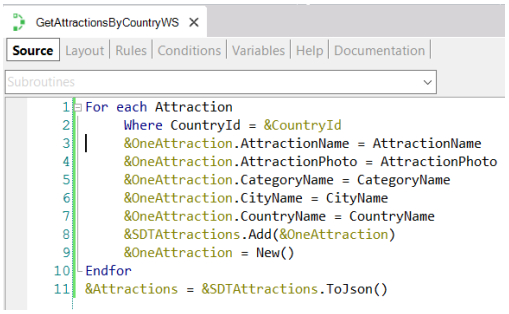
Web Services. Aspectos avanzados.

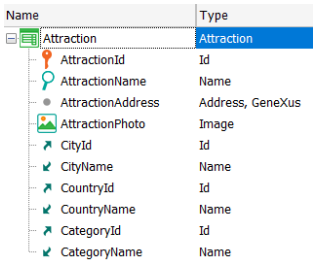
Publicando servicios SOAP con GeneXus.

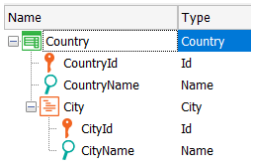
GeneXus™

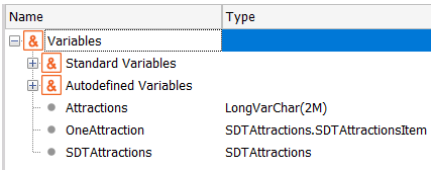
En este video nos concentraremos en la publicación, prueba y personalización de servicios SOAP con GeneXus, como por ejemplo asignarle un namespace, o incluir más de un método en un único web service.

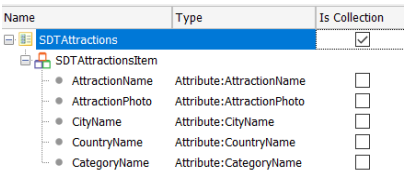
Publishing a procedure as SOAP web service

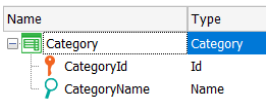












Vamos a publicar un objeto procedimiento como servicio.
Este servicio accederá a la base de datos de la aplicación de una Agencia de Viajes y devolverá la colección de atracciones turísticas registradas por la agencia, que pertenecen a un país dado.

Para esto creamos un objeto procedimiento y le ponemos de nombre GetAttractionsByCountryWS. Cuando publicamos un web service, es importante elegir un nombre que ayude a identificar qué función cumple el servicio, de forma que sea claro para quien lo consuma.

El procedimiento recibe por parámetro al identificador de país y devuelve en un string una estructura JSON con la lista de atracciones que pertenecen al país recibido.

Para definir la estructura creamos un SDT colección con los datos de las atracciones que queremos devolver y en el proc creamos una variable SDTAttractions del tipo de datos del SDT colección, una variable OneAttraction del tipo del item de la colección y una variable Attractions del tipo LongVarChar que contendrá el JSON que devolverá el procedimiento.

En el source, el For Each navega la tabla Attraction (asociada a la transacción base Attraction) filtrada por el país recibido por parámetro, para cada atracción encontrada carga los valores en un item y luego se agrega el item a la colección.
Por último, la variable SDT colección se serializa en un JSON y se asigna a la variable del parámetro de salida.

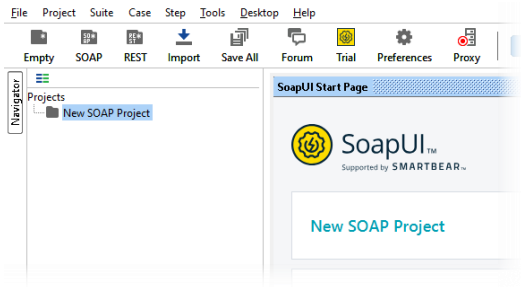
Para exponer el procedimiento GetAttractionsByCountryWS como servicio SOAP, ponemos la propiedad Expose as Web Service en el valor True, y seteamos la propiedad SOAP Protocol en True y REST Protocol en False.

Para que el servicio se publique en el servidor web, debemos hacer un Build.

Testing the procedure published as SOAP web service

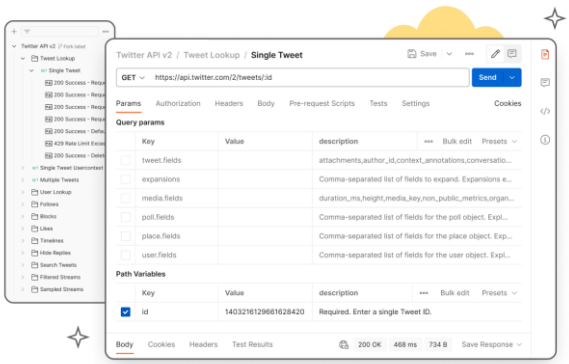
SoapUI
Supported by SMARTBEAR

https://www.soapui.org/



POSTMAN

https://www.postman.com/



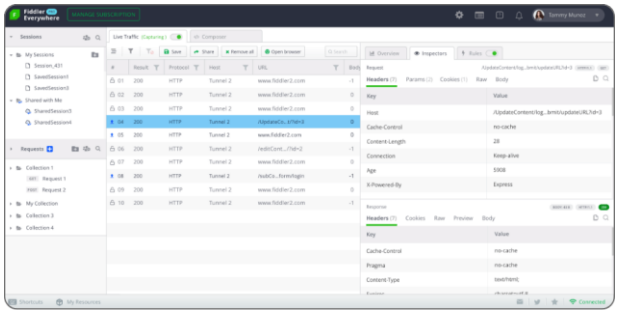
Ahora verificaremos que el procedimiento quedó correctamente expuesto como servicio SOAP.

Para probarlo, podemos usar varias herramientas, como SOAP UI o POSTMAN. Estas herramientas permiten consumir el web service como si fueran clientes, para ver si el servicio funciona correctamente y obtener información detallada del proceso, tanto para web services SOAP como REST.

Testing the procedure published as SOAP web service

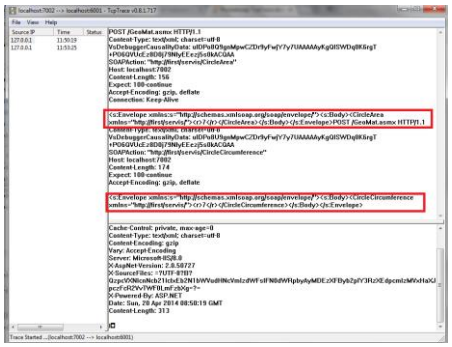
Progress Telerik Fiddler

https://www.telerik.com/fiddler



TcpTrace

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

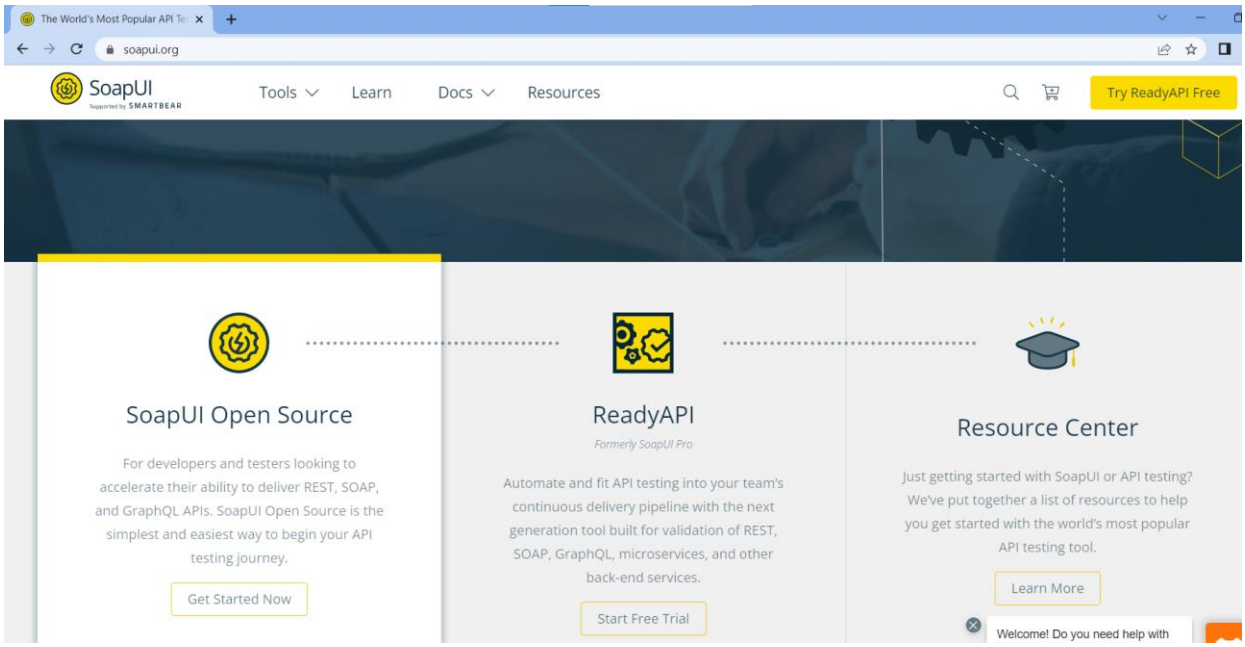


También es útil el uso de herramientas que permitan ver el flujo del servicio, es decir cómo fue la solicitud de ejecución al servidor (Request) y cómo fue la respuesta del servicio (Response).

Dos de las más conocidas son Fiddler y TcpTrace. TcpTrace es gratuita (open source), pero solo permite flujo http, para https debe usarse Fiddler.

Estas herramientas se paran en el medio entre la aplicación que consume y el servidor web, como si fuera un proxy y muestran el Request y el Response entre el cliente y el servicio.

Installing SoapUI Open Source

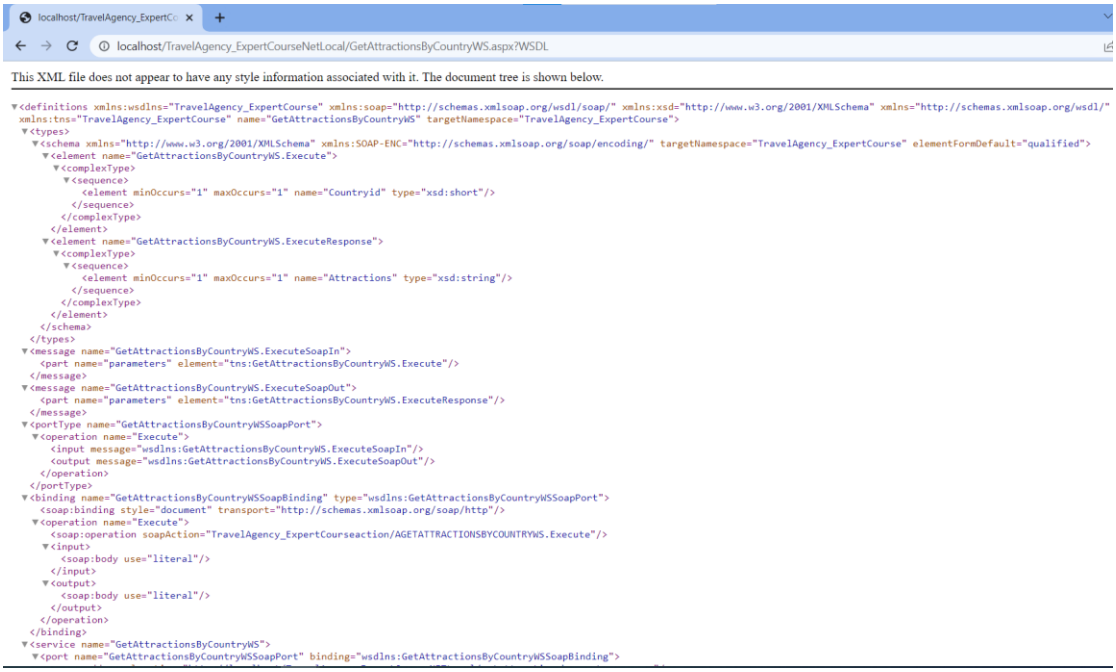


Vamos a utilizar la herramienta SoapUI para chequear nuestro servicio. Abrimos la página web soapui.org y descargamos SoapUI OpenSource.
La instalamos y cuando ejecutamos aparece la Start Page. Si en Resources hacemos clic en Test a SOAP API nos abre una página con instrucciones, así que vamos a seguirlas.

Creamos un Proyecto SOAP nuevo, en el nombre del Proyecto ponemos GetAttractionsByCountry y presionamos OK.
Ahora damos botón derecho sobre el Proyecto y elegimos Add WSDL.

Este archivo WSDL fue generado por GeneXus siguiendo la especificación Web Services Description Language y contiene la información de cómo está estructurado nuestro web service, como por ejemplo los métodos que tiene, los parámetros de cada método, etc.

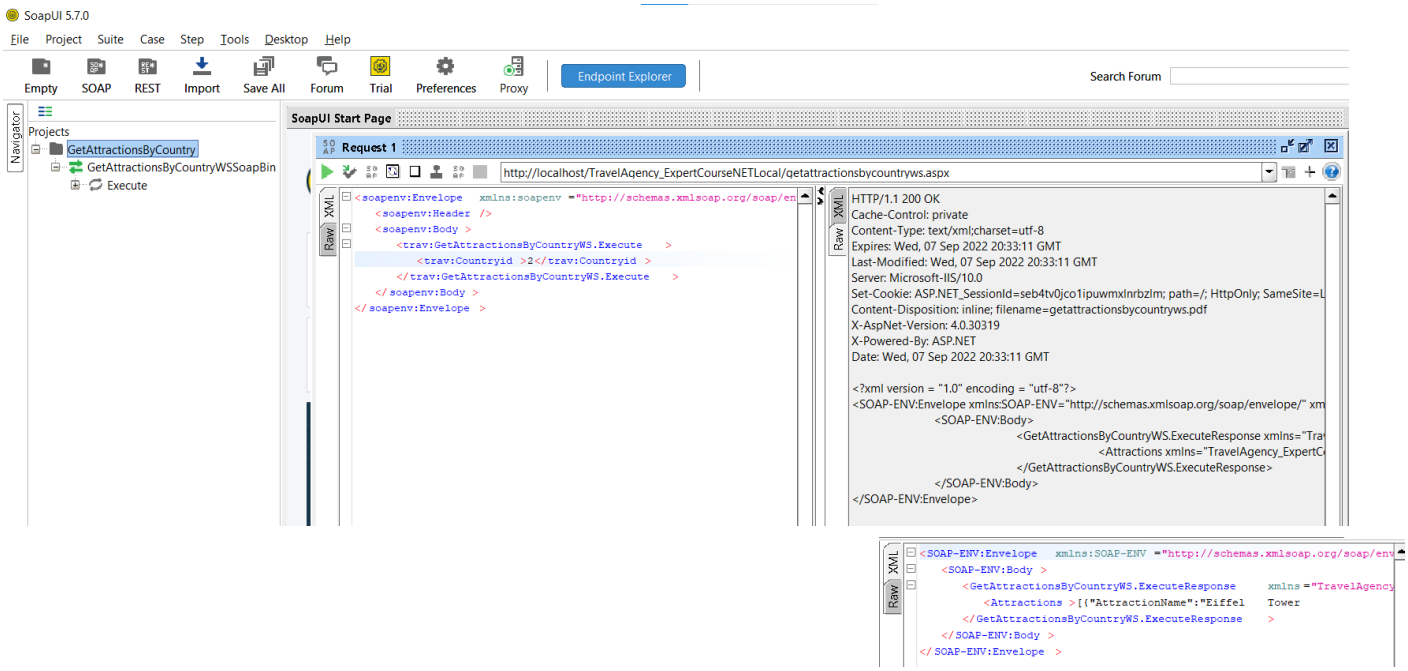
Using de browser to discover WSDL structure



Para ver el contenido de este archivo, abrimos una ventana del browser y escribimos la URL de nuestro servicio, que como estamos generando en .NET, escribimos la URL que formamos concatenando la URL de la propiedad Web root del generador, el nombre de nuestro procedimiento expuesto, luego .aspx, agregando después un signo de interrogación y las letras WSDL. Si el generador fuera Java, agregaríamos /servlet antes del nombre del objeto y no va la extensión aspx.
Si el objeto expuesto era un Business Component, el nombre del BC estará seguido de "_BC" si el generador es .NET y "_BC_WS" si el generador es Java.
Presionamos Enter y vemos que el browser nos muestra la estructura de nuestro web service, donde identificamos algunas cosas como el nombre, el método Execute que requiere el parámetro CountryId, etc.

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

Testing our web service in SoapUI



Ahora que sabemos que el WSDL se abrió correctamente, vamos a ingresar la misma URL en la ventana del proyecto SoapUi y presionamos OK.

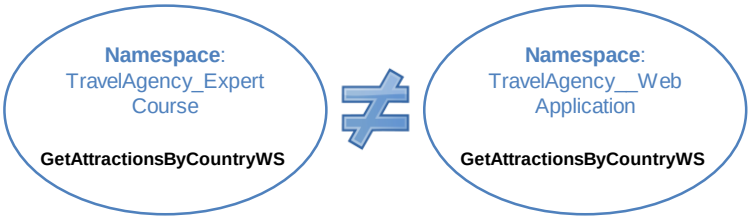
Vemos que bajo el proyecto que creamos, aparece una entrada con el nombre de nuestro servicio y el método Execute. Si presionamos el botón de + vemos que se nos generó automáticamente un Request para invocar al servicio. Damos doble clic y vemos que se abre el Request Editor, donde del lado izquierdo aparece un template del XML para la invocación y del lado derecho aparecerá la respuesta dada por el servicio al invocarlo.

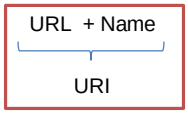
En la ventana del request, en el método Execute identificamos el parámetro CountryId, así que sustituimos al signo de interrogación por el identificador del país del cual queremos obtener información de sus atracciones. Escribimos 2 que es el Id de Francia.

Ahora presionamos el botón de Play y vemos que a la derecha apareció la estructura del response. Vemos que en el nodo Attractions, desplazándonos hacia la derecha, aparece entre paréntesis rectos, la colección de las atracciones turísticas de Francia.
También podemos ver esta información como XML, y verificamos que nuestro servicio GetAttractionsByCountry está funcionando perfectamente.

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





Veamos ahora el concepto de namespace y cómo esto nos puede ser útil a la hora de publicar un servicio.

Un espacio de nombres es un contenedor de nombres donde el mismo nombre no se puede repetir. Sin embargo, el mismo nombre puede estar presente en más de un namespace.

La propiedad **Exposed namespace** nos permite asignarle un espacio de nombres a un servicio.

Esta propiedad contiene un string que ayuda a identificar al web service. La combinación del namespace con el nombre del web service debe ser única, de modo que si tenemos dos servicios con el mismo nombre, pero que pertenecen a aplicaciones diferentes, cambiando el namespace quedan correctamente identificados.

Por defecto, la propiedad Exposed namespace tiene el nombre de la KB y eso no trae inconvenientes si estamos en etapa de prototipación, pero cuando pasamos el servicio a ambiente de producción, debemos escribir en esta propiedad una URL que identifique a la empresa o al proyecto al que pertenece el servicio, de lo contrario, algunos consumidores no lo van a poder procesar.

El hecho de que este URI (Uniform Resource Identifier), compuesto por la URL y el nombre del servicio, esté bien formado y sea único, cobra vital importancia cuando se agrega seguridad a los web services.

Procedure exposed as Web service SOAP, 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

Una cosa que puede surgirnos a la hora de exponer un servicio, es cómo podemos incuir varios métodos dentro del mismo servicio.

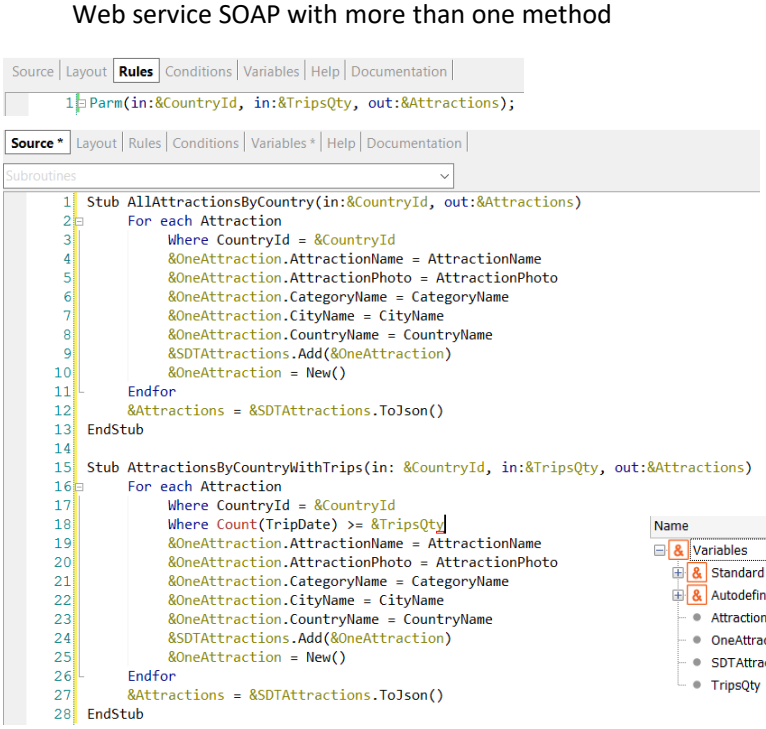
Como ya corroboramos, si exponemos un objeto procedimiento como web service, el servicio incluye un único método: Execute.

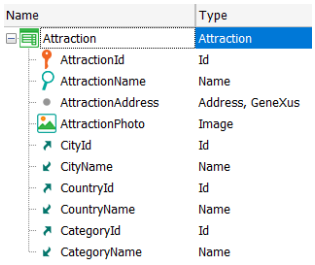
Si se requiere definir mas de un método en el mismo web service, se debe hacer uso de stubs en el source del procedimiento.

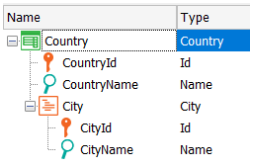
Los stubs son cláusulas que, dentro del source de un objeto procedimiento, nos permiten definir un bloque de código asociado a un nombre y luego ejecutar el código invocando a ese nombre. El concepto es similar al de un subprograma o subrutina y cada stub puede tener sus propios parámetros.

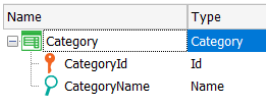
Algo importante a aclarar, es que el uso de stubs solamente es válido cuando exponemos el procedimiento con protocolo SOAP, los procedimientos expuestos como REST no soportan esta funcionalidad.

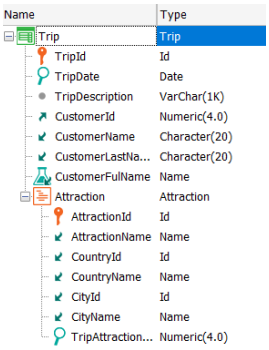
Web service SOAP with more than one method

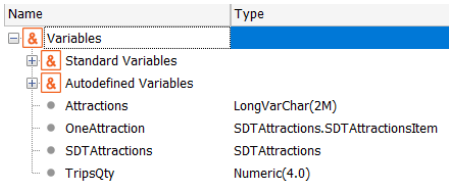












Veamos un ejemplo de exponer un procedimiento SOAP con más de un método.

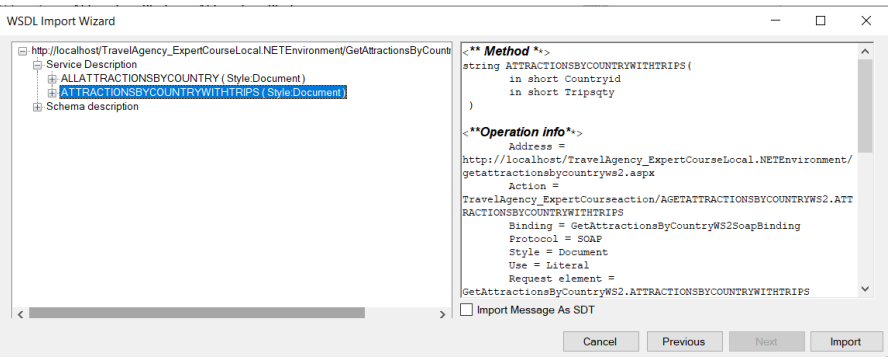
Supongamos que el procedimiento GetAttractionsByCountryWS que expusimos como web service, queremos que tenga dos métodos, uno para traernos todas las atracciones turísticas de un país y otro que nos devuelva únicamente las atracciones de un país que hayan tenido una cantidad de visitas (trips) mayor o igual a un número dado.

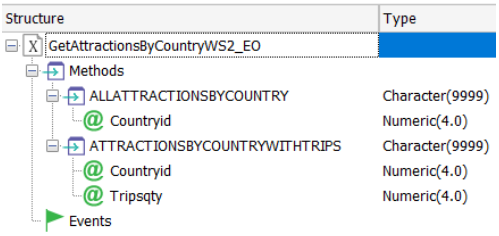
Salvamos al procedimiento GetAttractionsByCountryWS como GetAttractionsByCountryWS2 y modificamos su regla parm agregando la variable &TripQty como parámetro de entrada.

En el source creamos 2 stubs, uno con el nombre AllAttractionsByCountry, que recibe como parámetro el CountryId y devuelve un JSON con todas las atracciones de ese país (tengan o no tengan trips), y otro stub de nombre AttractionsByCountryWithTrips, que recibe como parámetros al CountryId y la cantidad de trips por la que queremos filtrar, y devuelve un JSON con las atracciones encontradas que tengan igual cantidad o más viajes que el valor de la variable &TripsQty.

Hagamos un build all para que el servicio se publique en el server.

Web service SOAP with more than one method





Vamos a importar el nuevo web service que creamos. Ejecutamos el wizard, escribimos la nueva URL con el nombre del nuevo objeto y presionamos Next.

Modificamos el nombre del objeto externo sugerido agregando _EO, escribimos el nombre de un folder de destino y presionamos Next.

Ahora si abrimos el nodo Service Description vemos que ya no está más el método Execute y están los dos métodos que corresponden a los stubs que creamos.

Si seleccionamos cada método, vemos que en la ventana de la derecha podemos ver los parámetros, que coinciden con los que definimos previamente en los stubs.

Si por razones de compatibilidad se requiere que el método Execute se siga exponiendo, se deberá crear expresamente un stub con el nombre Execute.

Continuando con el wizard, presionamos Import y verificamos que se creó el objeto externo dentro del folder que ya teníamos. Si lo abrimos verificamos que tiene los 2 métodos que definimos.

De esta forma comprobamos la flexibilidad de incluir varios métodos en un mismo servicio, cosa bastante útil para el que consume nuestro web service SOAP.

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

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