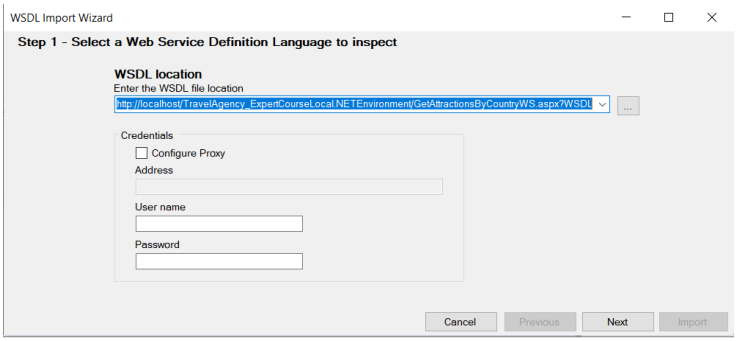
Web Services. Aspectos avanzados.

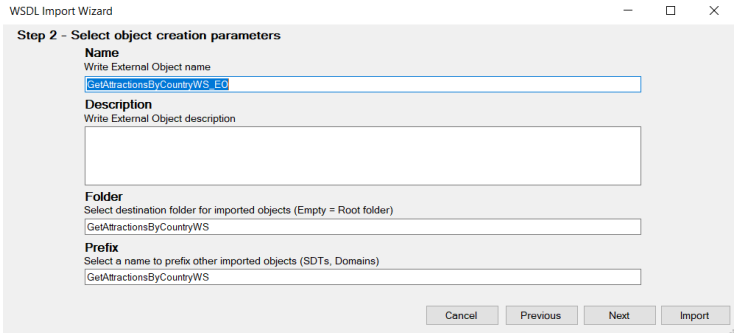
Consumiendo servicios SOAP con GeneXus

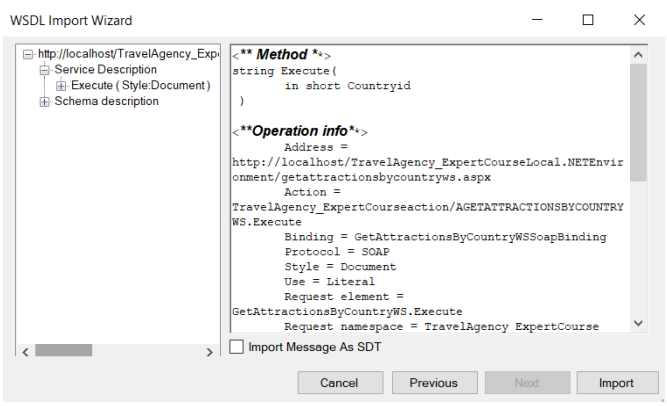
GeneXus™

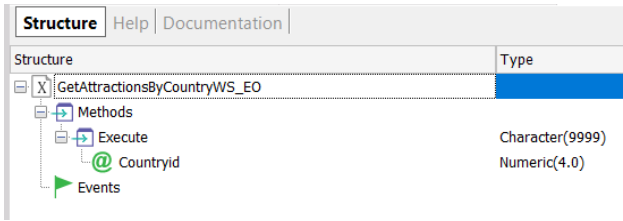
Veamos ahora cómo podemos probar un servicio SOAP publicado con GeneXus, desde GeneXus mismo, importándolo como objeto externo.
Luego nos concentraremos en algunos conceptos más avanzados que nos flexibilizan el consumo de este tipo de servicios.

Testing the procedure published as SOAP web service with GeneXus









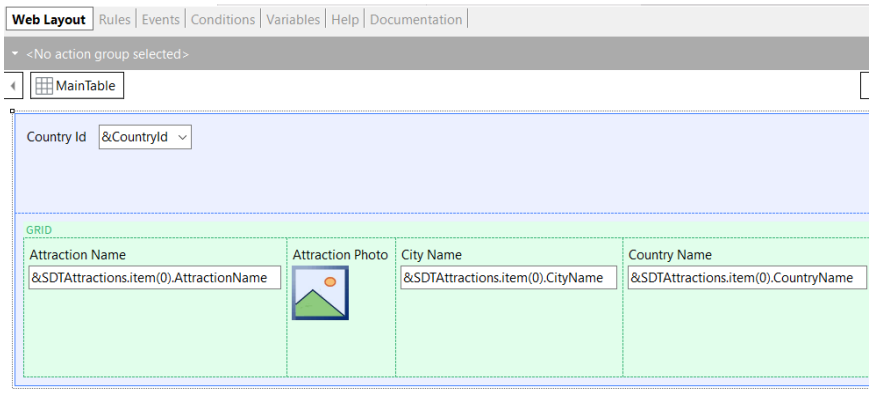
Para importar el servicio SOAP que publicamos, vamos a Tools /Application integration / WSDL Import y escribimos el mismo WSDL que probamos antes. [http://localhost/TravelAgency_ExpertCourseNETLocal/GetAttractionsByCountryWS.aspx?WSDL]

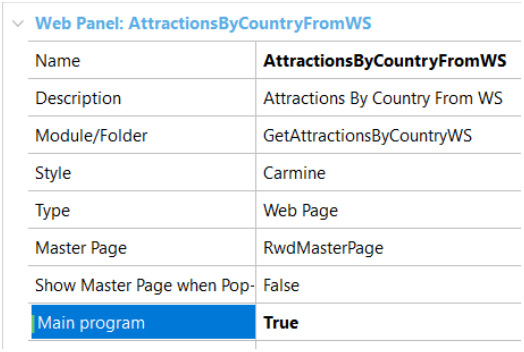
En el paso 2 del wizard escribimos el nombre de un folder de destino y presionamos Next.

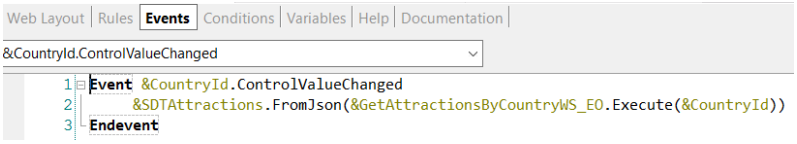
En el paso 3, vemos la estructura del servicio con dos nodos. Si abrimos el nodo Service Description vemos que hay un único método llamado Execute, y que si hacemos clic sobre él, vemos en la pantalla de la derecha que recibe por parámetro al identificador del país CountryId.

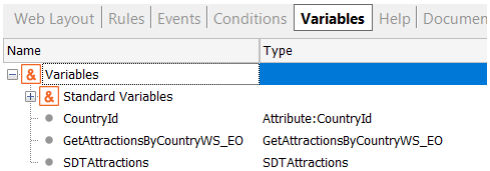
Si vamos al KB Explorer vemos que se creó el folder que definimos y si lo abrimos vemos que aparece el objeto externo. Si abrimos el objeto, podemos ver que tiene definido un único método llamado Execute y que recibe por parámetro el CountryId, tal como lo programamos en el procedimiento.

Testing the procedure published as SOAP web service with GeneXus







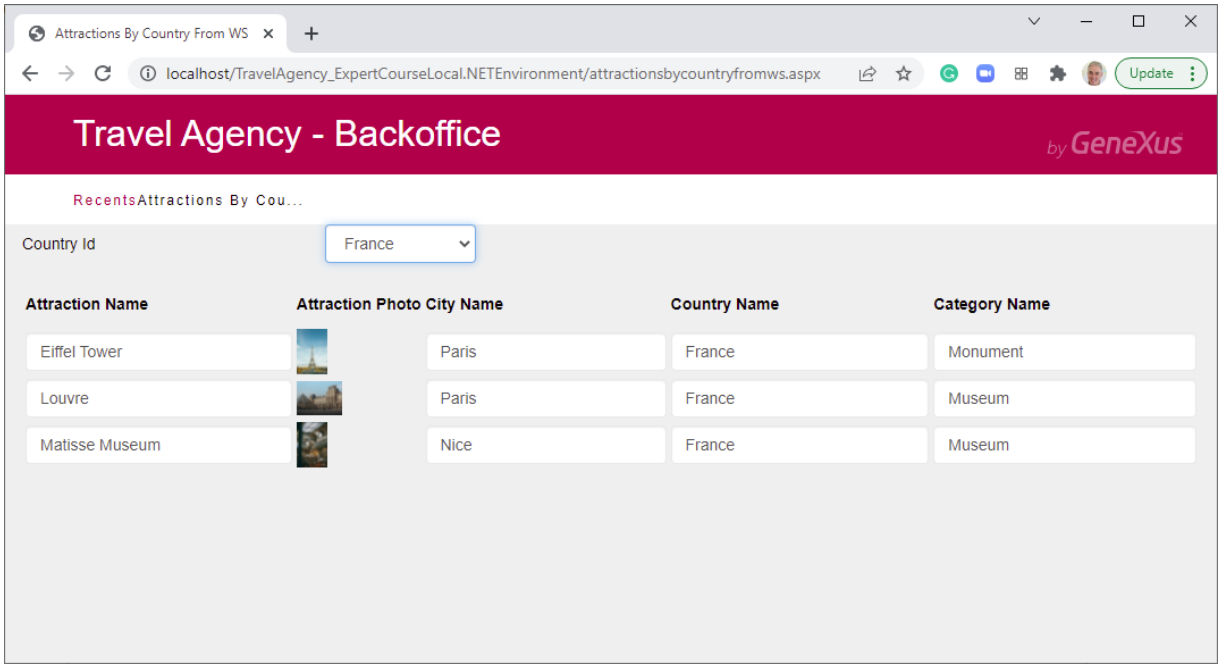


Ahora, para probar el web service que publicamos, creamos un web panel de nombre AttractionsByCountryFromWS que seteamos como main y en su sección variables definimos la variable &CountryId, y una variable SDTAttractions que nos queda basada en el SDT colección del mismo nombre. También creamos **una variable con el mismo nombre del objeto externo para que nos quede de ese tipo.**

La variable &CountryId al form la definimos como Dynamic Combo con la propiedad Item Descriptions en CountryName y la propiedad Empty Item en True.

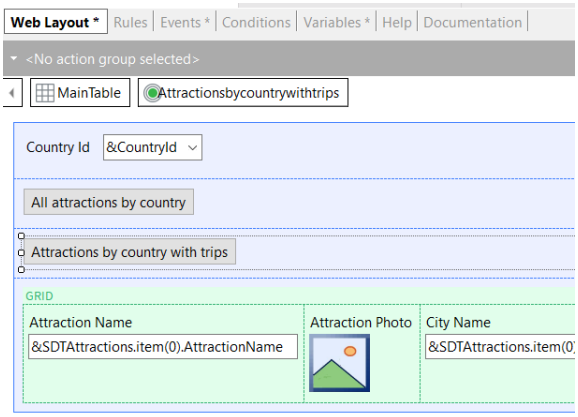
En los eventos, programamos el evento ControlValueChanged del Dynamic combobox, de forma que cuando elijamos un país, se dispare el evento. Dentro del evento escribimos la invocación al web service usando la variable &GetAttractionsByCountryWS, punto, Execute y pasando como parámetro el identificador de país.
Y esta invocación la ponemos dentro de los paréntesis del método FromJson de la variable &SDTAttractions, que contendrá la colección de atracciones devueltas por nuestro web service.

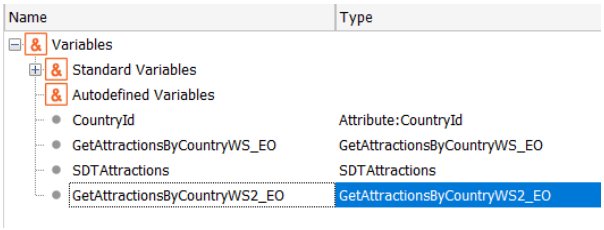
Testing the procedure published as SOAP web service with GeneXus

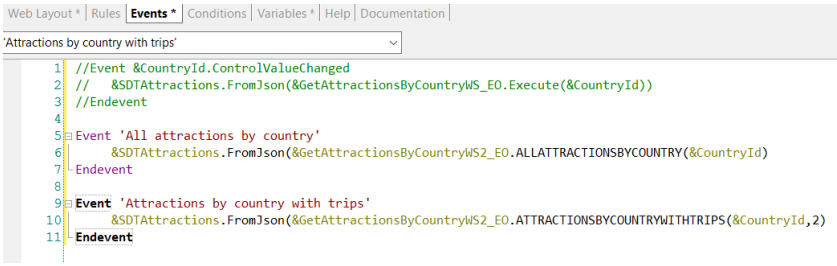


Si ejecutamos el web panel, vemos que al elegir un país, podemos ver todas las atracciones turísticas de dicho país, devueltas por nuestro web service, quien accedió a la base de datos para obtenerlas.

Web service SOAP with more than one method







Probemos también el servicio que publicamos con dos métodos.

Para invocar al servicio, hicimos una copia del web panel que habíamos creado. Agregamos dos botones con los captions “All attractions by country” y “Attractions by country with trips”.

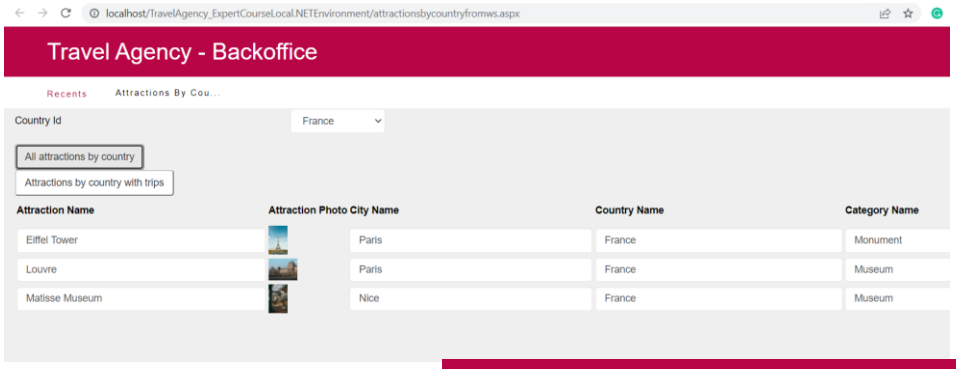
Creamos la variable con el mismo nombre que el objeto externo y comentamos el evento ControlValueChanged.

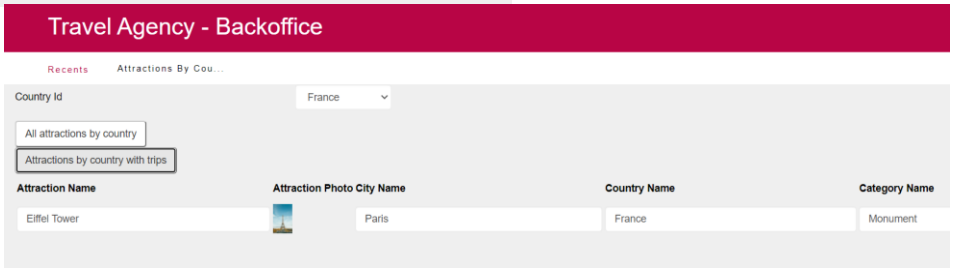
En el evento del primer botón realizamos una invocación similar a la que teníamos, usando la nueva variable e invocando al método AllAttractionsByCountry, pasándole como parámetro el CountryId.

Hacemos lo mismo para el evento del segundo botón, invocamos a AttractionsByCountryWithTrips, pasándole como parámetro el CounryId y el valor 2, para que recupere solamente las atracciones del país seleccionado que tengan al menos 2 viajes.

Ejecutamos el web panel ya que es main.

Web service SOAP with more than one method



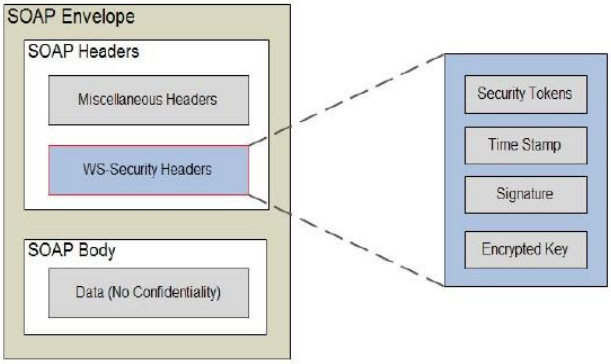


Seleccionamos al país France, presionamos el primer botón y vemos que nos trae todas las atracciones de Francia y si presionamos el segundo botón vemos que nos trae a la atracción Torre Eiffel y al Museo Matisse, que son las únicas que tienen más de un viaje regjstrado.

Si abrimos el Work With Trips, verificamos que solamente estas atracciones de Francia, tienen más de un viaje registrado.

Consuming secure SOAP web services with GeneXus: WS-SECURITY

WS-SECURITY (WSS)



Typical Message Flow with WS-Security [WSS2]

Un tema necesario a considerar en los servicios web, es el tema de la seguridad de la información transmitida.

En general cuando se requiere que la estructura de los datos y las operaciones no cambien y se requiere de autenticación o sesiones, se utiliza el protocolo SOAP.

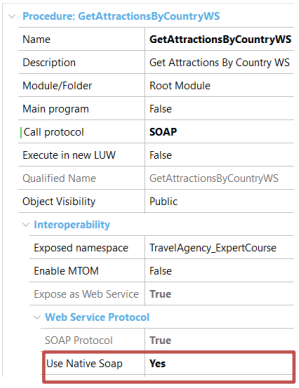
Para agregar seguridad al mensaje SOAP muchos proveedores de servicios utilizan el protocolo WSS (WS-Security), que agrega una firma digital, cifrado y métodos de autenticación, que aseguran la integridad del mensaje (es decir que el mensaje no fue modificado durante la transmisión), su confidencialidad (que el mensaje no sea visto por terceros) y que la autenticación sea segura (que no se violen las credenciales del mensaje).

Esta seguridad se agrega a nivel del servidor web, mediante herramientas de terceros específicas.

El mecanismo se basa en modificar los mensajes SOAP, se incluyen cabeceras SOAP específicas y se modifica el cuerpo SOAP con información que se utiliza para asegurar la integridad y confidencialidad. Estas modificaciones implican:

- Agregar un token de seguridad que especifique las credenciales del autor del mensaje.
- Agregar una descripción de cómo se firma el mensaje (clave utilizada, algoritmo, qué parte del mensaje se firma) y su firma.
- Agregar una descripción de cómo se encripta el mensaje (clave de encriptación, algoritmo y qué parte del mensaje está encriptada).

Consuming secure SOAP web services with GeneXus: WS-SECURITY



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	**Yes**

GeneXus WSSecurity Data Type:

https://wiki.genexus.com/commwiki/servlet/wiki?44552

GeneXus WSSecurity Data Type properties:

- WSSecurity
- WSSignature
- WSEncryption
- WSSecurityKeyStore

EXAMPLE:

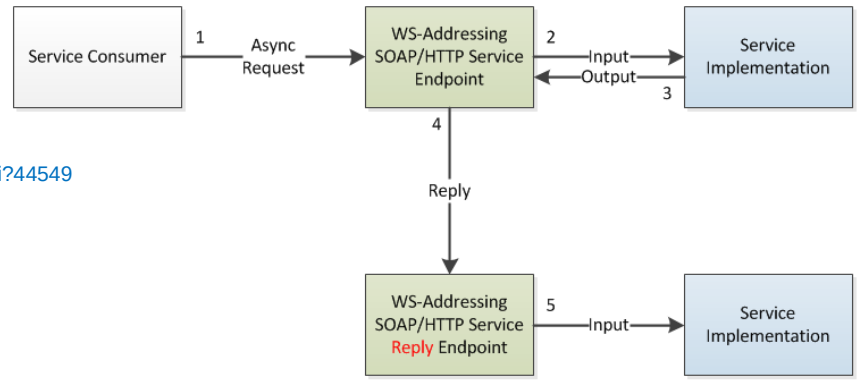
```
//Encryption
&WsSecurityKeyStore.Password = "prueba123"
&WsSecurityKeyStore.Type = WSSecurityKeyStore.JKS
&WsSecurityKeyStore.Source = "C:\temp\keystoreprueba.jks"

&wsEncryption.Alias = "epagos"
&wsEncryption.keyIdentifierType = WsSecurity.BINARY_SECURITY_TOKEN
&wsEncryption.Keystore = &WsSecurityKeyStore

&wssecurity.Encryption = &wsEncryption

&wssecurity.ExpirationTimeout = 5

//Signature
&wssecurity.Signature.Alias = "alias1"
&wssecurity.Signature.keyIdentifierType = WsSecurity.BINARY_SECURITY_TOKEN
&wssecurity.Signature.Keystore.Password = "PasswordAlias1"
&wssecurity.Signature.Keystore.Type = WSSecurityKeyStore.JKS
&wssecurity.Signature.Keystore.Source = "C:\temp\prueba2.jks"

&wssecurity.Signature.CanonicalizationAlgorithm = "CanonicalizationMethod.EXCLUSIVE"
&wssecurity.Signature.SignatureAlgorithm = "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"
&wssecurity.Signature.Digest = "DigestMethod.SHA256"

&location.WSSecurity = &wssecurity
```

Con GeneXus no publicamos web services con WS-Security, pero sí podemos consumir servicios de terceros que cumplan dicho protocolo, usando funciones nativas que nos proveen .NET o Java, que habilitamos con la propiedad Use Native Soap.

El protocolo SOAP utiliza el estándar XML para la definición de los mensajes de request y response y la propiedad Use Native Soap determina la forma en que se van a generar los XMLs. Si se pone en Yes, para la serialización del XML se utilizan funcionalidades que provee el lenguaje de la plataforma, es decir que serán instrucciones del framework .Net o del lenguaje Java los que resuelven la generación del XML a bajo nivel. Si está en No (que es el valor por defecto), GeneXus es quien se encargará de generar los XMLs que usa el servicio.

GeneXus además nos provee de tipos de datos especiales para consumir web services WS-Security y también constantes predefinidas que nos facilitan el uso de estos tipos de datos.

Tracking and routing SOAP web services with GeneXus: WS-ADDRESSING



GeneXus WSAddressing Data Type:

https://wiki.genexus.com/commwiki/servlet/wiki?44549

WSAddressing Properties

To	Character
Action	Character
MessageID	Character
From	WSAddressingEndPoint
ReplyTo	WSAddressingEndPoint
FaultTo	WSAddressingEndPoint

WSAddressingEndPoint Properties

Address	Character
PortType	Character
ServiceName	Character
Properties	Character
Parameters	Character

EXAMPLE:

```
&location = getLocation("EObjectName")
&wsaddressing.Action = "urn:antel:mdm:system:epagos:b2b:comercio:iniciarSolicitud"
&wsaddressing.MessageID = "uuid:e5403230-9bab-4152-a2ba-e4e47165f135"
&wsaddressing.To = "urn:antel:mdm:system:epagos"
&wsaddressing.From = &wsaddressingendpoint
&wsaddressing.ReplyTo = &otherwsadressingendpoint
&location.WSAddressing = &wsaddressing
```

En ciertas aplicaciones, se hace de gran utilidad que quien consuma un servicio publicado por nosotros, pueda enviar una respuesta al servidor que publica nuestro servicio.

GeneXus nos permite rastrear y enrutar web services SOAP mediante WS-Addressing, habilitando también la propiedad Use Native Soap.

WS-Addressing es un protocolo de la World Wide Web Consortium (W3C) que especifica cómo un consumidor de servicios puede indicar el punto final al que el servicio debe enviar su respuesta (y/o enviar fallas SOAP), en una invocación.

El cliente envía una solicitud SOAP al servicio y la infraestructura del servicio, sabe que debe tratar la solicitud de forma asíncrona (para que el cliente no espere la respuesta). La implementación del servicio genera una respuesta y la devuelve al componente de servicio que interpreta los encabezados de WS-Addressing y luego reenvía automáticamente el sobre SOAP de respuesta al URI mencionado en el encabezado ReplyTo.

Esto se hace mediante dos estructuras: Headers y EndPoint References.

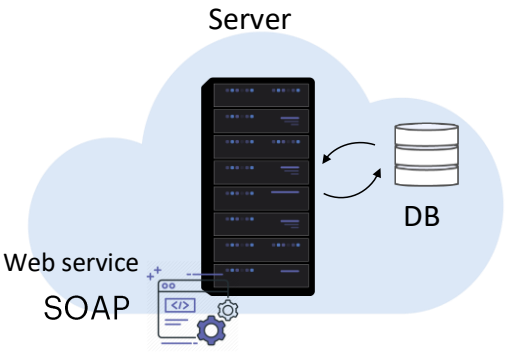
El poder de WS-Addressing es que el consumidor del servicio puede especificar a qué punto final se debe enviar la respuesta. No está codificado en la implementación del servicio, lo especifica el consumidor.

En el wiki encontraremos información del tipo de datos WSAddressing usado por GeneXus.

Location of a SOAP web service

LOCATION

- Host
- BaseUrl
- Port
- Secure
- Resource Name
- AuthenticationUser
- AutheticationPassword
- Timeout
- CancelOnError



Veamos ahora qué es el location de un web service.

Los web services se ejecutan en un servidor web, por lo que cada web service es identificado por una URL, de un cierto host y en un puerto determinado. También tiene otras características cuando se ejecuta, como por ejemplo la seguridad que usa, el método de autenticación, las credenciales, etc.

Estos valores se asignan por defecto por GeneXus cuando publicamos un web service y en caso de que el web service sea de un tercero, el que provee el servicio asigna estas características.

A este conjunto de propiedades que determinan dónde y cómo se ejecuta un servicio remoto, se le denomina “location”.

Cuando invocamos un web service del tipo SOAP, es posible cambiar su location y asignar otros valores distintos a los establecidos por defecto. Esto es especialmente útil cuando por ejemplo estamos ejecutando un servicio en un servidor de desarrollo y lo queremos ejecutar en un servidor de producción. Si bien el código que se ejecuta es el mismo, al cambiarle la dirección del host y otras propiedades, pasa a ser considerado otro servicio diferente del primero.

Podemos cambiar la location de un web service, en tiempo de generación (para cambiar los valores por defecto asignados por GeneXus) o en tiempo de ejecución, por código, cuando consumimos el web service.

Location of a SOAP web service

GetAttractionsByCountryWS

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

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

GetAttractionsByCountryWS

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

Subroutines

```
1  For each Attraction
2        Where CountryId = &CountryId
3        &OneAttraction.AttractionName = AttractionName
4        &OneAttraction.AttractionPhoto = AttractionPhoto
5        &OneAttraction.CategoryName = CategoryName
6        &OneAttraction.CityName = CityName
7        &OneAttraction.CountryName = CountryName
8        &SDTAttractions.Add(&OneAttraction)
9        &OneAttraction = New()
10 Endfor
11 &Attractions = &SDTAttractions.ToJson()
```

AttractionsByCountryFromWS *

Web Layout | Rules | **Events *** | Conditions | Variables | Help | Documentation

Events

```
1  Event &CountryId.ControlValueChanged
2        //Service behaviour setting
3        &Location = GetLocation("GetAttractionsByCountryWS_EO")
4        &Location.Host = "www.servername.com"
5        &Location.BaseUrl = "/base URL/"
6        &Location.Port = 123456
7        &Location.Secure = 1 or 2
8        &Location.ResourceName = "ServiceName"
9        &Location.CancelOnError = 2 //Do not stop execution
10
11       //Web service invocation
12       &SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))
13       if GetSOAPErr() > 0
14            msg("Error: " + GetSOAPErrMsg())
15       endif
16 Endevent
```

Vamos a volver al ejemplo del procedimiento GetAttractionsByCountryWS que publicamos como web service SOAP para obtener datos de atracciones por país y que luego consumimos desde el web panel AttractionsByCountryFromWS, para probarlo.

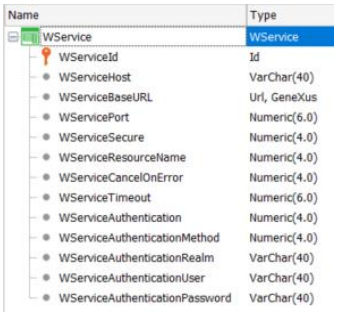
En los eventos del webpanel donde mostramos los datos devueltos por el servicio, podemos obtener el location del web service invocado asignando una variable Location, del tipo Location, con lo devuelto por el método GetLocation al que le pasamos como parámetro el nombre del External Object.

Esto nos permite asignar distintos valores a las propiedades del location, por ejemplo cambiar el host, la URL base, el puerto, el nombre del recurso y otros datos del servicio. Esto es de gran utilidad cuando tenemos un servicio ejecutando en un servidor y luego queremos que el servicio se ejecute en otro servidor, por ejemplo cuando pasamos del servidor de desarrollo al servidor de producción, o cuando queremos reutilizar un servicio en otra aplicación y queremos que se ejecute en otro server.

Observemos que después de ejecutar el web service, utilizamos los métodos GetSoapErr() y GetSoapErrMsg para obtener información del resultado de la ejecución. Profundizaremos en estos conceptos más adelante.

Customizing the location of a SOAP web service

Assigning Location variable from database



Name	Type
WService	WService
WServiceId	Id
WServiceHost	VarChar(40)
WServiceBaseURL	Url, GeneXus
WServicePort	Numeric(6.0)
WServiceSecure	Numeric(4.0)
WServiceResourceName	Numeric(4.0)
WServiceCancelOnError	Numeric(4.0)
WServiceTimeout	Numeric(6.0)
WServiceAuthentication	Numeric(4.0)
WServiceAuthenticationMethod	Numeric(4.0)
WServiceAuthenticationRealm	VarChar(40)
WServiceAuthenticationUser	VarChar(40)
WServiceAuthenticationPassword	VarChar(40)

```
Event &CountryId.ControlValueChanged
    //Service behaviour setting
    &Location = GetLocation("GetAttractionsByCountryWS_EO")
    For each WService
        Where WServiceId = &WSServiceId
        &Location.Host = WServiceHost
        &Location.BaseUrl = WServiceBaseURL
        &Location.Port =WServicePort
        &Location.Secure = WServiceSecure
        &Location.ResourceName = WServiceResourceName
        &Location.CancelOnError = WServiceCancelOnError
        Exit
    Endfor
    //Web service invocation
    &SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))
    if GetSOAPErr() > 0
        msg("Error: " + GetSOAPErrMsg())
    endif
Endevent
```

Location.xml

```
<GXLocations>
  <GXLocation name="GetAttractionsByCountryWS_EO"> // Name of the External Object
    <Common>
      <Host>"www.servername.com"</Host>   // Don't include protocol (i.e. HTTP/HTTPS)
      <Port>443</Port> // Port number
      <BaseUrl>/services/</BaseUrl>   // Start and end with a bar: /baseurl/
      <Secure>1</Secure> // 1 = HTTPS, 0 = HTTP
      <Proxyserverhost/>
      <Proxyserverport/>
      <Timeout/>
    </Common>
    <HTTP>
      <Authentication>
        <Authenticationmethod/>
        <Authenticationuser/>
        <Authenticationpassword/>
        <Authenticationrealm/>
      </Authentication>
      <Proxyauthentication>
        <Proxyauthenticationmethod/>
        <Proxyauthenticationrealm/>
        <Proxyauthenticationuser/>
        <Proxyauthenticationpassword/>
      </Proxyauthentication>s
    </HTTP>
  </GXLocation>
</GXLocations>
```

Los datos que asignamos al location de un servicio, pueden estar parametrizados en la base de datos. En el ejemplo, vemos que creamos una transacción WService y luego con un comando For Each accedemos a la tabla que contiene los valores de los atributos para asignar las propiedades del location.

Además de definir una variable del tipo de datos Location y cambiar los valores de sus propiedades por código como vimos recién, otra forma de asignar los valores de un location es utilizando un archivo Location.xml que colocamos en la carpeta web de nuestro target environment, por ejemplo la carpeta Web en un modelo .NET o web-inf en Java.

Cuando GeneXus compila la aplicación, si encuentra un archivo llamado location.xml en la carpeta antes mencionada, lo lee y asigna los valores del location dinámicamente en tiempo de ejecución.

El formato del achivo location.xml es el que se muestra en pantalla.

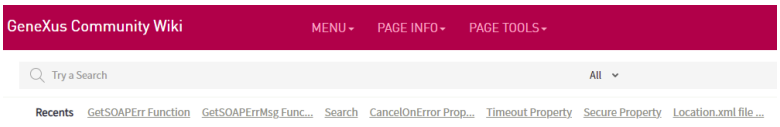
En GXLocation Name asignamos el nombre del objeto externo asociado a nuestro web service.
Host es el nombre del servidor que alojará al servicio, sin incluir el http o https, es decir por ejemplo solamente www.servername.com
El puerto es un valor numérico que depende del tipo de servidor, la url base se recomienda que se incluya entre barras, por ejemplo /services/.
Secure con el valor 1 indica que se utilizará un servidor seguro, es decir con protocolo HTTPS y el valor 0 para HTTP.
Proxyserverhost y Proxyserverport se utilizan en el caso de que haya un servidor

Proxy entre el cliente y el servidor que provee el servicio.
Timeout se asigna con el máximo tiempo en segundos que el sistema debe esperar por una respuesta a ser enviada al servidor luego de cada request. Si se asigna un valor 0 significa que el tiempo de espera es indefinido.
Luego, en caso de requerirse, pueden definirse las características de autenticación de la comunicación HTTP.

Las propiedades asignadas en tiempo de ejecución a una variable del tipo de datos Location, tendrán preferencia sobre las asignadas en tiempo de ejecución mediante un archivo location.xml, y estas últimas, a su vez, tendrán preferencia sobre las asignadas en tiempo de generación. Esto permite más dinamismo en la configuración de las ubicaciones.

Errors handling of a SOAP web service invocation

```
Event &CountryId.ControlValueChanged
    //Service behaviour setting
    &Location = GetLocation("GetAttractionsByCountryWS_EO")
    &Location.CancelOnError = 2 //No stop execution on error

    //Web service invocation
    &SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))

    //Error handling
    &error = GetSOAPErr()
    If &error <> 0 Or null(&location.Host)
        Do Case
        Case &error = -20007
            &errorMsg = "Unknown error to set a web service, unknown location:" + &Location.ResourceName + newline() + GetSOAPErrMsg()
        Case &error > 0
            &errorMsg = "Unknown error to set a web service:" + &Location.ResourceName + newline() + GetSOAPErrMsg()
        Otherwise
            &errorMsg = "Error from unknown host to set a web service:" + &Location.ResourceName
        EndCase
    EndIf
Endevent
```



Error Codes and Messages for Location Data Type

Cuando invocamos un servicio SOAP y asignamos valores a una variable location, podemos asignar un valor a la propiedad CancelOnError, lo que nos permite obtener el resultado de la operación y manejar los errores. Aquí tenemos un ejemplo de uso típico.

Dependiendo del valor asignado a la propiedad CancelOnError, será el comportamiento del servicio.

Por ejemplo, si asignamos el valor 0, el programa que invoca al web service cancelará siempre la ejecución al terminarse la invocación. Este es el valor por defecto.
Si asignamos el valor 1, el programa llamador cancelará la ejecución en caso de producirse un error.
Si asignamos el valor 2, el programa llamador NO cancelará la ejecución en caso de producirse un error y para obtener información del error podemos usar las funciones GetSOAPErr() y GetSOAPErrMsg().

La función GetSOAPErr() retorna un valor numérico correspondiente al código de error de la última operación SOAP.
Si la función retorna 0 no se produjo ningún error. En caso contrario, el código depende del tipo de error y este varía si el error se produjo en el cliente o en el servidor.

La función GetSOAPErrMsg() nos devuelve una descripción del error de la última operación SOAP, lo que nos permite dar un mensaje más amigable al usuario.

En el artículo del Wiki “Error Codes and Messages for Location Data Type” encontrará una tabla de los códigos de errores posibles, tanto del cliente como del servidor.

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