Getting Started with the Bullhorn SOAP API and C#/.NET

Introduction

This tutorial is for developers who develop custom applications that use the Bullhorn SOAP API and C#. You develop a sample application that creates a session to the Bullhorn system, retrieves candidates by using several methods, and displays the results in a table on the web page.

You learn how to do the following tasks:

- Setup your environment for development.
- Access the Bullhorn SOAP API.
- Create and publish a web application in Visual Studio that retrieves candidate data.

You can follow the step-by-step instructions and download and run the code sample files.

Prerequisites

- Knowledge of C#, the ASP.NET framework, and HTML
- Visual Studio is installed on your system, and familiarity with using it

Code files

The zip file includes the solutions for this tutorial. To skip following the steps, download and run the solution file to see the completed application. Before running the file, refer to the Getting Started section to add the authentication information and your API key to the solution file.

1. Unzip the zip file under the Projects directory in your Visual Studio workspace.
2. In Visual Studio, select File > Open Project.
3. Navigate to the GettingStartedWithWebServices_Solution folder in the Projects directory.
4. Double click on the GettingStartedWithWebServices_Solution.sln file to open the project.
5. Modify the code in Default.aspx.cs file to have your username, password and API key.
6. Build and run the code to see the complete application.

Getting started

To develop applications with the Bullhorn SOAP API, you require a **username**, **password** and **API key**. If you don't want to use your own, contact Bullhorn support to provide an API user account.

Note: Developers working directly for Bullhorn customers can get API access by contacting Bullhorn Support. After the APIs are enabled for a Bullhorn client, a client administrator can generate a customer-specific API key by going to Tools > BH Connect > Web Services API.If a key does not already exist, click **Add Account** to create a key.

Setting up Visual Studio

The following steps explain how to create and set up a project in Visual Studio that connects to the Bullhorn WSDL.

Note: in this tutorial, you create a web application project. Follow the same steps to create a web project, service project, or other type of Visual Studio project.

Creating a project

In this section, you create a project in Visual Studio using an existing web application template.

1. Launch Visual Studio and select File > New Project.
2. Select Project types under Visual C#.

3. Select ASP.NET Web Application as a template.
4. Name the project GettingStartedWithWebServices and click OK.

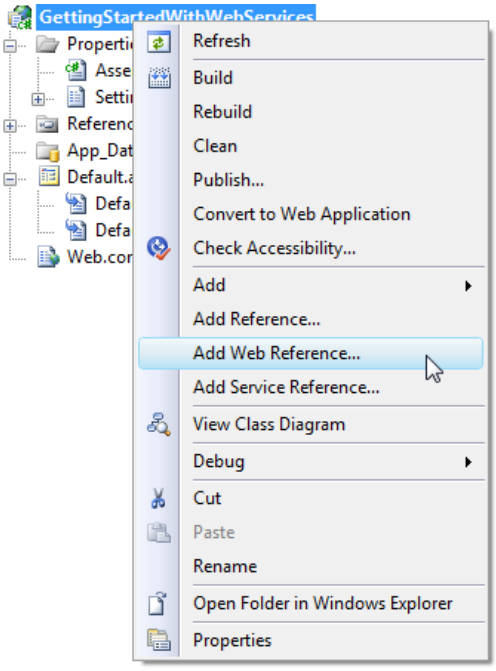
The Default.aspx file opens in the editor and the Solution Explorer lists all the files in the project.

Adding a web reference to the Bullhorn WSDL file

Before using the API, you must generate C# objects and classes from the Bullhorn's WSDL file, which serve as proxies for their server-side counterparts .

5. To access the web service from your code, right-click the project in the Solution Explorer,and select Add Web Reference.

Note: If the Add web reference option is not available, select Add Service Reference > Advanced > Add Web Reference.

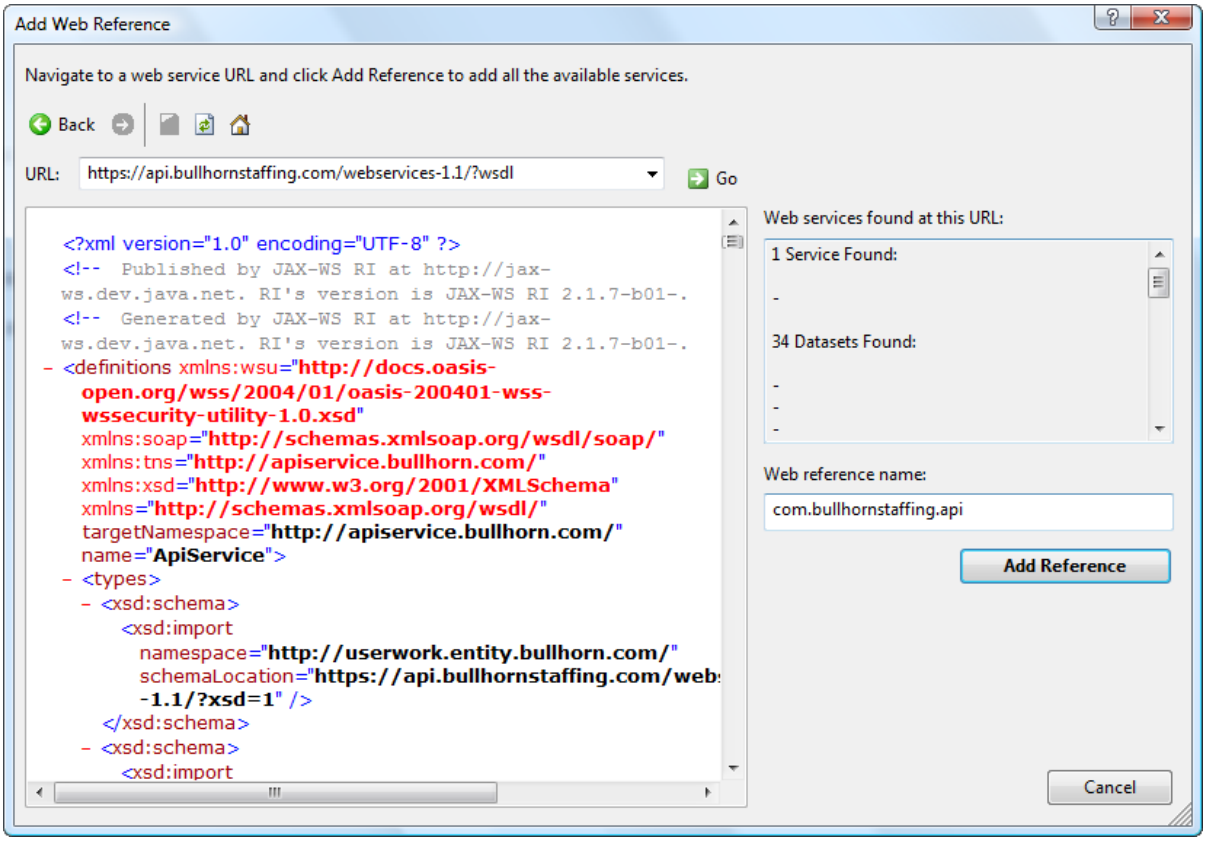


6. In the URL box of the Add Web Reference dialog box, type the URL to obtain the service description of the Bullhorn Web service **https://api.bullhornstaffing.com/webservices-1.1/?wsdl**

Note: At the time of publication, the latest version of the Bullhorn SOAP API is version 1.1

7. Click Go to retrieve information about the web service.

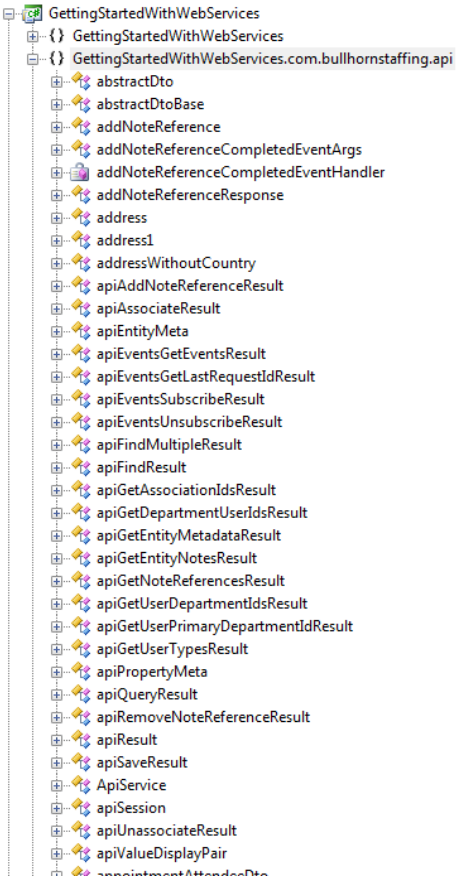
   After the web service is detected, com.bullhornstaffing.api displays in the Web reference name text field.
8. Click Add reference to add the detected web reference to the project.



In the Solution Explorer, a new folder called Web References displays a link to com.bullhornstaffing.api. Expand the node to display all the proxy classes that interface your application and the web service.

9. Double-click the link to open all the client proxy files for the API in an Object Browser.

   You can now explore all the classes and their methods.



Now that the web reference is associated with the project, you can start making calls to the API within your C# code.

Understanding how the Bullhorn SOAP API works

The sample C# client application shows the required steps for creating a session between the client application and the Bullhorn system, and demonstrates the invocation and subsequent handling of some API calls.

You program the sample application to perform the following tasks:

- Create a session to the Bullhorn web service by using your credentials.
- Build a basic query using the `dtoQuery` class.
- Retrieve a single instance of the Candidate DTO.
- Retrieve multiple instances of Candidate DTO.
- Publish the results on a status web page.

Importing the web service in your code

Before you make calls to the API, you must import the web service reference that you created earlier.

1. Open Default.aspx.cs file in the editor.
2. Add the namespace for the web service by inserting the following code to the beginning of the file

```
using GettingStartedWithWebServices.com.bullhornstaffing.api;
```

Note: The web service is referenced through a namespace prepended on the project name.

Creating a session

You must create a session for all calls to the Bullhorn SOAP API. To create a session to the Bullhorn system in the web application, you require a username, password and API key. In this section, you define the authentication mechanism for access and create a session to the Bullhorn service.

1. In the Default.aspx.cs file, define three string variables. Bullhorn Support provides these values, as described in the **Getting Started** section.

```
String username =  "yourusername";
  String password =  "yourpassword";

  String apiKey =  "yourapikey";
```

4. Instantiate the ApiService class by adding the following code.

```
ApiService service = new ApiService();
```

5. Instantiate the apiSession class by adding the following code.

```
apiSession currentSession = new apiSession();
```

6. Start an API session by invoking the `startSession()` method in the ApiService class. The method `startSession()` accepts `username`, `password` and `apiKey` as arguments. The method returns a session object in the response. Assign the session object to the `currentSession` variable that you created.

```
currentSession =  service.startSession(username,password,apiKey);
```

Tip: Refreshing sessions

Sessions have a short expiration and need to be refreshed with each call to the Bullhorn SOAP API. Each call to the Bullhorn SOAP API requires a valid session object, and each response returns a new session object. Store that new session object in the current session variable so you always have a recently refreshed value.

Use the session object that is returned as part of the response in the next call that is made to the API, as previous session objects will expire in 5 minutes.

Building a basic query

Using the Bullhorn SOAP API query operation

One of the most powerful operations in the Bullhorn web services APIs is the query operation. This operation allows you to construct SQL-like queries for a particular type of entity.

The Bullhorn query operation is built on top of Hibernate, an object-relational mapping tool that exposes relational data as a series of objects. The Hibernate Query Language (HQL) is based on standard SQL but adapts its concepts to an object-oriented language (C#). The query operation exposes a subset of the operations supported by HQL.

To execute a query, you must construct a C# query Data Transfer Object (DTO) and then pass it as an argument when you call the `query` operation. The two most important fields in the query DTO are the `entityName` property, which specifies the name of the entity you are querying, and the `where` property, which contains the where clause for your query. In the where property, you specify a single parameter or create a more complex query by using `AND`, `OR`, or `NOT`.

In this section, you create a query that returns the Candidate list with at most 10 candidates that are not deleted.

Note: For referential integrity reasons, records in the Bullhorn system are never deleted. There is an `isDeleted`property on all records that is used to mark the record for deletion.

1. Create an instance of the `dtoQuery` class.

   ```
   dtoQuery candidateQuery = new dtoQuery();
   ```

2. Specify the `entityName` property as Candidate.

   ```
   candidateQuery.entityName =  "Candidate";
   ```

3. Set the `maxResults` property equal to 10.

   ```
   candidateQuery.maxResults = 10;
   ```

4. Set the `maxResultsSpecified` property to true.

   ```
   candidateQuery.maxResultsSpecified = true;
   ```

   Tip: Specified properties

   Any non-string values in .NET must have the `maxResultsSpecified` property set to `true`. If this property is not set, it is not passed with the SOAP call. For more details, refer to the **last section** in the tutorial.

5. Construct the where clause for the query.

   Note: The where clause is a SQL-like string that will be executed by the Bullhorn server. For our purposes, the query only checks to ensure that the Candidate has not been deleted, but you can query on any of the properties exposed on the DTO. For a full list of Candidate properties, see the **reference documentation.**

   ```
   candidateQuery.where = "isDeleted = 0";
   ```

6. In the `query` method in the `service` object, pass the `currentSession` and the `candidateQuery` variables as arguments.

   ```
   apiQueryResult queryResult =  service.query(currentSession, candidateQuery);
   ```

   The method returns a SOAP response that is stored in the `queryResult` variable.

7. Specify the new session in the `currentSession` global variable to refresh the session.

   ```
   currentSession = queryResult.session;
   ```

   Note: Verify results

   Run the debugger to see the `queryResult` variable with the returned data. Notice that the result contains a list of IDs. Each of these IDs corresponds to an instance of a candidate in Bullhorn. There are several operations in Bullhorn that return a list of IDs,
   including `getAssociationIds()` and `getEntityNotes()` methods. When using these operations, usually you then retrieve the specific instance data by using `find()` or `findMultiple()` methods, as explained in the next section.

Retrieving instances of DTOs

After the Bullhorn system returns the results of the query in the `queryResult` variable, you can extract the ID nodes of candidates and use each ID to retrieve a single instance of a DTO or a list of DTOs.

Retrieving a single DTO by using the find() method

The `find()` method in the Bullhorn API allows you to retrieve an instance of the DTO. It needs a session instance, the entity name and the primary key (ID of the entity instance) to retrieve a specific record.

1. Create a loop to access individual ID nodes in the `queryResult` variable.

```
foreach (int i in queryResult.ids)
  {

  }
```

4. Within the loop, you can use the `find()` method in the service class to retrieve an instance of the DTO. The `find()` method takes the session, entity name and the ID of the `candidate` object as arguments and returns the result as a DTO object in the `apiFindResult` class.

```
apiFindResult candidate =  service.find(currentSession, "Candidate", i);
```

5. Assign the new session to the global variable `currentSession` to refresh the session.

```
currentSession = candidate.session;
```

6. Type cast the `dto` object returned in the result as a `candidateDto` object.

```
candidateDto thisCandidate =  (candidateDto)candidate.dto;
```

Retrieving a list of DTOs using findMultiple() method

You can also use the `findMultiple()` method to retrieve several instances of the DTO together. If you know you will need to fetch more than 2-3 DTOs, it is more efficient to use `findMultiple()` as it will reduce the number of round trips required to get the data. However, it will also increase the size of the responses you receive (**see exceeding default size**).

The `findMultiple()` method takes the session, entity name and an array of up to 20 ID nodes as arguments and returns the result as an array of dto objects in the `apiFindMultipleResult` class.

1. After the closing brace of the foreach loop, add the following code to retrieve multiple instances of the candidate DTO.

```
apiFindMultipleResult candidatesMultiple =  service.findMultiple(currentSession,
"Candidate", queryResult.ids);
```

3. Assign the new session to the global variable `currentSession` to refresh the session.

```
currentSession  = candidatesMultiple.session;
```

4. To access individual DTOs, loop over the result array by using the length of the array as a condition.

```
for (int i = 0; i <  candidatesMultiple.dtos.Length; i++ )
  {

  }
```

7. Within the `for` loop, access the individual dto's by type casting each object in the result array to `candidateDto`.

```
candidateDto thisCandidate = (candidateDto)candidatesMultiple.dtos[i];
```

Publishing the results on a web page

To view the results of the data retrieved from a Bullhorn system, create a `DataGrid` component in your ASP code and bind it to a `DataTable` class in C#.

Creating a DataTable class in C#

After the invocation of the `query` method that returns the SOAP response, create an instance of the `DataTable` class. The `DataTable` class includes the three columns for the properties of the candidates that you display on the ASP page.

1. Add the following code to store results of the `find()` method. Add this code after the `query` method returns the result.

```
DataTable dt_candidate  = new DataTable("candidate");
  dt_candidate.Columns.Add("Name",  Type.GetType("System.String"));
  dt_candidate.Columns.Add("Title",  Type.GetType("System.String"));

  dt_candidate.Columns.Add("Date  available", Type.GetType("System.String"));
```

Populating the DataTable class

Add the following code to populate the `DataTable` class with the properties of each of the retrieved candidate object. Add the code for the `find()` and `findMultiple()` methods in this section.

1. In the `foreach` loop for the `find()` method, add the highlighted code to insert a row of data for each candidate into the `DataTable` instance.

```
foreach (int i in  queryResult.ids)
  {
  apiFindResult  candidate = service.find(currentSession, "Candidate", i);
  currentSession = candidate.session;
  candidateDto thisCandidate =  (candidateDto)candidate.dto;
  DataRow row = dt_candidate.NewRow();
  row["Name"] =  thisCandidate.name;
  row["Title"] = thisCandidate.occupation;
  row["Date available"] =  thisCandidate.dateAvailable;
  dt_candidate.Rows.Add(row);

  }
```

Binding the `DataTable` instance to the ASP component

You must bind the in-memory DataTable instance to the user interface component that displays the results in the web page.

1. After the closing brace of the `foreach` loop, add the following code to bind the `DataTable` instance in C# to the `DataGrid` component created in the ASP page.

```
frm_candidate.DataMember = "Name";
  frm_candidate.DataMember = "Title";
  frm_candidate.DataMember = "Date available";
  frm_candidate.DataSource = dt_candidate;

  frm_candidate.DataBind();
```

Note: The `id` of the ASP component you create in the next section is `frm_candidate`.

Storing the session and service object in ASP.NET scope

The session and service object in C# must be stored within the ASP.NET session scope.

1. Add the following lines of code at the end of the `Page_Load` method in the Default.aspx.cs file.

```
Session["service"] = service;

  Session["session"] = currentSession;
```

Code for the `findMultiple()` method

```
DataTable dt_candidate_findMultiple = new  DataTable("candidate");
  dt_candidate_findMultiple.Columns.Add("Name",  Type.GetType("System.String"));
  dt_candidate_findMultiple.Columns.Add("Title",  Type.GetType("System.String"));
  dt_candidate_findMultiple.Columns.Add("Date  available", Type.GetType("System.String"));

  for (int i = 0; i <  candidatesMultiple.dtos.Length; i++ )
  {
  candidateDto  thisCandidate = (candidateDto)candidatesMultiple.dtos[i];
  DataRow  row = dt_candidate_findMultiple.NewRow();
  row["Name"] = thisCandidate.name;
  row["Title"] = thisCandidate.occupation;
  row["Date available"] = thisCandidate.dateAvailable;
  dt_candidate_findMultiple.Rows.Add(row);
  }
  frm_candidateMultiple.DataMember =  "Name";
  frm_candidateMultiple.DataMember =  "Title";
  frm_candidateMultiple.DataMember = "Date  available";
  frm_candidateMultiple.DataSource = dt_candidate_findMultiple;
  frm_ candidateMultiple.DataBind();
```

The `id` of the ASP component you will create in the next section is `frm_candidateMultiple`.

Creating the DataGrid component to display results

In this section, you add the code to create the user interface components that will display the results in a web page. The tutorial is using a `DataGrid` control as an example here.

1. Open Default.aspx file in the editor.
2. Add the following lines of code after the opening div tag.

```
<table  title="Candidate list using find() method">
<tr>
<td>
<asp:DataGrid  ID="frm_candidate" runat="server">
</asp:DataGrid>
</td>
</tr>
</table>

<table  title="Candidate list using findMultiple() method">
<tr>
<td>
<asp:DataGrid  ID="frm_candidateMultiple" runat="server">
</asp:DataGrid>
</td>
</tr>
</table>
```

Running the completed application

Build and run your application to see the candidates retrieved from the Bullhorn system on the web page.

Tips for using the .NET Framework and the Bullhorn WSDL

*Specified properties

When you use .NET with the Bullhorn WSDL, .NET generates an extra Boolean field for each non-string field. For example, if you have a date value in the candidate.dateAvailable field, .NET generates a Boolean field called `candidate.dateAvailableSpecified`. The default value for the field is `false`. If a Specified field value is `false`, the value in the corresponding original field is not included in the SOAP message generated by your client application. To include the field, you must set the Boolean field to `true`.

```
DateTime dateTime = DateTime.Now;
candidate.dateAvailable = dateTime;
candidate.dateAvailableSpecified =  true;
```

For more information, please refer to the link **http://msdn.microsoft.com/en-us/library/bb402199.aspx**

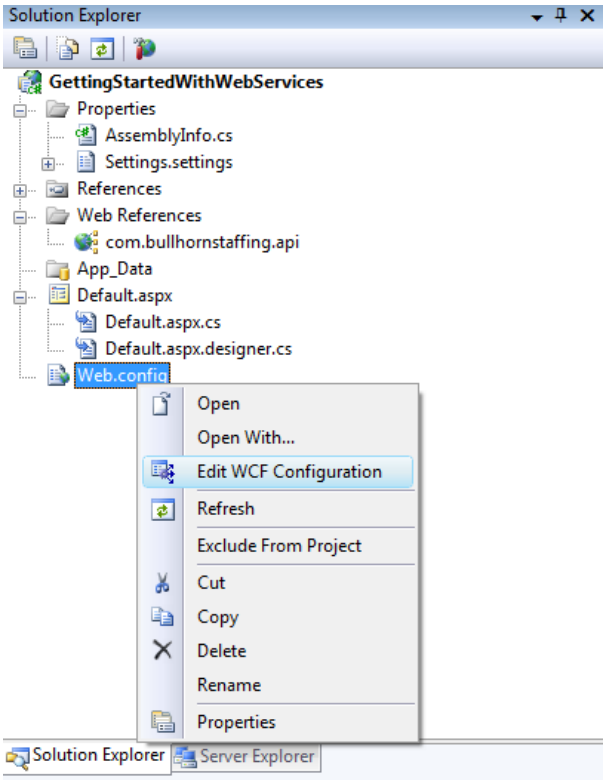
Exceeding default values for XML classes

Some of the DTOs in the Bullhorn system exceed the default size values for the XML classes in the .NET framework. For these DTOs, you adjust the MaxStringContentLength, MaxReceivedMessageSize and MaxBufferSize properties.

Editing properties within the .NET framework

To modify the default values, follow the steps in this section.

1. In the Solution Explorer, right-click the Web.config file and select Edit WCF Configuration.

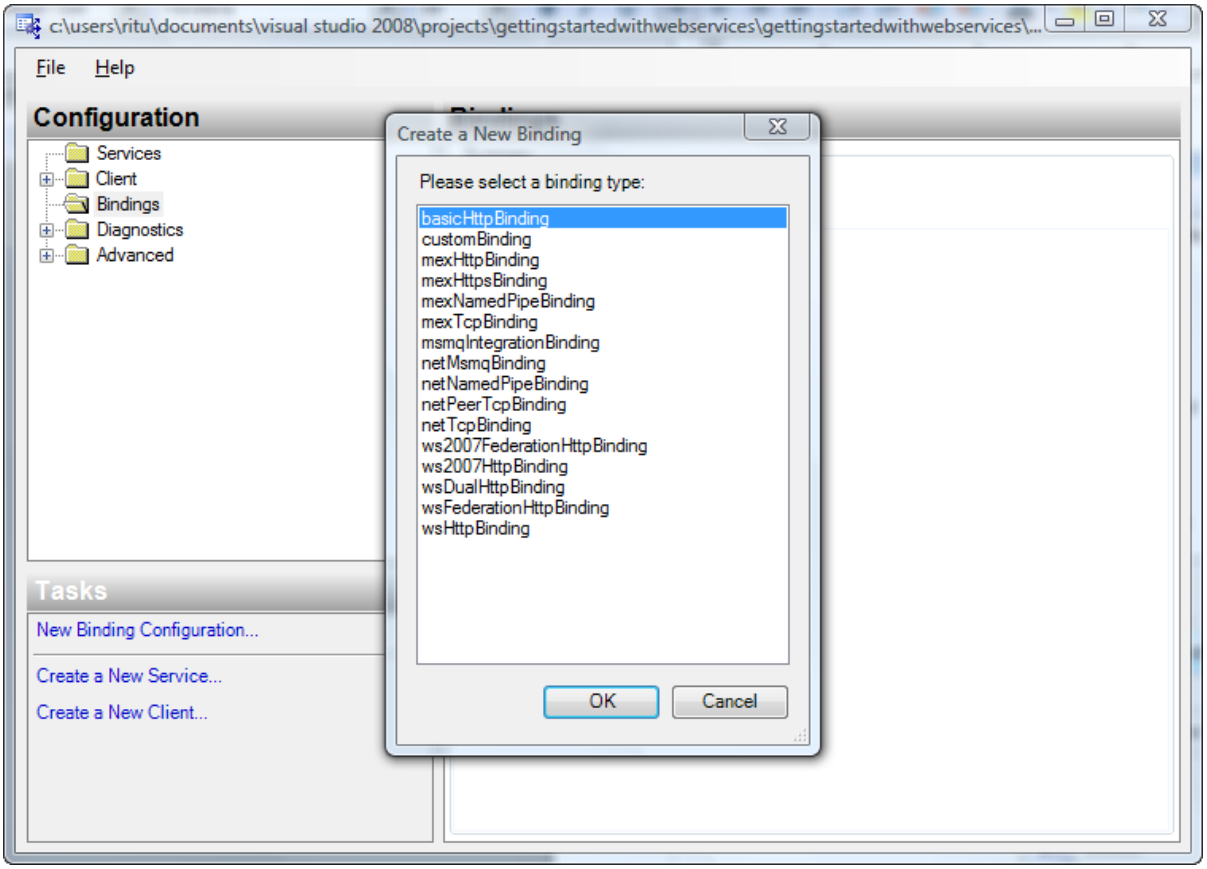


If the option for Edit WCF Configuration is not available, right-click the Web.config file and select Open With. Click Add. In the Add Program dialog box, navigate to C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\SvcConfigEditor.exe(,or just copy paste this path).

Set Friendly Name to WCF Config Editor.

Now, you can right-click the Web.config file, select Open With > WCF Config Editor, and click OK.

2. In the dialog box, select Bindings > New Binding Configuration.
3. Select basicHttpBinding.



4. Click OK.

In the editor, look at the following two properties:

- maxReceivedMessageSize property – This property is a positive integer that defines the maximum message size, in bytes, including headers, for a message that can be received on a channel configured with this binding. The sender receives a SOAP fault if the message is too large for the receiver. Edit the field value for your application if you want to increase the default value. For more information, see **http://msdn.microsoft.com/en-us/library/ms731361.aspx**
- maxStringContentLength property on the ReaderQuota – This property is a positive integer that specifies the maximum characters allowed in XML element content. The default is 8192. Edit this field value for your application if you want to change the default value. For more information, see **http://msdn.microsoft.com/en-us/library/ms731325.aspx**