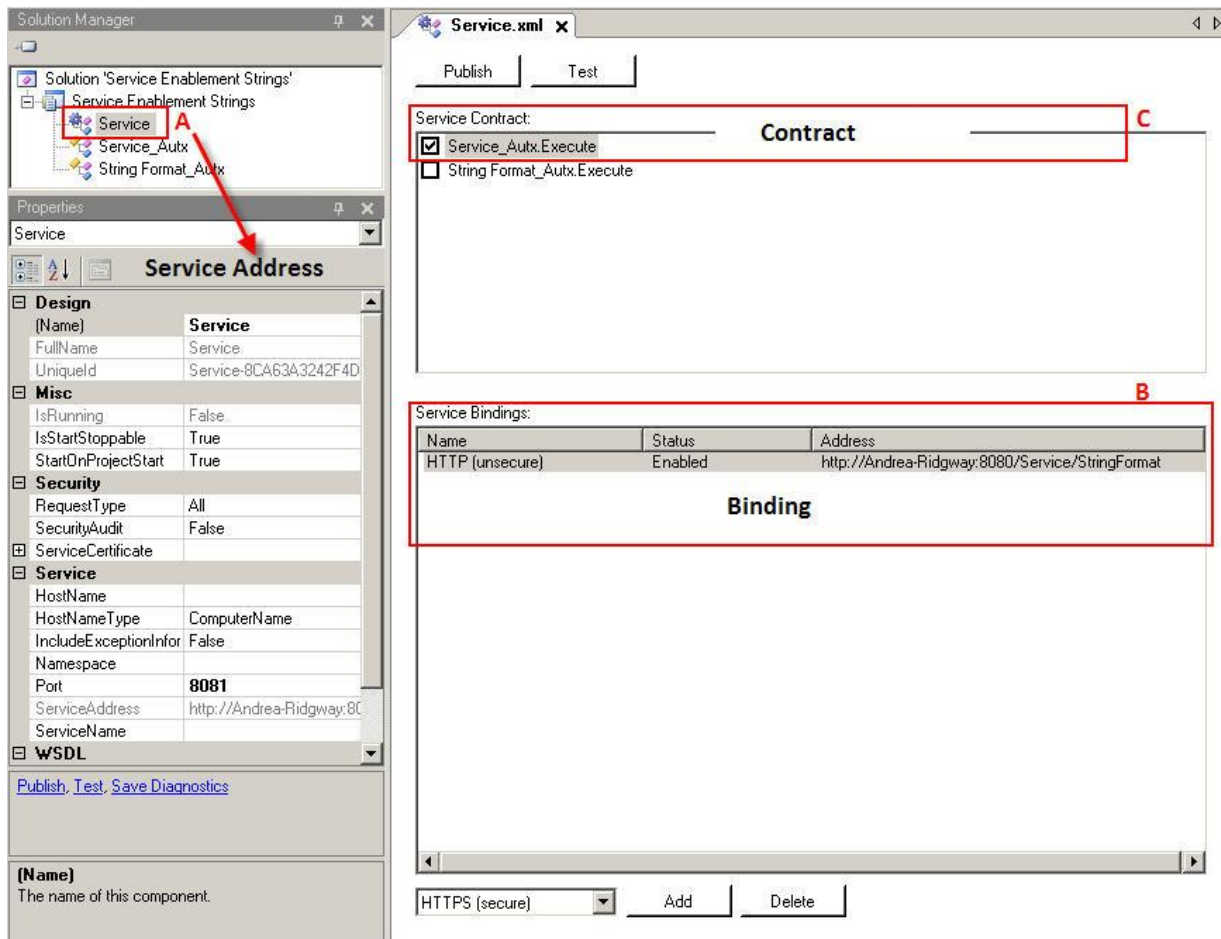
The steps to create and publish an automation as a Service are as follows:

1. Create an automation containing Entry and Exit points
2. Add the Service project item
3. Select the Service Contract
4. Add and Configure the Service Binding(s)
5. Configure the Service Port
6. Test the Service
7. Publish the WSDL (optional step)
8. Run solution containing service (either through OpenSpan Studio or OpenSpan Integrator)

You can consume the web service using another instance of OpenSpan Studio/Integrator and a solution containing the ServiceClient or WebService components, or any other web service client software. The sample projects detailed in the sections to follow shows how to implement these steps and then query the service from a web browser.

Chapter 1: Using Desktop Service Enablement

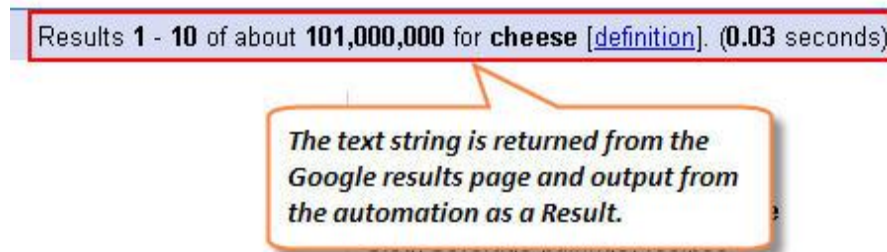
OpenSpan Studio provides support for the **WCF Service** via a Service project item with Contract, Binding and Service Address as shown in the following image:



Refer to the OpenSpan online Help Topic: **Service Enablement – Properties, Methods, and Events** for descriptions of the Service Address, Service Contract, and Binding properties.

Chapter 2: Creating a Service Enablement Project with Single Input and Result

In this sample project, an automation is created and then a web service is created from the automation. The automation receives input, a text string, and initiates a Google search on the string. Once the search is complete, the text string naming the number of results from the Google results page is returned as a result. See the following illustration:



The sample solution will be created in two parts. First, a WinForm will be used to test the functionality ensuring that the Google search initiates and returns the desired result string. Once the base functionality is confirmed, the solution will be modified so that the automation is service enabled. Execute Entry and Exit points will be added and a Service project item added to create a web service from the automation.

Add a Web Adapter

1. Create a new solution, with a Web Application. Set the URL to www.google.com. Rename the Web Application **Google Page**.

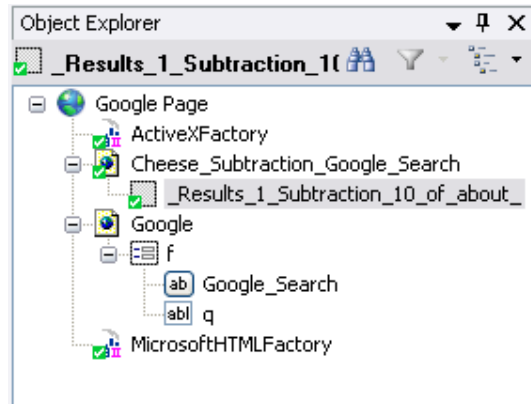
Interrogate Google web application

2. On the Google Home page, interrogate the data entry field where you enter search data and the **Google Search** button. Then enter a search, such as **Cheese** and click the **Google Search** button.
3. Once search results are returned, interrogate the results field.

Results 1 - 10 of about 18,223,916 for cheese.

Chapter 2: Creating a Service Enablement Project with Single Input and Result

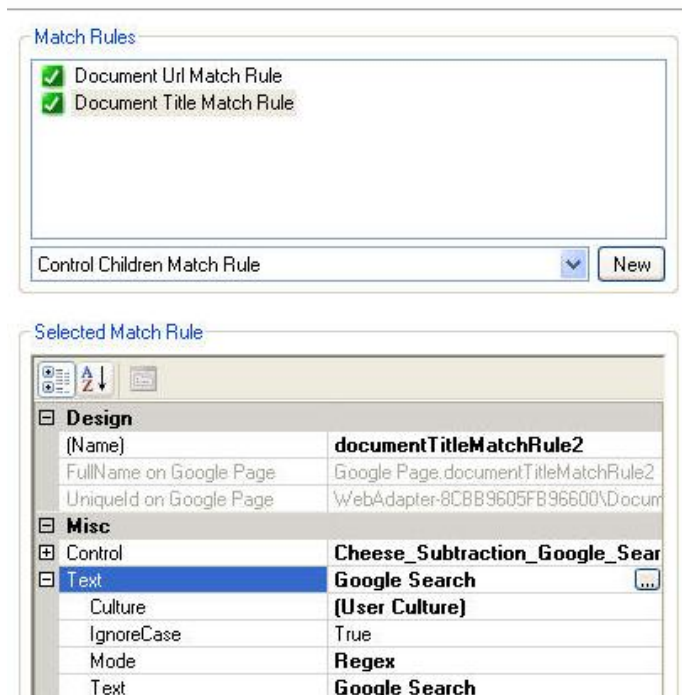
Your Object Explorer should look similar to the following:



Modify Match Rules

Since OpenSpan matches on the exact text of the search item, you will need to modify the match rules for the results page to broaden the search.

4. For the **cheese_Subtraction_Google_Search** web page object, select the **Document Title** match rule and change the **Mode** to **Regex**. Edit the **Text** for the rule to **Google Search**. Your Match Rules Pane should look like the following:



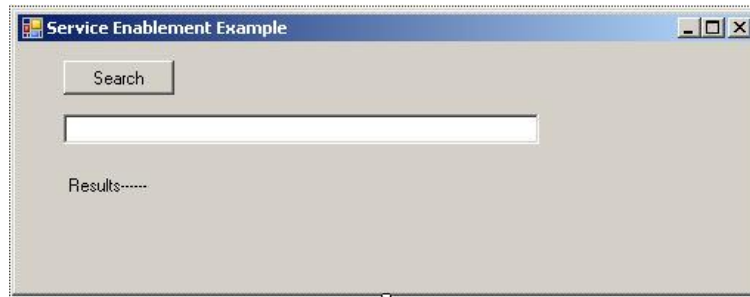
5. Stop the interrogator and save the solution.

Chapter 2: Creating a Service Enablement Project with Single Input and Result

Add a Windows Form

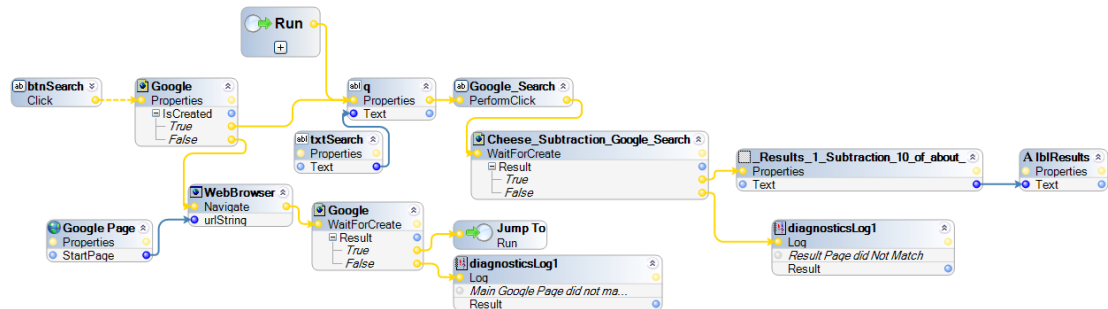
- In order to **test** pushing and retrieving data from the web application integration, we will use a simple Windows Form. Note that this form will not be used by the final services solution. Continue by adding a Windows Form to your project.
- Add a Button, Textbox, and Label to the Windows Form.

Your Form should look similar to the following:



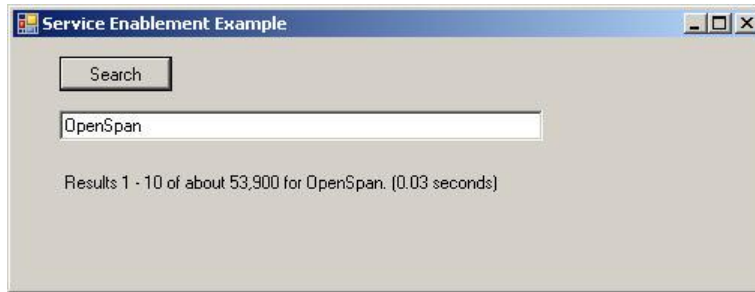
Create Automation

8. Add a new automation that sets the logic so that when the Windows Form Search button is pressed text is sent from the Windows Form text box to the Google text box and the **Google Search** button is automatically clicked. An example of the completed automation is shown below:



Chapter 2: Creating a Service Enablement Project with Single Input and Result

9. Save and run the solution. Test the solution with various entries, an example follows:

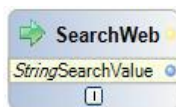


Changing the Automation to Activate Service Enablement

The base solution performs the intended task – to perform a Google search and return the “Result” text. The next step is to modify the solution for Service Enablement. For Service Enablement there are no user interactions. The first step in modifying the solution is changing how data are input and output from the search. Instead of providing the data through a WinForm, the automation will be supplied data when it is called as a service. To make this change, we will add Entry and Exit points for calling the automation/service and passing data.

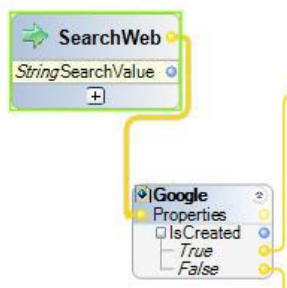
Add Entry Point Value

10. Add an Entry Point to the automation by right-clicking on the automation surface and selecting Add Entry Point from the context menu. Rename the entry block to **SearchWeb**. Enter the String parameter **SearchValue**.



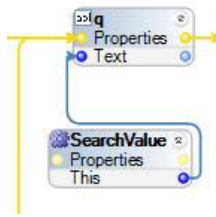
Note: When designating Input parameters for service automations, use simple types such as String, Boolean, Integer, and Float. Complex types are not supported.

11. Disconnect the **Search** Button **Click** event from the **Web Browser IsCreated** property and replace the link with the **SearchWeb Execute** event link.



Chapter 2: Creating a Service Enablement Project with Single Input and Result

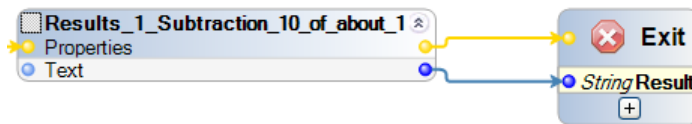
12. Disconnect the **WinForm Textbox Text** property from the Google **q** textbox **Text** property and replace the input with the **This** property for the **Search Value** parameter.



Note that the **This** property contains the current value of the **Search Value** parameter.

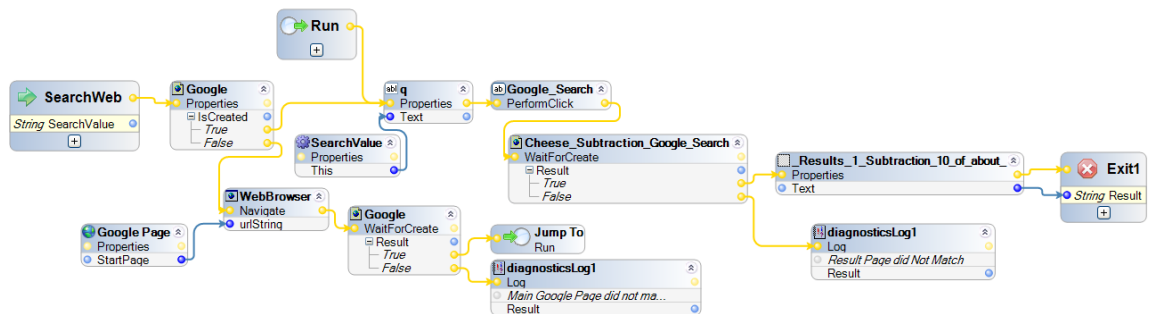
Add Exit Point and Result

13. Add an Exit point (the result returned by the service). Pass the execution link from the **Result** object to the **Exit** point as shown in the following illustration:



Note: When designating Output parameters and Results for service automations, use simple types such as String, Boolean, Integer, and Float. Complex types are not supported. You may have multiple output parameters. (Refer to the [Creating a Service Enablement Project with Multiple Inputs and Results](#) section.)

The completed automation looks like the following:



Note: When creating a service, do not interrupt execution flow waiting for user interaction as user interaction will not occur during the running of the service.

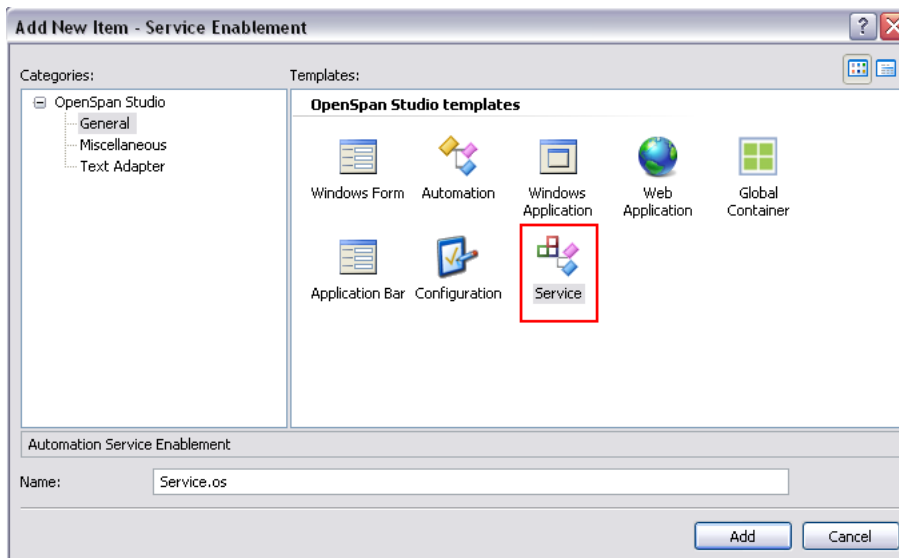
14. Save the solution.

Creating and Configuring the Service Enablement Project Item

Now that you have correctly created the automation, you are ready to add the Service project item. This project item will be defined based on the automation entry point you created, along with binding and port information. Continue the example solution as follows:

Add Service Project Item

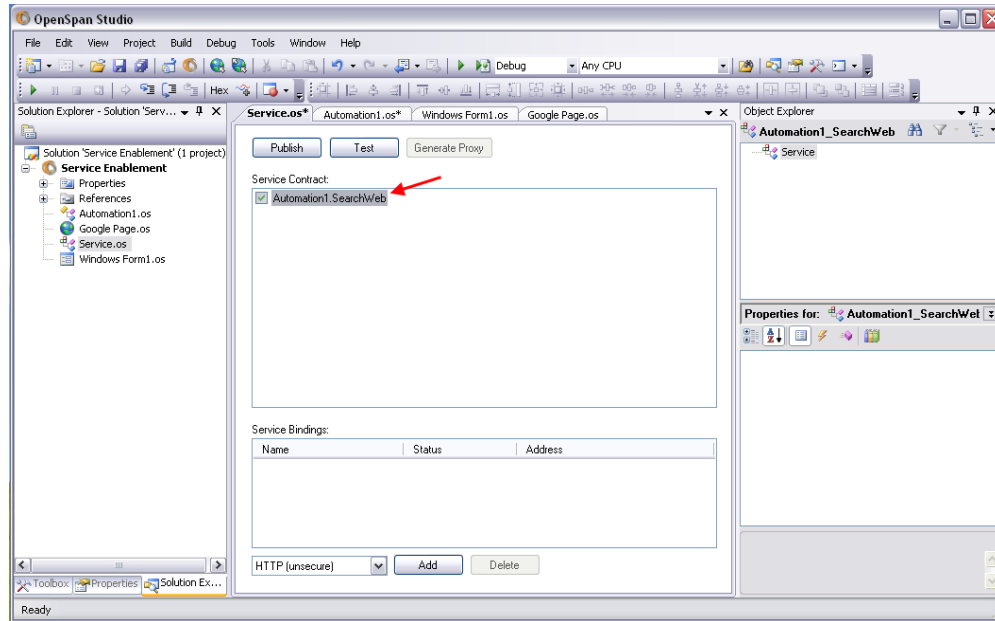
15. Right-click on the project in the **Solution Explorer** and select **Add | New Item** from the context menu.
16. From the **Add New Item** window select **Service** from the templates and click Add.



Chapter 2: Creating a Service Enablement Project with Single Input and Result

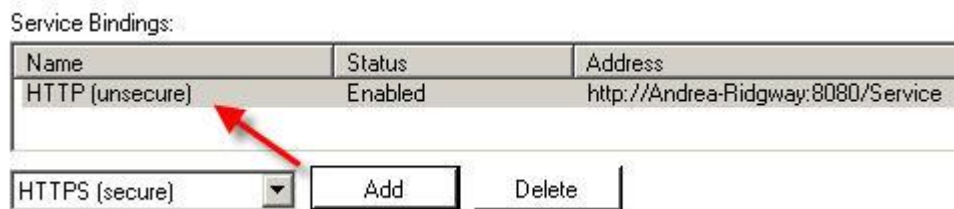
Select Service Contract

17. In the **Service Contract** screen section, a list of available automations displays along with their entry points. Since this project has only one automation entry point, a single service contract is listed. Select this automation as the **Service Contract** by checking the box:



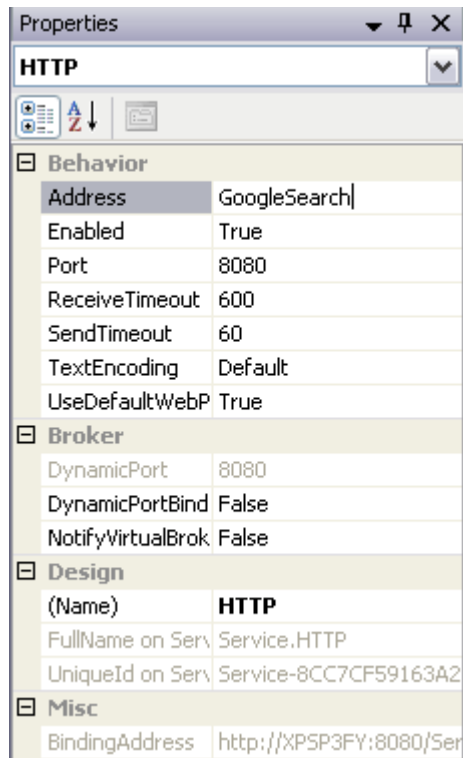
Add and Configure Service Binding

18. Click the **Add** button to add a service binding. In this case, we will use HTTP (unsecure), the default.



Chapter 2: Creating a Service Enablement Project with Single Input and Result

19. Highlight the Service Binding and set Properties for the binding. Enter an address. This can be virtually anything but for our purposes, enter the address [GoogleSearch](#) and a port. The port can be in the range of 1 to 65535.



The screenshot shows the 'Properties' window for an 'HTTP' service binding. The window is divided into several sections: Behavior, Broker, Design, and Misc. The 'Behavior' section contains properties like Address, Enabled, Port, ReceiveTimeout, SendTimeout, TextEncoding, and UseDefaultWebP. The 'Broker' section contains DynamicPort, DynamicPortBind, and NotifyVirtualBrok. The 'Design' section contains (Name), FullName on Serv, and UniqueId on Serv. The 'Misc' section contains BindingAddress.

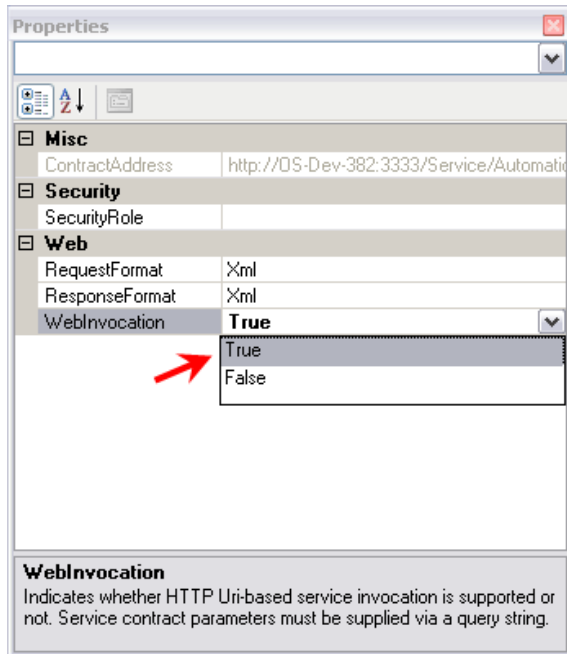
HTTP	
Behavior	
Address	GoogleSearch
Enabled	True
Port	8080
ReceiveTimeout	600
SendTimeout	60
TextEncoding	Default
UseDefaultWebP	True
Broker	
DynamicPort	8080
DynamicPortBind	False
NotifyVirtualBrok	False
Design	
(Name)	HTTP
FullName on Serv	Service.HTTP
UniqueId on Serv	Service-8CC7CF59163A2
Misc	
BindingAddress	http://XPSP3FY:8080/Ser

Chapter 2: Creating a Service Enablement Project with Single Input and Result

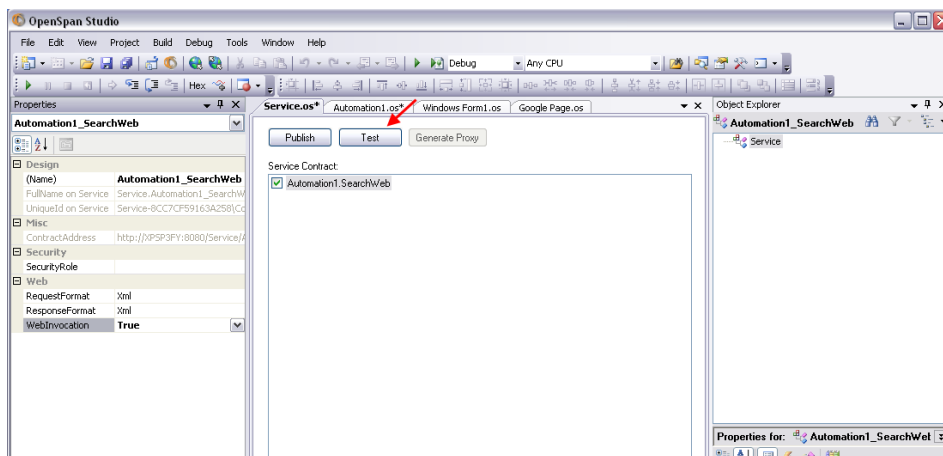
Configure Service Port

20. In the **Service.os** pane, select the Service Contract. In the **Properties** window for the service contract, set the **WebInvocation** property to **True**. This setting will allow us to call the web service from a browser for testing. Save your solution.

Sample properties for the Service Contract follow:

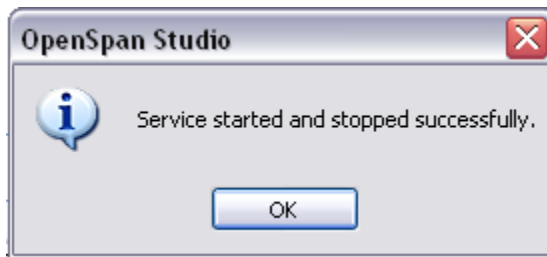


21. Click the **Test** button to ensure that all of the settings are correct and that the port is free.



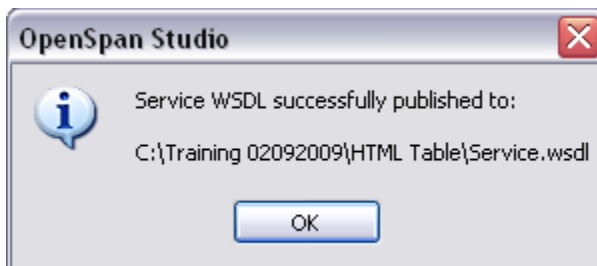
Chapter 2: Creating a Service Enablement Project with Single Input and Result

If the settings are applied successfully and the port was accessible, you should receive the following message:



Publish WSDL

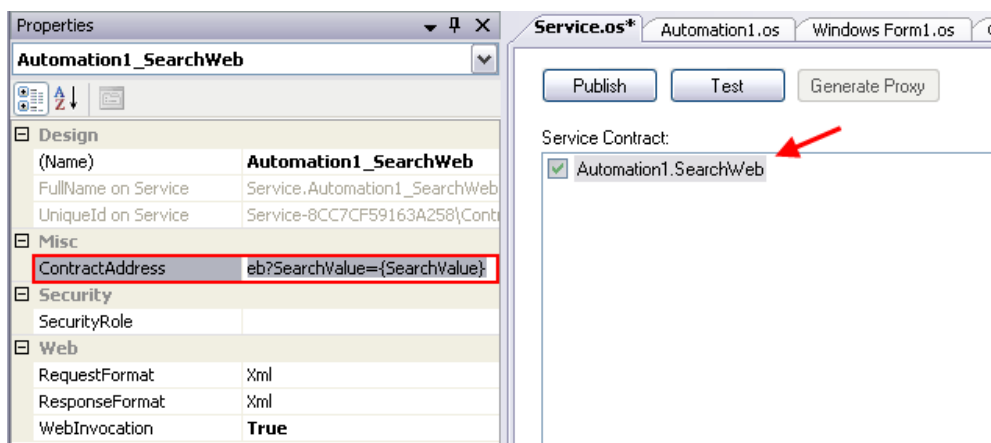
22. If you are providing the WSDL to a client, click the **Publish** button and navigate to a location to save the WSDL. Once you click **Save**, you will receive a message indicating that the WSDL was successfully created:



Note: The WSDL is available directly from the service when it is running, as per the ServiceWSDLAddress property which is the serviceaddress plus ?wsdl. For example <http://myhost:5420/example?wsdl>. See [View WSDL](#) below.

Test Service

23. With the **Service Contract** still selected, copy the **ContractAddress** property. This will be the URL used to test the service.



24. Start the solution.

Chapter 2: Creating a Service Enablement Project with Single Input and Result

25. To access the REST service invocation URL, paste the **ContractAddress** address into a Web Browser Address Bar. **The Web Browser must be a different instance than was launched as part of the solution.**
26. Change the {SearchValue} to something else for a desired Google search. For example, to search for OpenSpan, you would change the SearchValue as follows:

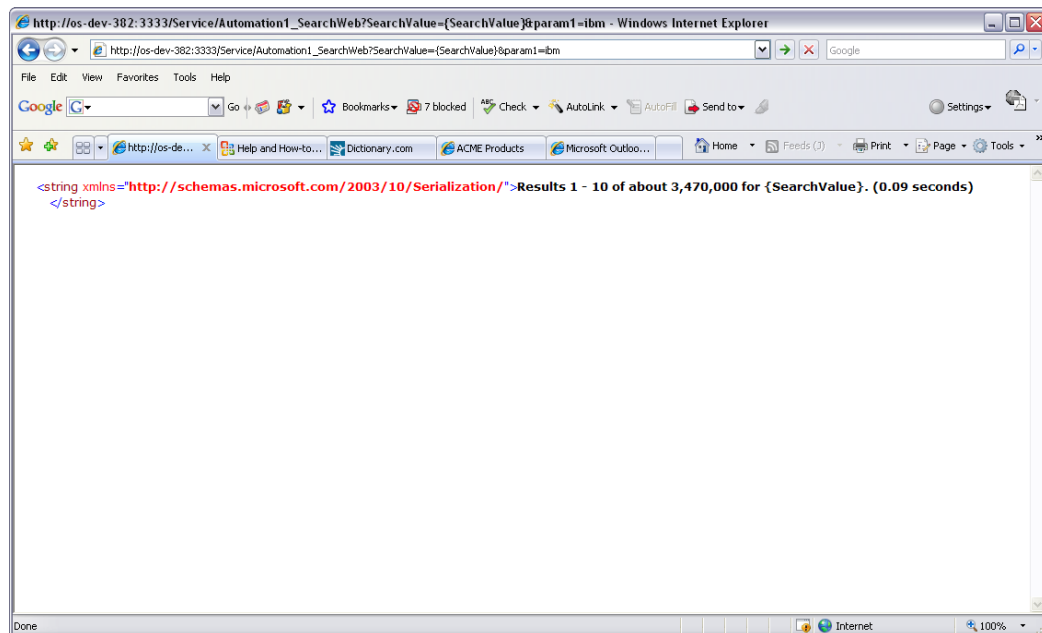
From:

http://OS-Dev-382:3333/Service/Automation1_SearchWeb?SearchValue={SearchValue}

To:

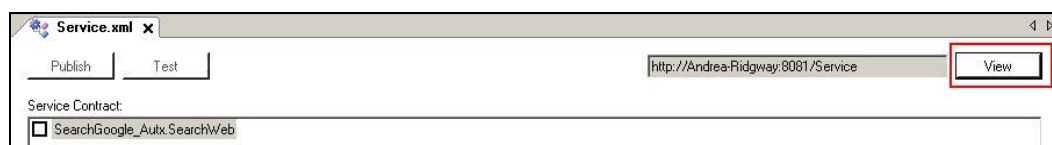
http://OS-Dev-382:3333/Service/Automation1_SearchWeb?SearchValue=OpenSpan

An example result screen follows:



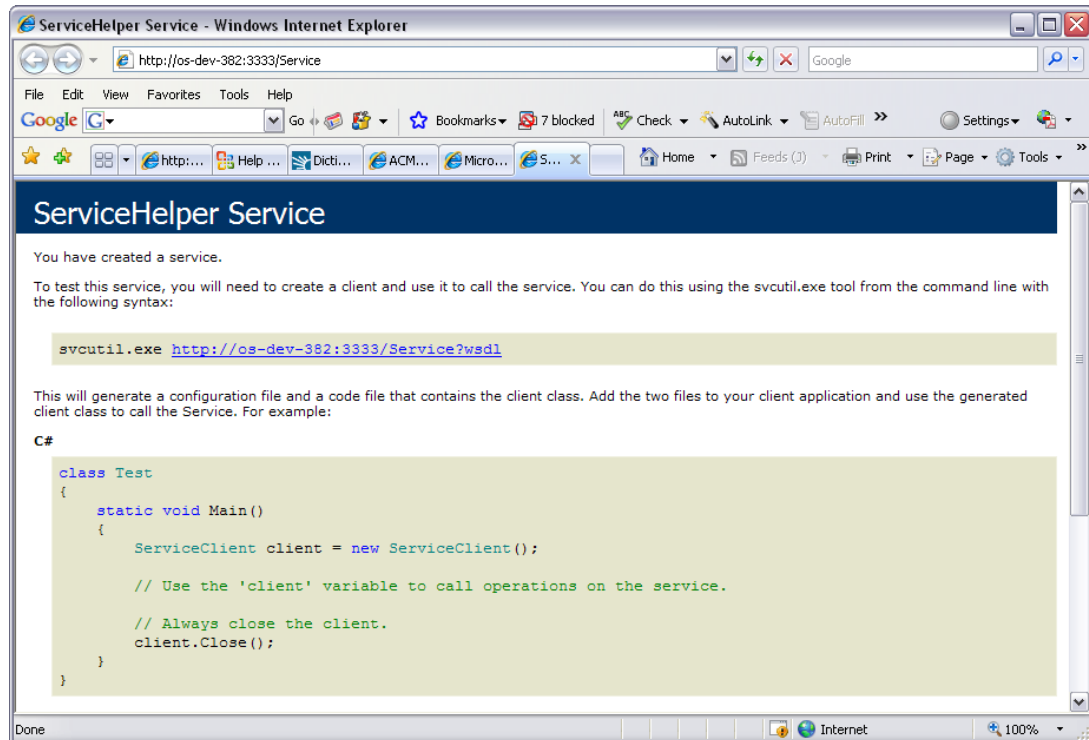
View WSDL

27. To view the WSDL file created by your solution, click the **View** button on the Service.os pane while the solution is running:

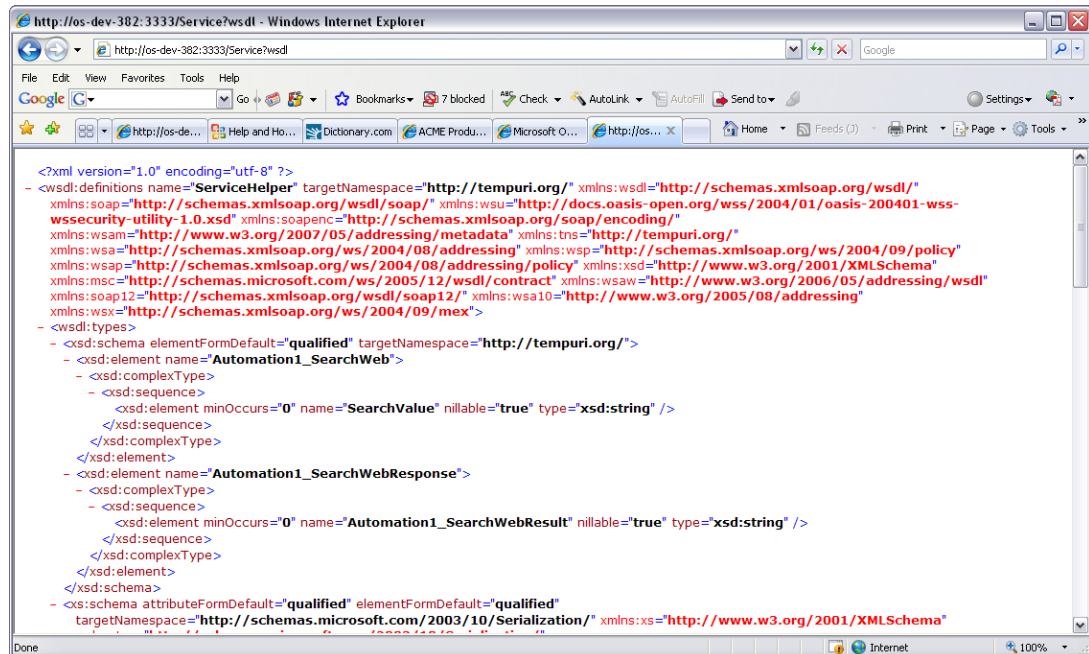


Chapter 2: Creating a Service Enablement Project with Single Input and Result

Once you click the view button, a browser opens displaying general information about how to use the WSDL:



28. Click on the WSDL link shown on the page to open the WSDL. An example of the WSDL for the sample solution follows:



Note: The View function is only available while the solution is running.

Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results

When using Service Enablement you are not limited to solutions with single inputs, results, or single automations. In this next sample project, we will create a simple solution that uses two string inputs, calls another automation to format the strings, and then outputs the two formatted strings. This solution shows you how you can:

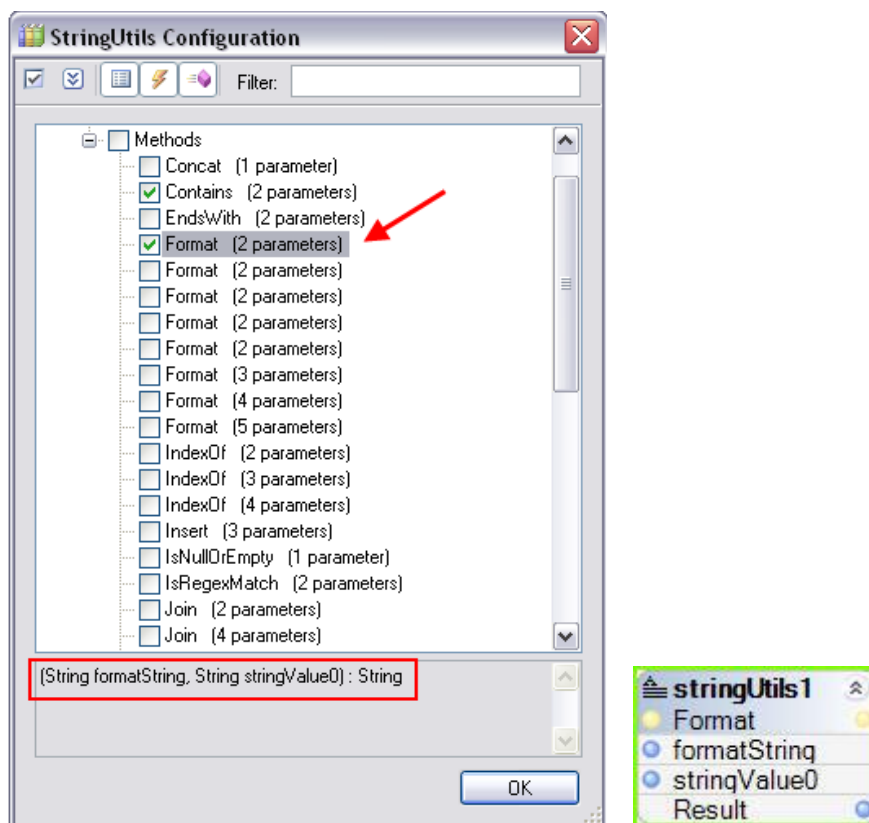
- use multiple input points
- call another automation from the service automation
- yield multiple results in your service automation

Create Solution

1. Create a solution named **Service Enablement Strings**.

Create Automation

2. Add a new automation named **String Format_Autx**, which formats text using the **StringUtils** component. Use the Format method with 2 parameters.



Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results

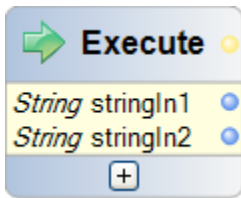
- For the **formatString** parameter, enter the following:

This is your first string: "{0}" Formatted.

- Add a second **StringUtils** component and set the **Format** method similarly, with the **formatString** value:

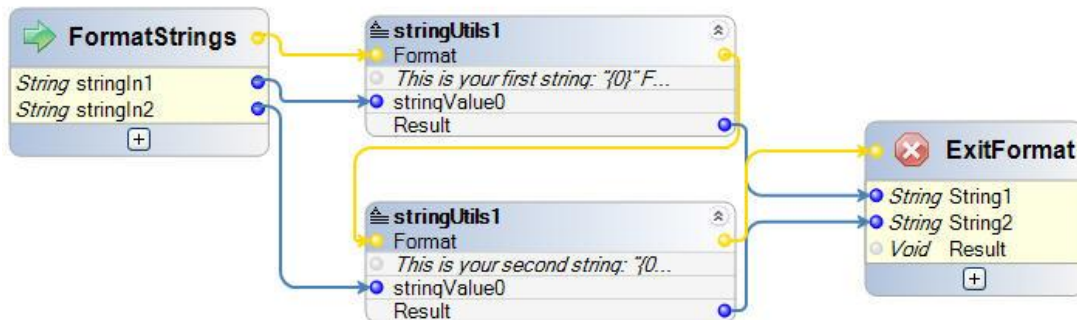
This is your second string: "{0}" Formatted.

- Add an **Entry** point to the automation and add two String parameters: **stringIn1**, **stringIn2**.



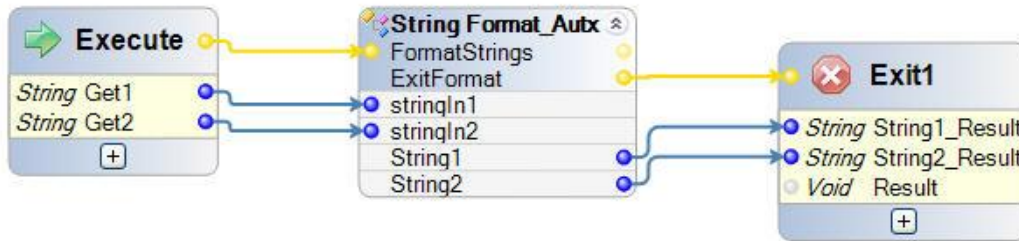
- Add an Exit point to the automation and add two string parameters. Name the parameters String1 and String 2.

Your automation should look like the following:



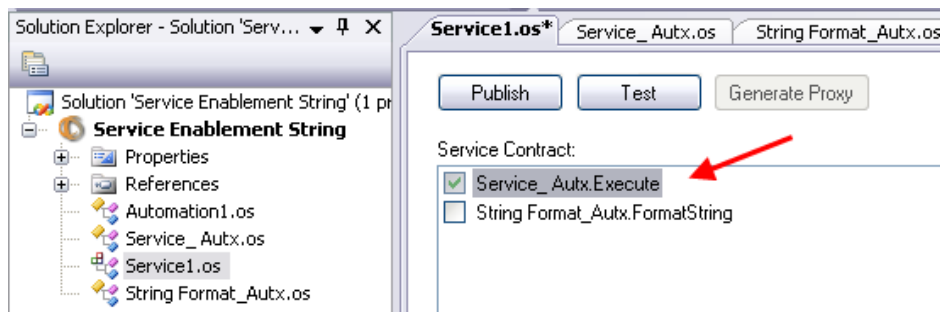
- Add a new automation named **Service_Autx**. Add an entry point with two String parameters and an Exit point with two String Parameters **Get1** and **Get2**.
- Name the Exit Point Parameters **String1_Result** and **String2_Result**.
- Add the **Execute** method for the **StringFormat_Autx** automation. Your complete automation should look like the following:

Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results



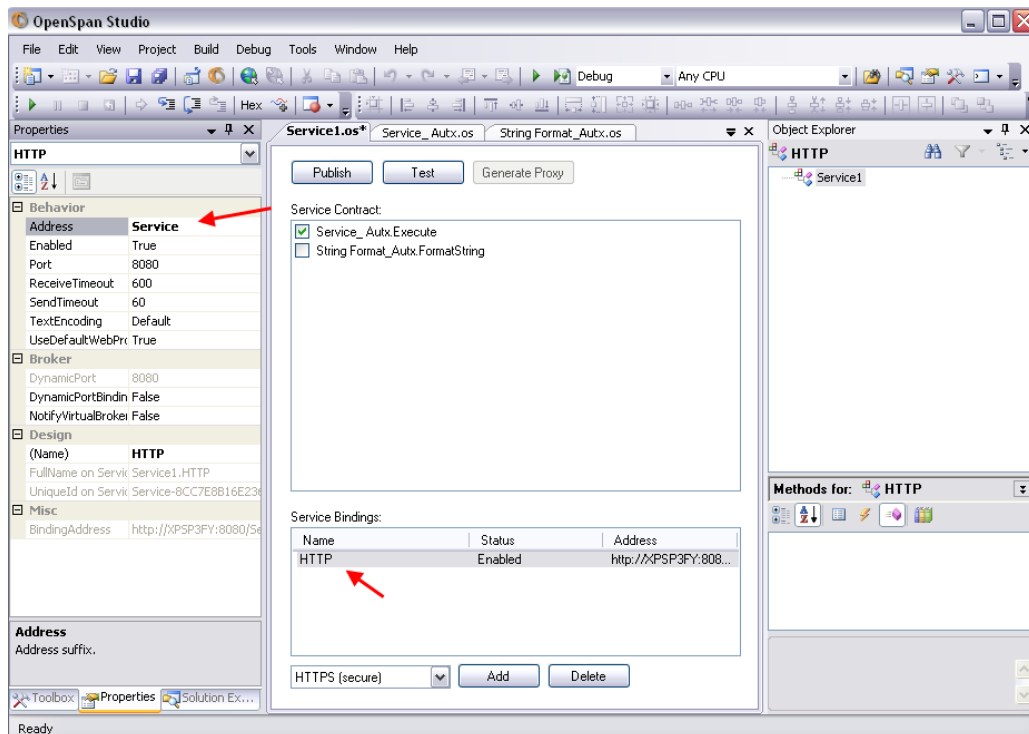
Add Service Project Item

10. Add a Service to the solution and define the **Service Contract** for the **Service_Autx** automation.



Add and Configure Service Binding

11. Add the **HTTP (Unsecured)** Service Binding for the service and enter **Service** for the **Address** property.

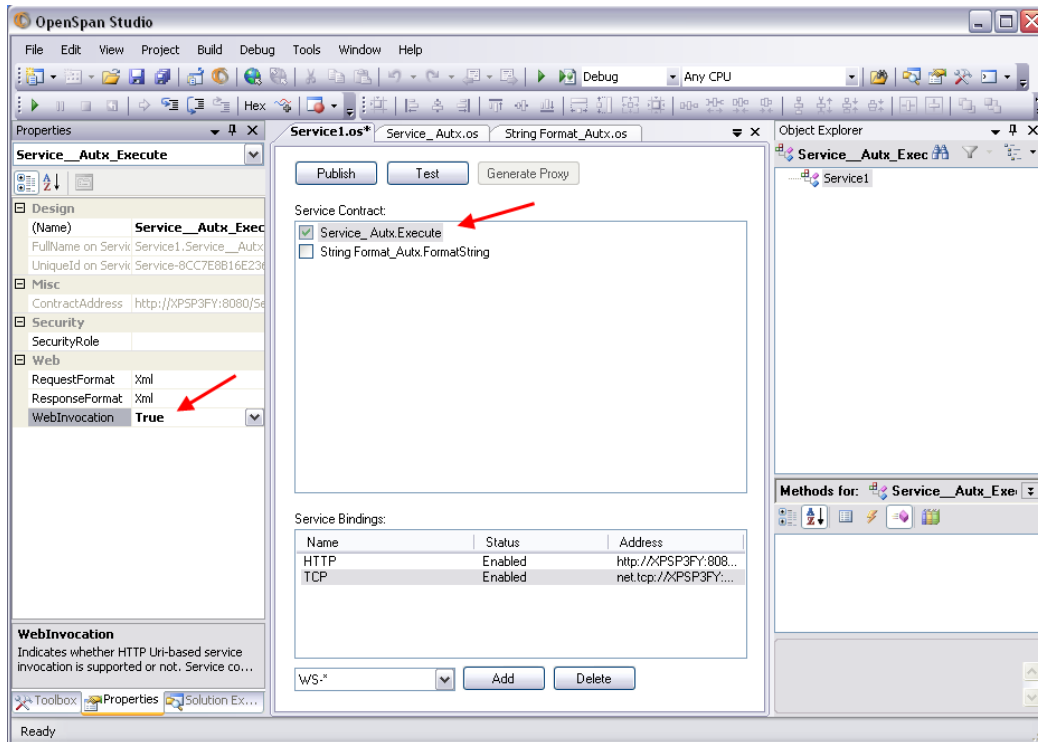


Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results

12. Add another binding (which will be used in another exercise) **TCP** Service Binding and enter **StringsTCP** for the **Address** property.

Configure Service Contract

13. Select the **Service_Autx.Execute** Service Contract. In the **Properties** window for the service contract, set the **WebInvocation** property to **True**. Save your solution.



Test the Service

14. Test the service. Once you receive notice that the test is successful, publish the service.
15. Copy the **ContractAddress** for the **Service_Autx.Execute** Service Contract from the **Properties** window. Yours should be similar to the following:

http://ServerName:Port/Service1/Service_Autx_Execute?Get1={Get1}&Get2={Get2}

16. Run the solution. Open an internet browser and paste the **ContractAddress** into the address bar.
17. Replace the parameter placeholders {Get1} and {Get2} with text of your choice. For example, Openspan and Atlanta would be entered as follows:

Chapter 3: Creating a Service Enablement Project with Multiple Inputs and Results

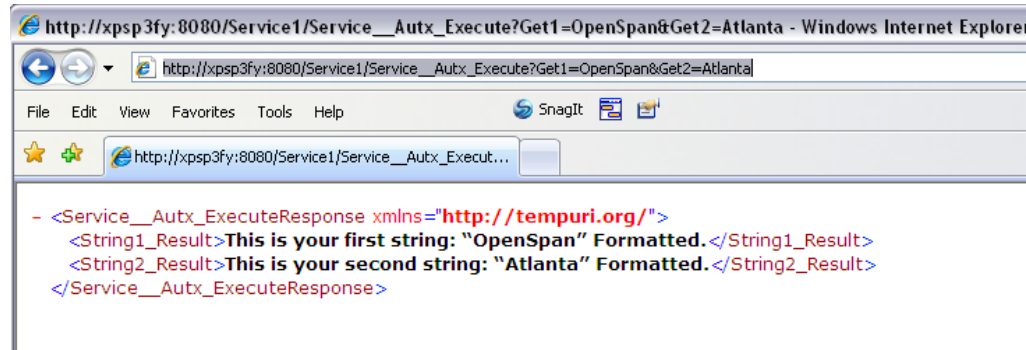
Before Replacing Parameters:

http://OS-Dev-382:8081/Service/Service_Autx_Execute?param1={param1}¶m2={param2}

After Replacing Parameters:

http://xpsp3fy:8080/Service1/Service_Autx_Execute?Get1=OpenSpan&Get2=Atlanta

18. Select the Browser **Go To** or **Refresh** command to invoke the Web Service. An example of the results returned by the service is shown below:



This page intentionally left blank.

Chapter 4: Service Client Component

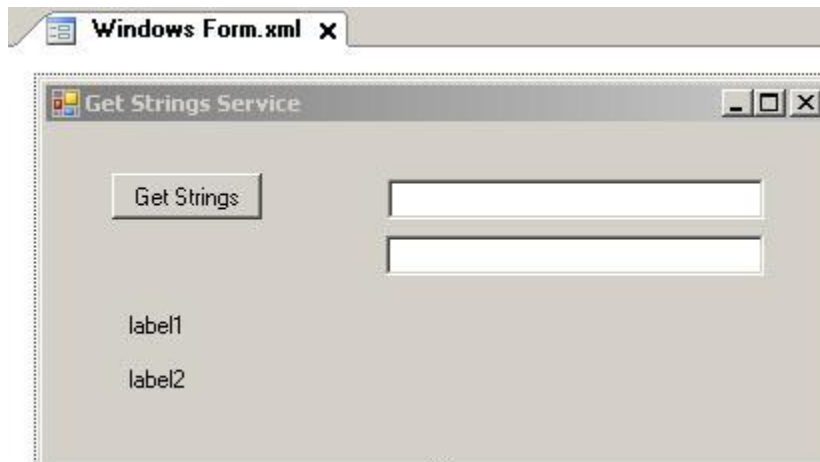
OpenSpan Studio offers a component, Service Client, for consuming web services. This component has the enhanced ability that it supports the same set of protocols available on the server side of our SOA offering: NP, TCP, HTTP, HTTPS and WS*. ServiceClient adds support for WS* based, WCF based and SOAP style web services support.

Refer to the OpenSpan online Help Topic: [ServiceClient](#) for definitions of the component's properties, methods, and events.

Creating a Service Client Component with Multiple Bindings

In this sample project, a service created in OpenSpan Studio is used. The FormatStrings solution created earlier will be the service accessed by this Service Client solution. The solution will access the service and display the results in labels on a Windows Form. While creating the solution we will work with the Bindings property to see how multiple bindings are handled by the Service Client.

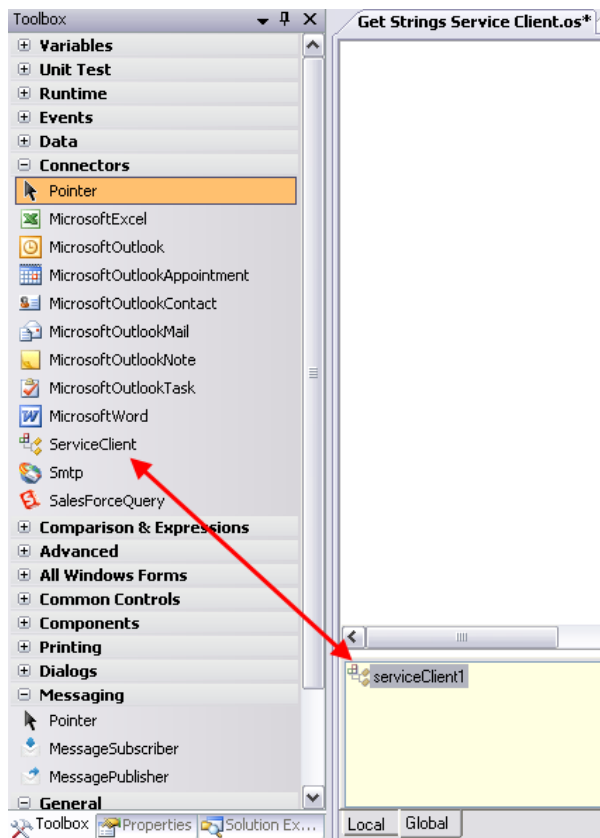
1. Begin by opening a second instance of OpenSpan Studio.
2. In the new instance of OpenSpan Studio, create a new solution: [GetStringsService](#).
3. Add a Windows Form to the solution.
4. Add a button, two textboxes and two labels to the form. Your Windows Form should look like the following:



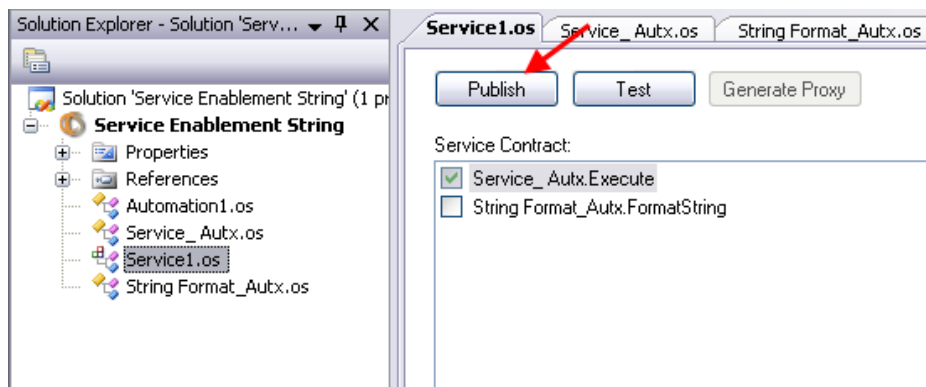
5. Save the solution.
6. Add an automation [Get Strings Service Client](#).

Chapter 4: Service Client Component

7. Add the **ServiceClient** component to the automation. Note, if the component is not in your Connectors tab in the toolbox, you can add it from the **OpenSpan.Service.dll**.

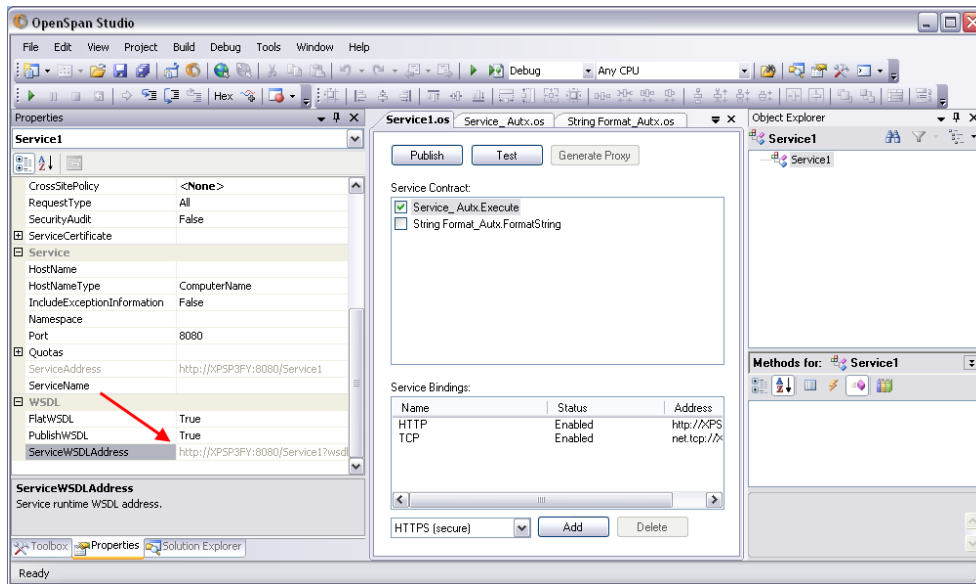


8. Switch to the first instance of OpenSpan Studio and open the **Service Enablement Strings** solution (if it is not already open).
9. Open the Service project item and click the **Publish** button to publish the WSDL to a location on your computer.

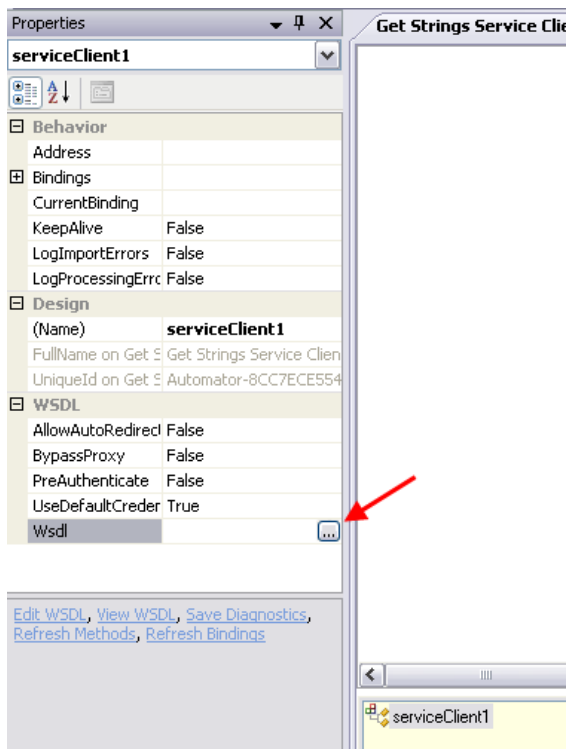


10. Copy the contents of the **ServiceWSDLAddress** property for the service to Notepad.exe/Clipboard.

Chapter 4: Service Client Component

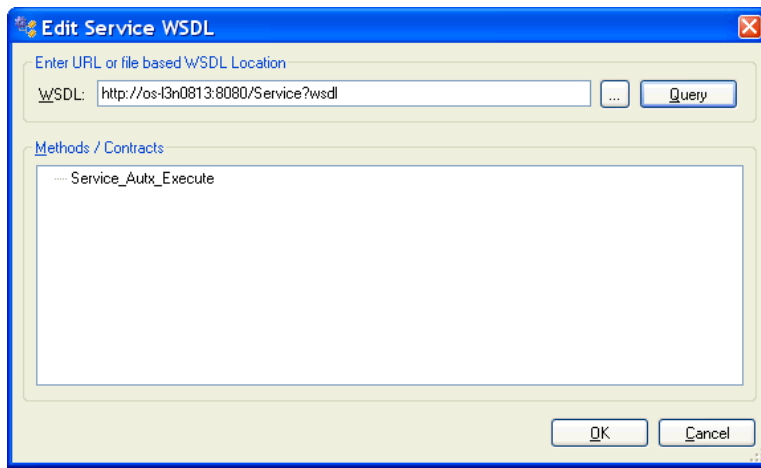


11. Run the **Service Enablement Strings** solution in the *first* instance of OpenSpan Studio.
12. Return to the *second* instance of OpenSpan Studio.
13. For the ServiceClient component browse the **WSDL** property to open the **Edit Service WSDL** dialog.



Chapter 4: Service Client Component

14. Paste the contents of the Clipboard (or Notepad) which is the WSDL for your service into the **WSDL** field and then click **Query**. After a few moments, **Service_Autx_Execute** should appear under methods/contracts.



15. Click **OK** to continue.
16. Highlight the Service Client in the **Object Explorer**. The **Service_Autx_Execute** method should appear as shown below:



17. Drag the **Service_Autx_Execute** method to the **Get Strings Service Client** automation. The design block shows connection points for the input parameters and results:



Note: The **String1_Result** is automatically converted to the **Result** since no **Result Exit** point was provided by the service.

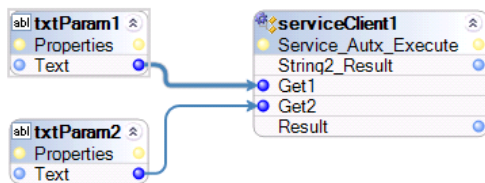
Chapter 4: Service Client Component

18. Note the values in the **Bindings** property:

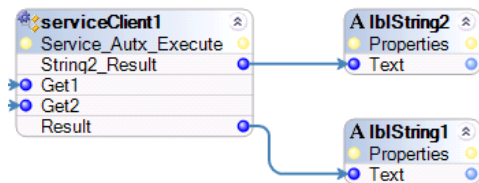
Bindings	HTTP, TCP
CurrentBinding	HTTP

Both bindings are listed. The first binding listed for the Format Strings service is the Current Binding. You can change this by clicking the drop-down button and choosing the other binding.

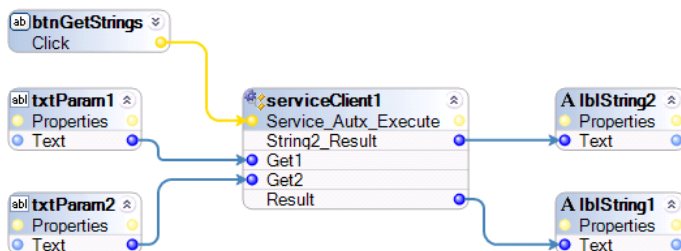
19. Use the **Textboxes.Text** properties from the Windows Form as inputs to the Service Client.



20. Use the **Labels.Text** properties from the Windows Form to receive the results from the Service Client.

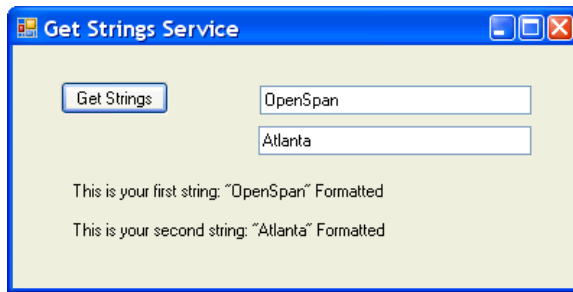


21. Connect the **Button.Click** event to the **ServiceClient.Service_Autx_Execute** event to trigger the method. Your completed automation should look like the following:



22. With the service running in the first instance of OpenSpan Studio, run your Service Client solution in the second instance OpenSpan Studio. Enter text in the textboxes and click the button to send the data to the service and format the text. The results should be returned to the labels as follows:

Chapter 4: Service Client Component



23. Stop the solutions in both instances of OpenSpan Studio.

*This simple set of solutions shows the basics of communicating between a service enabled automation and a solution using a Service Client component. For more examples using the Service Client component, refer to elective training course entitled **Working with the OpenSpan Service Client Component**.*

Appendix A: Running Services on a User Workstation

If the workstation on which you are running the service project has a logged in user without Admin rights to the machine, then you will need to prepare the workstation to run a service. The steps needed to prepare the workstation depend on the binding type for the service. Use the steps which follow to prepare the workstation.

HTTP Binding

1. Download the Windows XP Service Pack 2 Support Tools from the following web site:
 - <http://www.microsoft.com/downloads/details.aspx?FamilyID=49ae8576-9bb9-4126-9761-ba8011fabf38&displaylang=en&Hash=LNg70HERU6vSgWxSajaTkqxZ1iPHsgHv%2bQIKM9MPe2mAlq9CPgPsw%2bY%2faVY3xwQNtQS4MwYyFSqssbRoqDR6Pg%3d%3d>
2. When installing the Support Tools, make sure to select the Complete installation option (instead of the Typical option) as this installs the Optional Tools including the HTTP Configuration Utility (httpcfg.exe).
3. While logged into the workstation with Admin rights, select:
 - **Start | Program | Windows Support Tools | Command Prompt**
4. In the Command window, type the following command:
 - `Httpcfg Set urlacl -u http://+:port/ -a D:(A;;GA;;;WD)`
where:
port = The Port for the Service
-a D:(A;;GA;;;WD) = The ACL SDDL encrypted Security String

Appendix A: Running Services on a User Workstation

For Example, for the Service with properties shown below, the command string is:

- `Httpcfg Set urlacl -u http://+:8080/ -a D:(A;;GA;;;WD)`

Service	
Design	
(Name)	Service
FullName	Service
Uniqueld	Service-8CAB0B36AC31882
Misc	
IsRunning	False
IsStartStoppable	True
StartOnProjectStart	True
Security	
RequestType	All
SecurityAudit	False
ServiceCertificate	
Service	
HostName	
HostNameType	ComputerName
IncludeExceptionInformation	False
Port	8080
Quotas	
ServiceAddress	http://DS-L3N0813:8080/Service
ServiceName	
WSDL	
FlatWSDL	True
PublishWSDL	True
ServiceWSDLAddress	http://DS-L3N0813:8080/Service?wsdl

5. Logoff the Admin user and log into the workstation as a non-Admin user. Run the project containing the Service in OpenSpan Studio or OpenSpan Integrator.
6. If you want to remove the URLACL, type the following in the Command window:
 - `httpcfg delete urlacl -u http://+:port/`

Appendix A: Running Services on a User Workstation

General Information

httpcfg set urlacl

/u {http://URL:Port/ | <https://URL:Port/>}

The **/u** parameter takes a string containing a fully qualified URL that will serve as the record key for the reservation being made. When using **set urlacl**, the **/u** parameter is required.

/aACL

The **/a** parameter takes a string containing an Access Control List in the form of a Security Descriptor Definition Language (SDDL) string. For more information see [Httpcfg Remarks](#). When using **set urlacl**, the **/a** parameter is required.

Refer to the following for more information:

<http://blogs.msdn.com/drnick/archive/2006/04/14/configuring-http.aspx>

<http://technet.microsoft.com/en-us/library/cc781601.aspx>