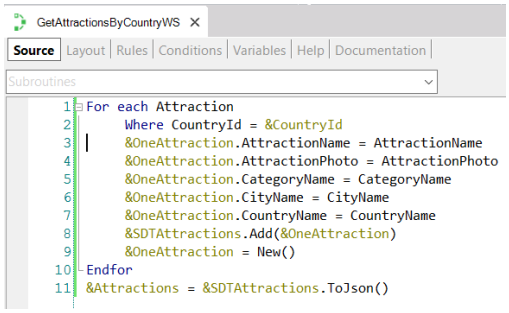
Web Services. Advanced topics.

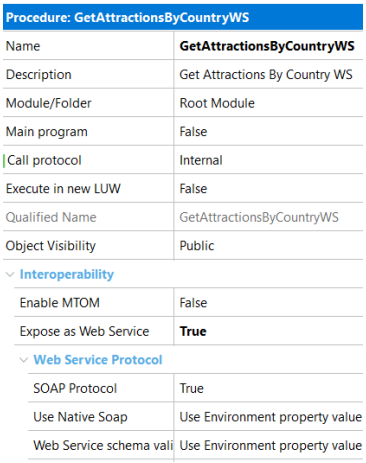
Publishing SOAP services with GeneXus

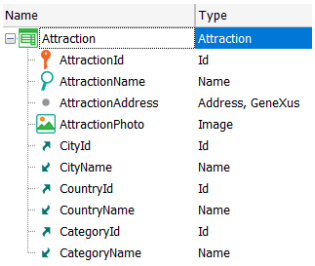
GeneXus™

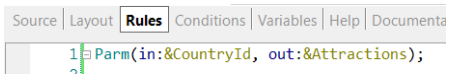
In this video, we will focus on publishing, testing, and customizing SOAP services with GeneXus, including assigning a namespace, or including more than one method in a single web service.

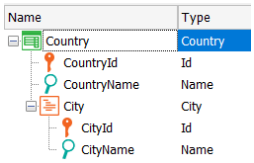
Publishing a procedure as a SOAP web service

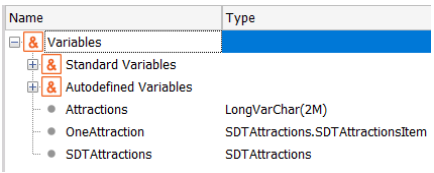


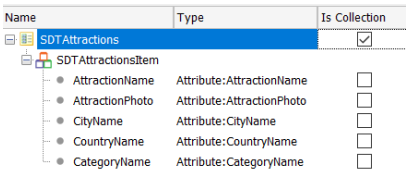


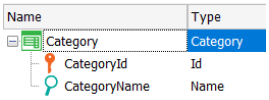












We are going to publish a procedure object as a service.
This service will access the database of an application for a Travel Agency, and will return the collection of tourist attractions registered by the agency that belong to a given country.

To do so, we create a procedure object and name it GetAttractionsByCountryWS. When we publish a web service, it is important to choose a name that helps identify the function of the service, so that it is clear to the consumer.

The procedure receives the country identifier by parameter, and returns in a string a JSON structure with the list of attractions that belong to the received country.

To define the structure, we create an SDT collection with the data of the attractions we want to return, and in the procedure we create an SDTAttractions variable of the SDT collection data type, a OneAttraction variable of the collection item type, and an Attractions variable of the LongVarChar type that will contain the JSON that the procedure will return.

In the source, the For Each navigates the Attraction table (associated with the Attraction base transaction) filtered by the country received by parameter. For each attraction encountered, it loads the values into an item and then the item is added to the collection. Finally, the collection SDT variable is serialized to a JSON and assigned to the output parameter variable.

To expose the GetAttractionsByCountryWS procedure as a SOAP service, we set the Expose as Web Service property to True, and set the SOAP Protocol property

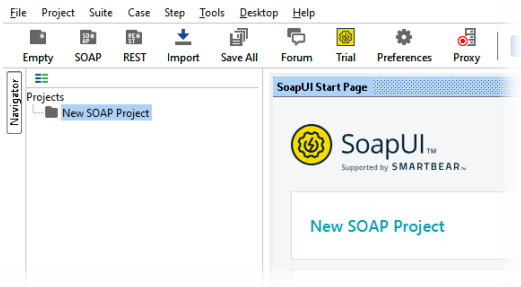
to True and REST Protocol to False.

For the service to be published on the web server, we must do a Build.

Testing the procedure published as a SOAP web service

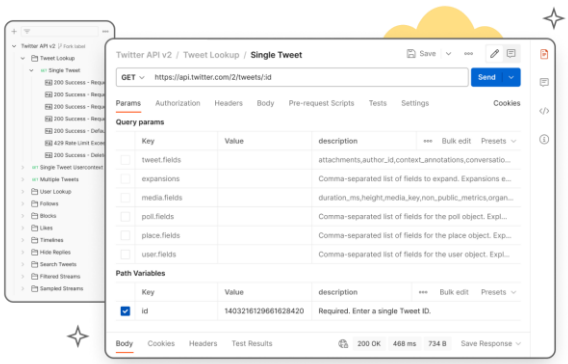
SoapUI
Supported by SMARTBEAR

https://www.soapui.org/



POSTMAN

https://www.postman.com/



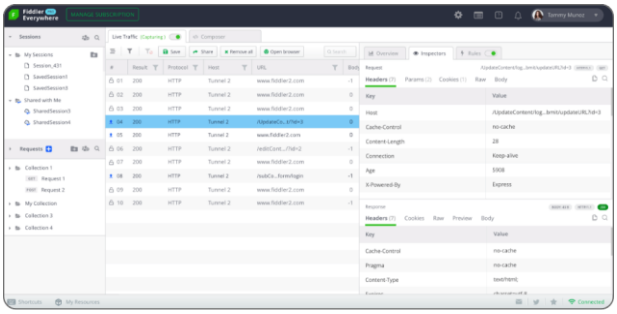
Now we will verify that the procedure is correctly exposed as a SOAP service.

To test it, we can use several tools, such as SOAP UI or POSTMAN. These tools allow consuming the web service as clients, to check if the service is working correctly and to obtain detailed information about the process, both for SOAP and REST web services.

Testing the procedure published as a SOAP web service

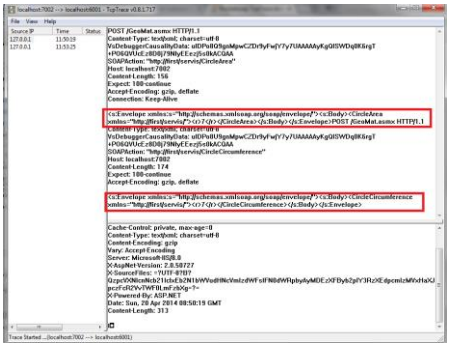
Progress Telerik Fiddler

https://www.telerik.com/fiddler



TcpTrace

https://sourceforge.net/projects/open-tcptrace/

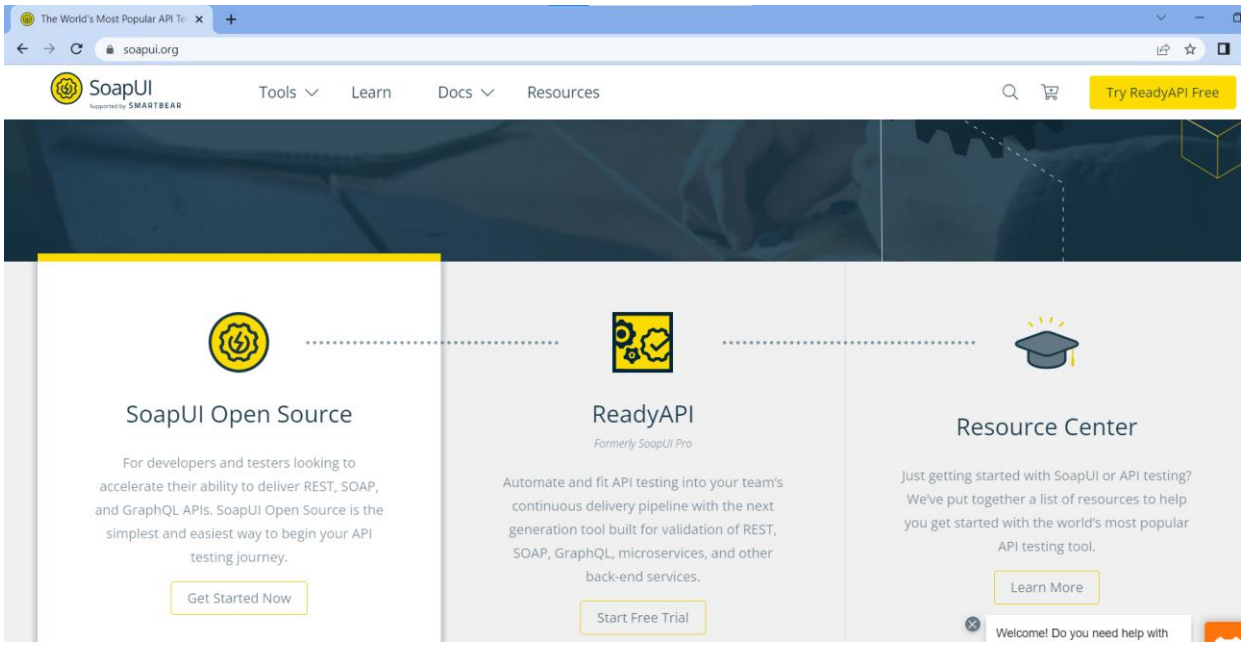


It is also useful to use tools that allow you to view the service flow; that is, to see the execution request to the server (Request) and the response of the service (Response).

Two of the best known tools are Fiddler and TcpTrace. TcpTrace is free (open source), but only allows HTTP flow; for HTTPS, Fiddler must be used.

These tools are between the consumer application and the web server, as if it were a proxy, and show the Request and Response between the client and the service.

Installing SoapUI Open Source

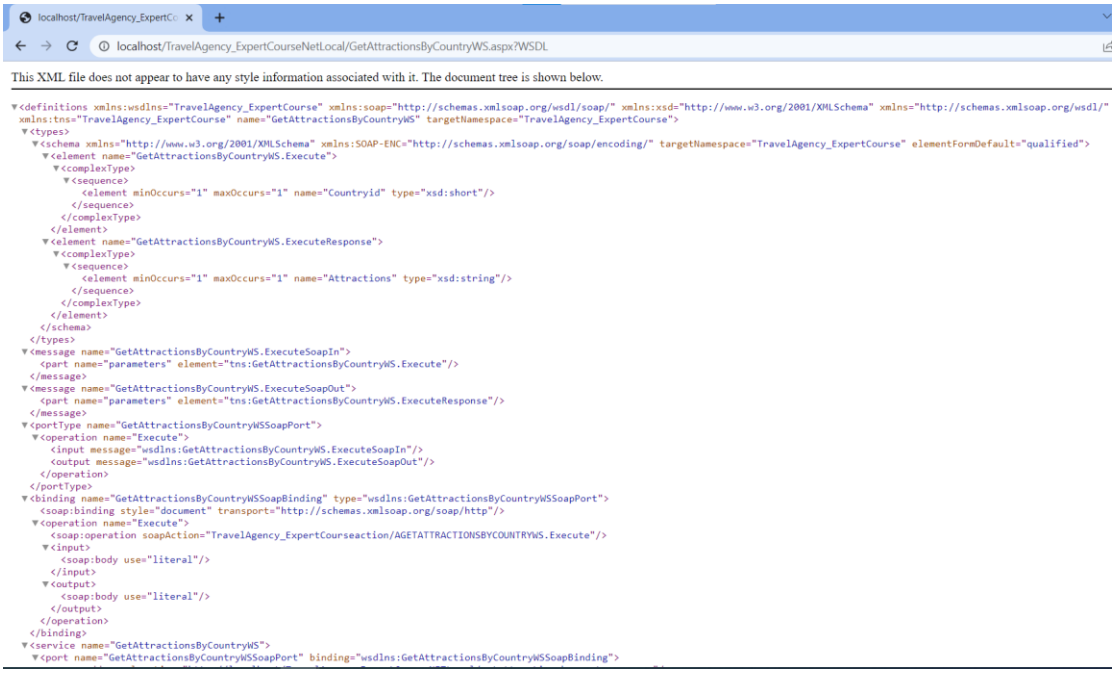


We are going to use the SoapUI tool to test our service. To do so, we open the soapui.org web page and download SoapUI OpenSource.
Next, we install it and when we run it the Start Page appears. If in Resources we click on Test a SOAP API, it opens a page with instructions, so we will follow them.

We create a new SOAP Project; in the Project name, we type GetAttractionsByCountry and press OK.
Now we right-click on the Project and choose Add WSDL.

This WSDL file was generated by GeneXus following the Web Services Description Language specification; it contains the information of how our web service is structured, including its methods, the parameters of each method, etc.

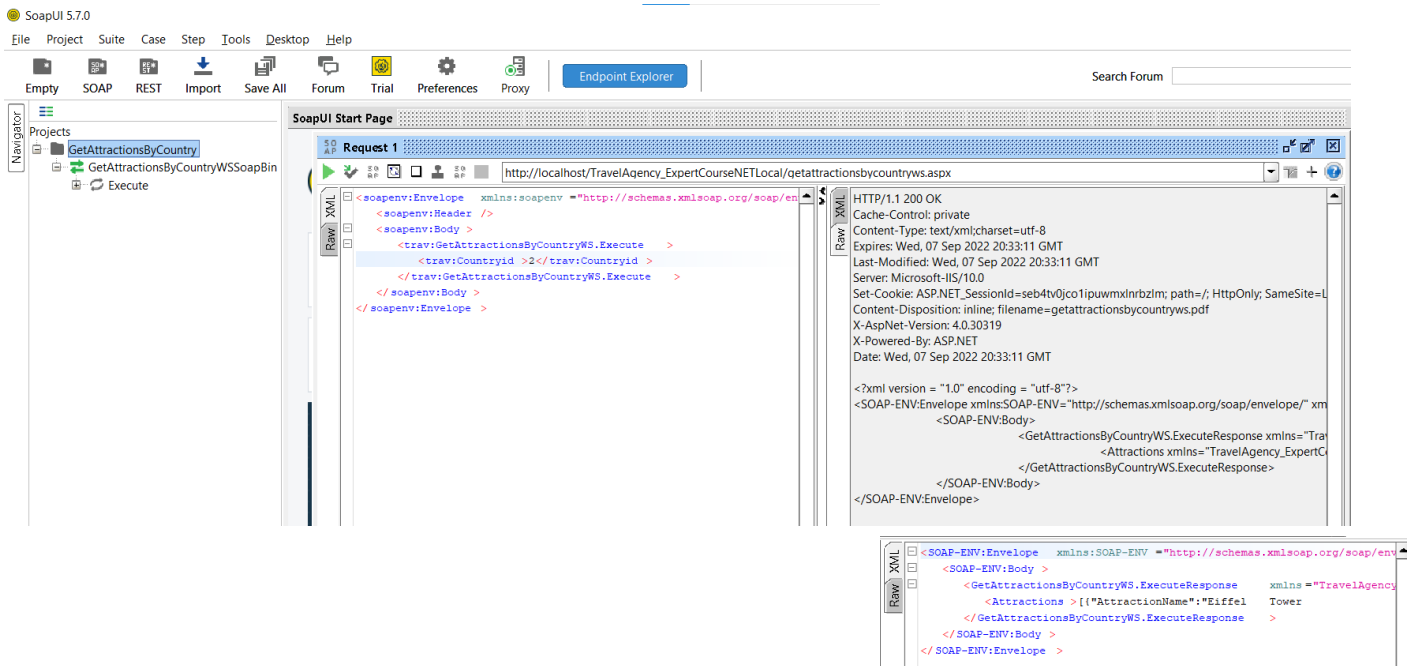
Using the browser to discover the WSDL structure



To view the contents of this file, we open a browser window and type the URL of our service. Since we are generating in .NET, we write the URL formed by concatenating the URL of the Web root property of the generator, and the name of our exposed procedure. Then we add .aspx, a question mark, and the letters WSDL. If the generator were Java, we would add /servlet before the object name and not the aspx extension.
If the exposed object were a Business Component, the BC name would be followed by "_BC" if the generator were . NET, and by "_BC_WS" if the generator were Java.
We press Enter and see that the browser shows the web service structure, where we identify some things such as the name, the Execute method required by the CountryId parameter, etc.

[http://localhost/TravelAgency_ExpertCourseNETLocal/GetAttractionsByCountryWS.aspx?WSDL]

Testing our web service in SoapUI



Now that we know that the WSDL was opened correctly, we are going to enter the same URL in the SoapUI project window and press OK.

Under the project we created, an entry is displayed with the name of our service and the Execute method. If we press the + button, a Request is automatically generated to invoke the service. We double-click and the Request Editor opens; on the left side there is a template of the XML for the invocation, and on the right side there is the response given by the service when it is invoked.

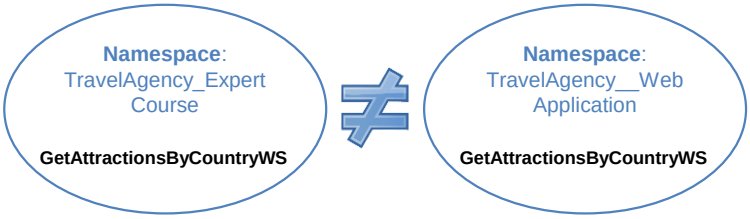
In the request window, in the Execute method we identify the CountryId parameter, so we replace the question mark with the identifier of the country for which we want to obtain information about its attractions. We write 2, which is the ID of France.

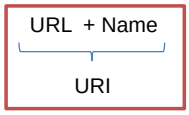
Now we press the Play button and see that the structure of the response appears on the right. In the Attractions node, moving to the right, the collection of tourist attractions in France appears in straight brackets.

We can also see this information as XML, and we verify that our GetAttractionsByCountry service is working perfectly.

Use of Namespaces in a SOAP web service

Procedure: GetAttractionsByCountryWS	
Name	**GetAttractionsByCountryWS**
Description	Get Attractions By Country WS
Module/Folder	Root Module
Main program	False
Call protocol	**SOAP**
Execute in new LUW	False
Qualified Name	GetAttractionsByCountryWS
Object Visibility	Public
Interoperability	
Exposed namespace	TravelAgency_ExpertCourse
Enable MTOM	False
Expose as Web Service	**True**
Web Service Protocol	
SOAP Protocol	True
Use Native Soap	Use Environment property value
Web Service schema validation	Use Environment property value
REST Protocol	False





Now let's look at the concept of namespace and how this can be useful when publishing a service.

A namespace is a container of names where the same name cannot be repeated. However, the same name can be present in more than one namespace.

The **Exposed namespace** property allows us to assign a namespace to a service.

This property contains a string that helps to identify the web service. The combination of the namespace with the name of the web service must be unique; in this way, if we have two services with the same name—but which belong to different applications—they will be correctly identified by changing the namespace.

By default, the Exposed namespace property has the name of the KB and that does not cause any problems if we are in the prototyping stage. However, when we move the service to the production environment, we must write in this property a URL that identifies the company or the project to which the service belongs; otherwise, some consumers will not be able to process it.

When adding security to web services, it is crucial that this URI (Uniform Resource Identifier), which consists of the URL and the name of the service, be well formed and unique.

Procedure exposed as SOAP Web service with more than one method

Rules

```
Parm (in: param1, in:param2, in: param3, out: param4)
```

Source

```
Stub mehod1 (in: param1, in:param2, out: param4)
  .....
  .....
EndStub

Stub mehod2 (in: param3, out: param4)
  .....
  .....
EndStub
```

Stubs use: Only in SOAP web services

Something that may arise when exposing a service is how we can include several methods within the same service.

As we already confirmed, if we expose a procedure object as a web service, the service includes only one method: Execute.

If you need to define more than one method in the same web service, you must use stubs in the procedure source.

Stubs are clauses which, within the source of a procedure object, allow us to define a block of code associated with a name and then execute the code by invoking that name. The concept is similar to that of a subprogram or subroutine and each stub can have its own parameters.

It is important to point out that the use of stubs is only valid when we expose the procedure with the SOAP protocol; the procedures exposed as REST do not support this functionality.

SOAP web service with more than one method

Source | Layout | **Rules** | Conditions | Variables | Help | Documentation

```
1  Parm(in:&CountryId, in:&TripsQty, out:&Attractions);
```

Source * | Layout | Rules | Conditions | Variables * | Help | Documentation

Subroutines

```
1  Stub AllAttractionsByCountry(in:&CountryId, out:&Attractions)
2      For each Attraction
3          Where CountryId = &CountryId
4          &OneAttraction.AttractionName = AttractionName
5          &OneAttraction.AttractionPhoto = AttractionPhoto
6          &OneAttraction.CategoryName = CategoryName
7          &OneAttraction.CityName = CityName
8          &OneAttraction.CountryName = CountryName
9          &SDTAttractions.Add(&OneAttraction)
10         &OneAttraction = New()
11     Endfor
12     &Attractions = &SDTAttractions.ToJson()
13 EndStub
14
15 Stub AttractionsByCountryWithTrips(in: &CountryId, in:&TripsQty, out:&Attractions)
16     For each Attraction
17         Where CountryId = &CountryId
18         Where Count(TripDate) >= &TripsQty
19         &OneAttraction.AttractionName = AttractionName
20         &OneAttraction.AttractionPhoto = AttractionPhoto
21         &OneAttraction.CategoryName = CategoryName
22         &OneAttraction.CityName = CityName
23         &OneAttraction.CountryName = CountryName
24         &SDTAttractions.Add(&OneAttraction)
25         &OneAttraction = New()
26     Endfor
27     &Attractions = &SDTAttractions.ToJson()
28 EndStub
```

Name	Type
Attraction	Attraction
AttractionId	Id
AttractionName	Name
AttractionAddress	Address, GeneXus
AttractionPhoto	Image
CityId	Id
CityName	Name
CountryId	Id
CountryName	Name
CategoryId	Id
CategoryName	Name

Name	Type
Country	Country
CountryId	Id
CountryName	Name
City	City
CityId	Id
CityName	Name

Name	Type
Category	Category
CategoryId	Id
CategoryName	Name

Name	Type
Trip	Trip
TripId	Id
TripDate	Date
TripDescription	VarChar(1K)
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastNa...	Character(20)
CustomerFulName	Name
Attraction	Attraction
AttractionId	Id
AttractionName	Name
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name
TripAttraction...	Numeric(4.0)

Name	Type
Variables	
Standard Variables	
Autodefined Variables	
Attractions	LongVarChar(2M)
OneAttraction	SDTAttractions.SDTAttractionsItem
SDTAttractions	SDTAttractions
TripsQty	Numeric(4.0)

Let's see an example of exposing a SOAP procedure with more than one method.

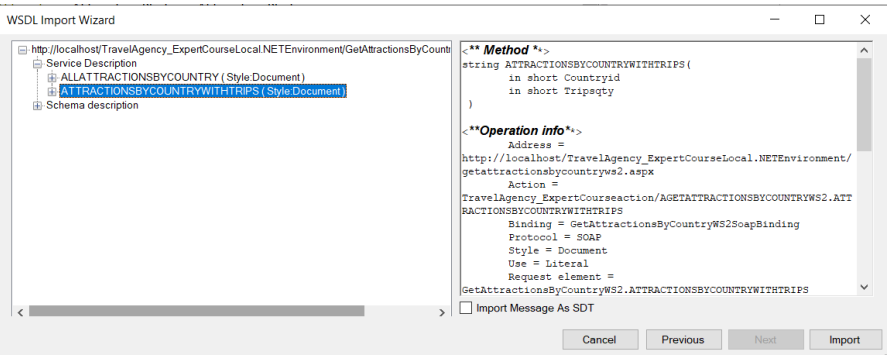
Let's suppose that we want the procedure GetAttractionsByCountryWS that we exposed as a web service to have two methods, one to bring all the tourist attractions of a country and another one to return only the attractions of a country with a number of trips greater than or equal to a given number.

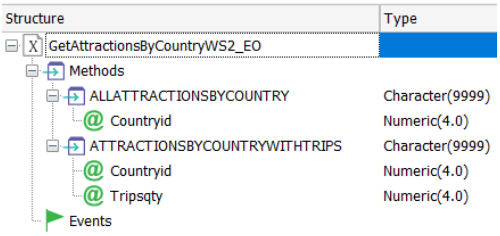
We save the procedure GetAttractionsByCountryWS as GetAttractionsByCountryWS2 and modify its parm rule by adding the variable &TripQty as input parameter.

In the source we create 2 stubs, one named AllAttractionsByCountry, which receives as parameter the CountryId and returns a JSON with all the attractions of that country (with or without trips), and another stub named AttractionsByCountryWithTrips, which receives as parameters the CountryId and the number of trips by which we want to filter, and returns a JSON with the attractions found that have the same number or more trips than the value of the variable &TripsQty.

Let's do a build all so that the service is published on the server.

SOAP web service with more than one method





We are going to import the new web service we created. To do so, we run the wizard, type the new URL with the name of the new object and press Next.

Next, we modify the name of the suggested external object by adding _EO, type the name of a destination folder, and click on Next.

Now if we open the Service Description node, we see that the Execute method is no longer there and there are the two methods corresponding to the stubs we created.

If we select each method, in the window on the right we can see the parameters, which match those previously defined in the stubs.

If for compatibility reasons it is required that the Execute method continues to be exposed, a stub with the name Execute must be expressly created.

Moving on with the wizard, we click on Import and verify that the external object was created inside the folder we already had. If we open it, we confirm that it has the 2 methods that we defined.

In this way, we've experienced the flexibility of including several methods in the same service, something very useful for the consumer of our SOAP web service.

[http://localhost/TravelAgency_ExpertCourseNETLocal/GetAttractionsByCountryWS2.aspx?WSDL]

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications