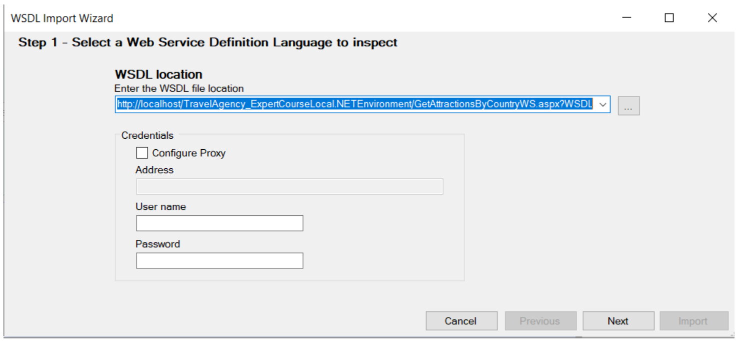
Web Services. Advanced topics

Consuming SOAP services with GeneXus

Now let's see how we can test a SOAP service published with GeneXus, from GeneXus itself, importing it as an external object.
Then we will focus on some more advanced concepts that make it more flexible to consume this type of services.

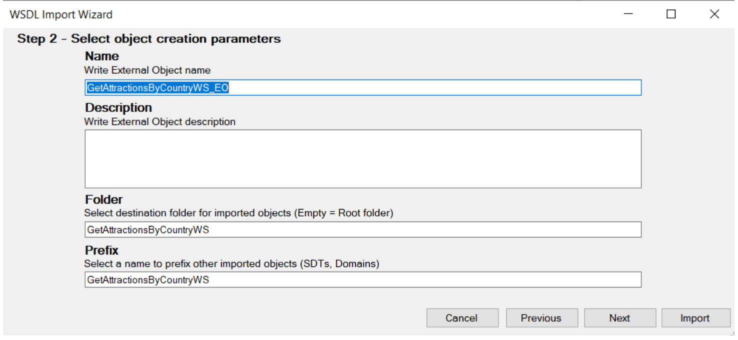
Testing the procedure published as a SOAP web service with GeneXus



Step 1 - Select a Web Service Definition Language to inspect

WSDL location
Enter the WSDL file location
http://localhost/TravelAgency_ExpertCourseLocal.NETEnvironment/GetAttractionsByCountryWS.aspx?WSDL

Credentials
☐ Configure Proxy
Address
User name
Password



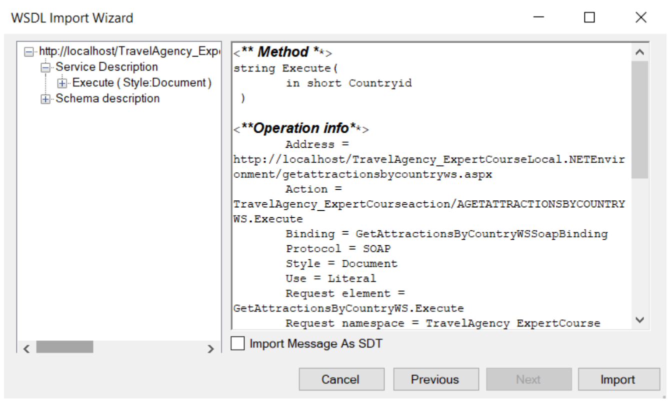
Step 2 - Select object creation parameters

Name
Write External Object name
GetAttractionsByCountryWS_EO

Description
Write External Object description

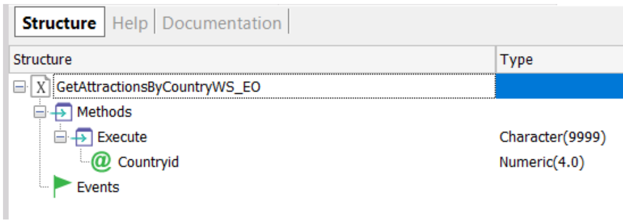
Folder
Select destination folder for imported objects (Empty = Root folder)
GetAttractionsByCountryWS

Prefix
Select a name to prefix other imported objects (SDTs, Domains)
GetAttractionsByCountryWS



```
<** Method **>
string Execute(
        in short Countryid
 )

<**Operation info**>
        Address =
http://localhost/TravelAgency_ExpertCourseLocal.NETEnvir
onment/getattractionsbycountryws.aspx
        Action =
TravelAgency_ExpertCourseaction/AGETATTRACTIONSBYCOUNTRY
WS.Execute
        Binding = GetAttractionsByCountryWSSoapBinding
        Protocol = SOAP
        Style = Document
        Use = Literal
        Request element =
GetAttractionsByCountryWS.Execute
        Request namespace = TravelAgency ExpertCourse
```

☐ Import Message As SDT



Structure	Type
GetAttractionsByCountryWS_EO	
Methods	
Execute	Character(9999)
Countryid	Numeric(4.0)
Events	

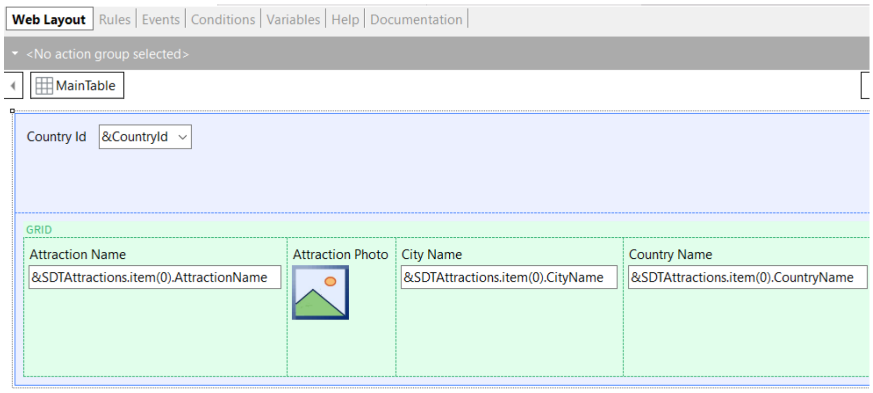
To import the SOAP service we published, we go to Tools /Application integration / WSDL Import and write the same WSDL we tested before. [http://localhost/TravelAgency_ExpertCourseNETLocal/GetAttractionsByCountry WS.aspx?WSDL]

In step 2 of the wizard, we type the name of a destination folder and press Next.

In step 3, we see the structure of the service with two nodes. If we open the Service Description node, we find a single method called Execute, and if we click on it, we see in the screen on the right that it receives the country identifier CountryId as a parameter.

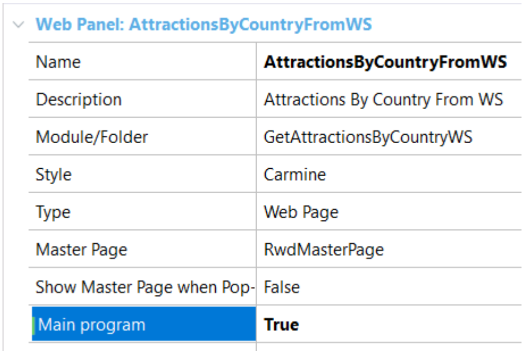
By going to the KB Explorer, we can see that the folder we defined was created and if we open it, the external object appears. If we open the object, we can see that it has only one method defined called Execute and that it receives by parameter the CountryId, as we programmed it in the procedure.

Testing the procedure published as a SOAP web service with GeneXus


Web Layout | Rules | Events | Conditions | Variables | Help | Documentation
<No action group selected>
MainTable
GRID

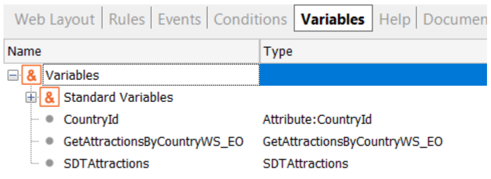
Country Id &CountryId

Attraction Name	Attraction Photo	City Name	Country Name
&SDTAttractions.item(0).AttractionName		&SDTAttractions.item(0).CityName	&SDTAttractions.item(0).CountryName

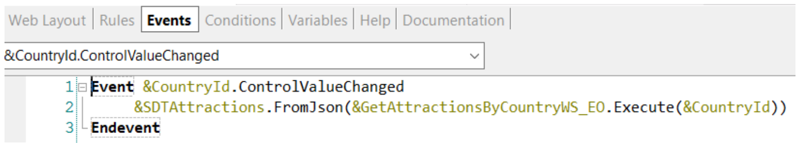


Web Panel: AttractionsByCountryFromWS

Name	AttractionsByCountryFromWS
Description	Attractions By Country From WS
Module/Folder	GetAttractionsByCountryWS
Style	Carmine
Type	Web Page
Master Page	RwdMasterPage
Show Master Page when Pop-	False
Main program	True


Web Layout | Rules | Events | Conditions | Variables | Help | Documen

Name	Type
Variables	
Standard Variables	
CountryId	Attribute:CountryId
GetAttractionsByCountryWS_EO	GetAttractionsByCountryWS_EO
SDTAttractions	SDTAttractions


Web Layout | Rules | Events | Conditions | Variables | Help | Documentation
&CountryId.ControlValueChanged

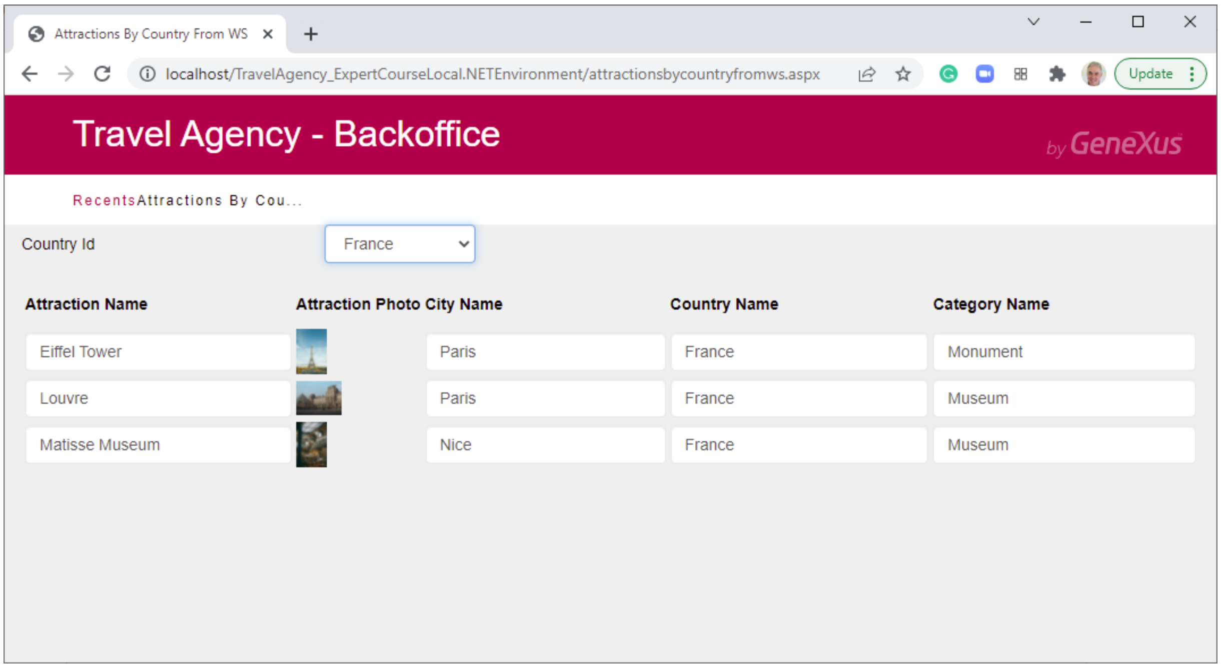
```
Event &CountryId.ControlValueChanged
	&SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))
Endevent
```

Now, to test the web service that we published, we create a web panel named AttractionsByCountryFromWS and set it as main. In its variables section, we define the &CountryId variable, and an SDTAttractions variable that is based on the SDT collection of the same name. We also create **a variable with the same name as that of the external object so that it is of that type.**

We define the &CountryId variable for the form as a Dynamic Combo with the Item Descriptions property set to CountryName and the Empty Item property set to True.

In the events, we program the ControlValueChanged event of the Dynamic combo box, so that when we choose a country, the event is fired. Inside the event we write the invocation to the web service using the variable &GetAttractionsByCountryWS, period, Execute and passing as parameter the country identifier.
We place this invocation inside the brackets of the FromJson method of the variable &SDTAttractions, which will contain the collection of attractions returned by our web service.

Testing the procedure published as a SOAP web service with GeneXus

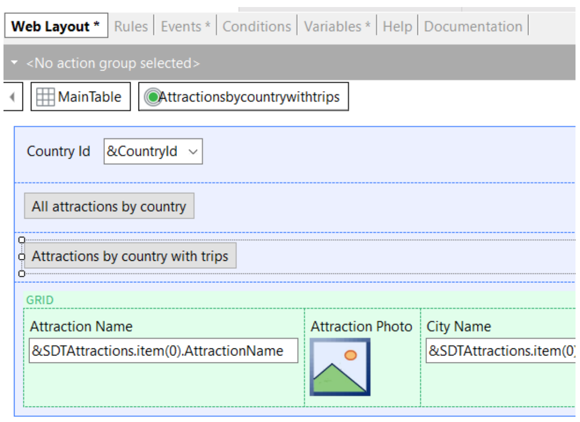

Attractions By Country From WS
localhost/TravelAgency_ExpertCourseLocal.NETEnvironment/attractionsbycountryfromws.aspx
Update
Travel Agency - Backoffice
by GeneXus
RecentsAttractions By Cou...

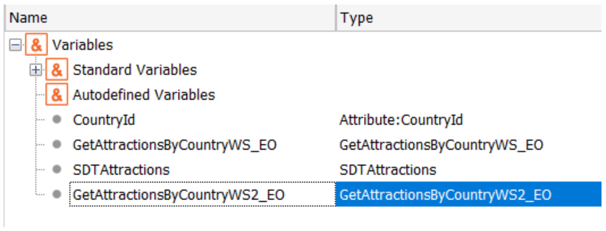
Country Id: France

Attraction Name	Attraction Photo	City Name	Country Name	Category Name
Eiffel Tower		Paris	France	Monument
Louvre		Paris	France	Museum
Matisse Museum		Nice	France	Museum

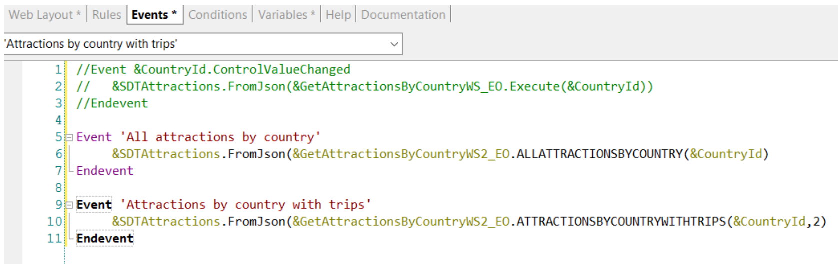
If we run the web panel, we see that by choosing a country, all the tourist attractions of that country are displayed, returned by our web service that accessed the database to obtain them.

SOAP web service with more than one method





Name	Type
Variables	
Standard Variables	
Autodefined Variables	
CountryId	Attribute:CountryId
GetAttractionsByCountryWS_EO	GetAttractionsByCountryWS_EO
SDTAttractions	SDTAttractions
GetAttractionsByCountryWS2_EO	GetAttractionsByCountryWS2_EO



```
//Event &CountryId.ControlValueChanged
//    &SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))
//Endevent

Event 'All attractions by country'
     &SDTAttractions.FromJson(&GetAttractionsByCountryWS2_EO.ALLATTRACTIONSBYCOUNTRY(&CountryId)
Endevent

Event 'Attractions by country with trips'
     &SDTAttractions.FromJson(&GetAttractionsByCountryWS2_EO.ATTRACTIONSBYCOUNTRYWITHTRIPS(&CountryId,2)
Endevent
```

Let's also test the service we published with two methods.

To invoke the service, we made a copy of the web panel we had created. We added two buttons with the captions “All attractions by country” and “Attractions by country with trips.”

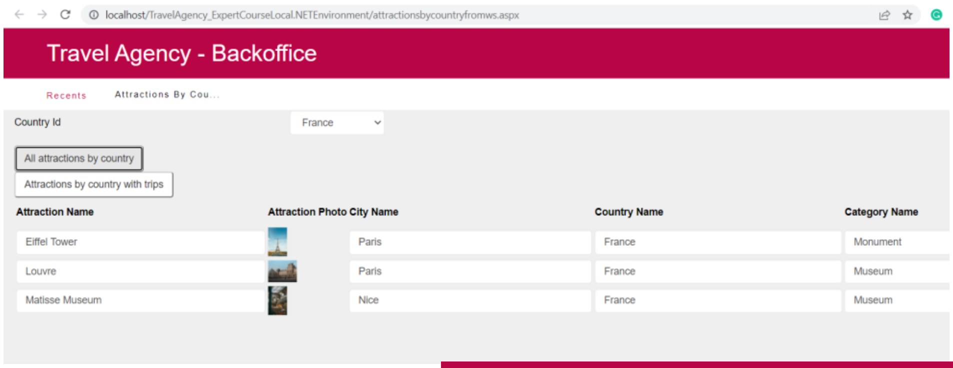
We created the variable with the same name as the external object and commented the ControlValueChanged event.

In the event of the first button, we make an invocation similar to the one we had, using the new variable and invoking the AllAttractionsByCountry method, passing it as parameter the CountryId.

We do the same for the event of the second button; that is, we invoke AttractionsByCountryWithTrips, passing it as parameter the CountryId and the value 2, so that it retrieves only the attractions of the selected country that have at least 2 trips.

We execute the web panel because it is main.

SOAP web service with more than one method


localhost/TravelAgency_ExpertCourseLocal.NETEnvironment/attractionsbycountryfromws.aspx

Travel Agency - Backoffice

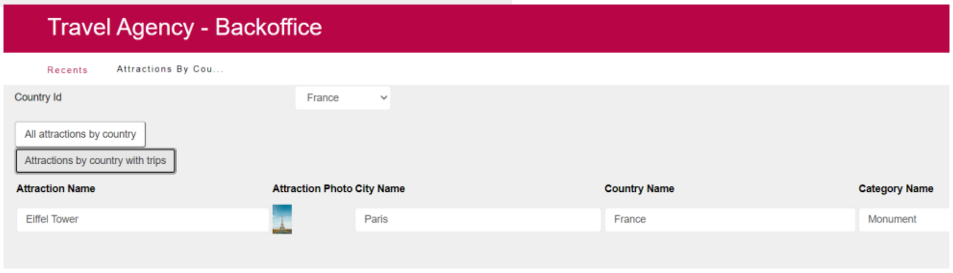
Recents   Attractions By Cou...

Country Id: France

All attractions by country

Attractions by country with trips

Attraction Name	Attraction Photo	City Name	Country Name	Category Name
Eiffel Tower		Paris	France	Monument
Louvre		Paris	France	Museum
Matisse Museum		Nice	France	Museum


Travel Agency - Backoffice

Recents   Attractions By Cou...

Country Id: France

All attractions by country

Attractions by country with trips

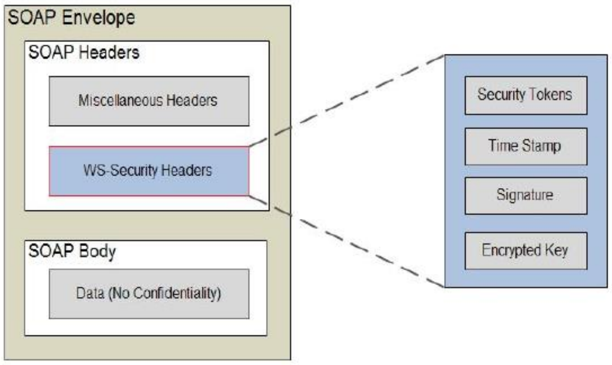
Attraction Name	Attraction Photo	City Name	Country Name	Category Name
Eiffel Tower		Paris	France	Monument

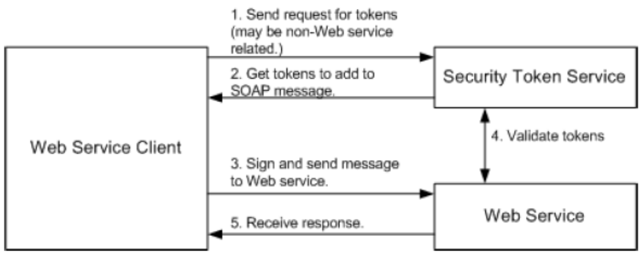
We select the country France, press the first button and see that it brings all the attractions of France. If we press the second button, it brings the Eiffel Tower and the Matisse Museum, which are the only ones that have more than one trip registered.

If we open Work With Trips, we verify that only these attractions in France have more than one trip registered.

Consuming secure SOAP web services with GeneXus: WS-SECURITY

WS-SECURITY (WSS)




1. Send request for tokens (may be non-Web service related.)
2. Get tokens to add to SOAP message.
3. Sign and send message to Web service.
4. Validate tokens
5. Receive response.
Web Service Client
Security Token Service
Web Service

Typical Message Flow with WS-Security [WSS2]

Something that should be taken into account for web services is the security of the information that is transmitted.

In general, when it is mandatory that the data structure and operations remain unchanged and authentication or sessions are required, the SOAP protocol is used.

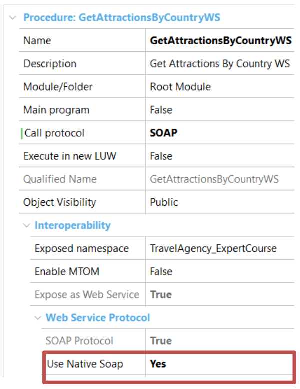
To add security to the SOAP message, many service providers use the WSS (WS-Security) protocol, which adds a digital signature, encryption, and authentication methods that ensure the integrity of the message (i.e. that the message was not modified during transmission), its confidentiality (that the message is not seen by third parties), and that the authentication is secure (that the message credentials are not compromised).

This security is added at the web server level, using specific third-party tools.

The mechanism is based on modifying SOAP messages, specific SOAP headers are included and the SOAP body is modified with information used to ensure integrity and confidentiality. These modifications involve:

- Adding a security token specifying the credentials of the message author.
- Adding a description of how the message is signed (key used, algorithm, what part of the message is signed) and its signature.
- Adding a description of how the message is encrypted (encryption key, algorithm and what part of the message is encrypted).

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:

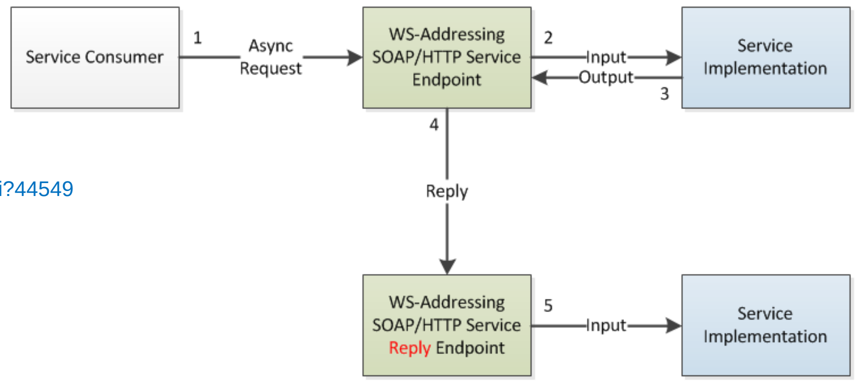
```
//Encryption
&WsSecurityKeyStore.Password = "prueba123"
&WsSecurityKeyStore.Type = WSSecurityKeyStore.JKS
&WsSecurityKeyStore.Source = "C:\temp\keystoreprueba.jks"

&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
```

With GeneXus we do not publish web services with WS-Security, but we can consume third-party services that use that protocol, using native functions provided by .NET or Java, which we enable with the Use Native Soap property.

The SOAP protocol uses the XML standard for the definition of the request and response messages, and the Use Native Soap property determines the way in which the XMLs will be generated. If set to Yes, the XML serialization will use functionalities provided by the platform language; that is, instructions from the .Net framework or Java language will be used to generate the XML at a low level. If set to No (which is the default value), GeneXus will be in charge of generating the XMLs used by the service.

GeneXus also provides specific data types to consume WS-Security web services and predefined constants that make it easier to use these data types.

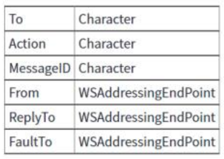
Tracking and routing SOAP web services with GeneXus: WS-ADDRESSING



GeneXus WSAddressing Data Type:

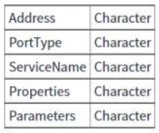
https://wiki.genexus.com/commwiki/servlet/wiki?44549

WSAddressing Properties



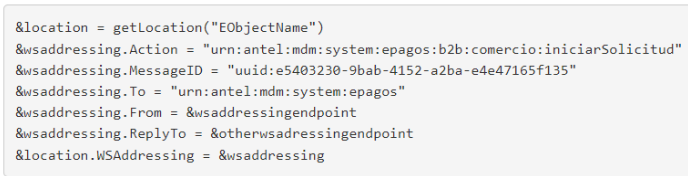
To	Character
Action	Character
MessageID	Character
From	WSAddressingEndPoint
ReplyTo	WSAddressingEndPoint
FaultTo	WSAddressingEndPoint

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
```

In certain applications, it is very useful for the consumer of a service published by us to be able to send a response to the server that publishes our service.

GeneXus allows us to track and route SOAP web services via WS-Addressing, also enabling the Use Native Soap property.

WS-Addressing is a World Wide Web Consortium (W3C) protocol that specifies how a service consumer can indicate the endpoint to which the service should send its response (and/or send SOAP failures), in an invocation.

The client sends a SOAP request to the service and the service infrastructure must handle the request asynchronously (so that the client does not have to wait for the response). The service implementation generates a response and returns it to the service component, which interprets the WS-Addressing headers and automatically forwards the SOAP response envelope to the URI mentioned in the ReplyTo header.

This is done using two structures: Headers and EndPoint References.

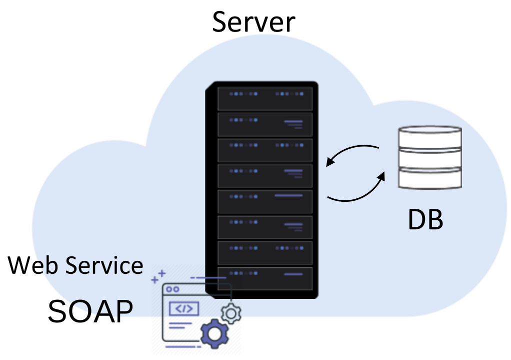
The power of WS-Addressing is that the service consumer can specify to which endpoint the response should be sent. It is not hard-coded in the service implementation; instead, it is specified by the consumer.

Information on the WSAddressing data type used by GeneXus can be found in the wiki.

Location of a SOAP web service

LOCATION

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



Now let's see what the location of a web service is.

Web services run on a web server, so each web service is identified by a URL, from a certain host and on a certain port. It also has other characteristics when it is executed, such as the security it uses, authentication method, credentials, etc.

These values are assigned by default by GeneXus when we publish a web service, and when the web service comes from a third party, the service provider assigns these characteristics.

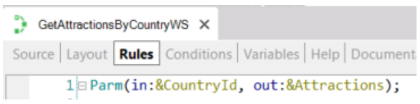
This set of properties that determine where and how a remote service is executed is called “location.”

When invoking a SOAP web service, it is possible to change its location and assign other values than the default ones.

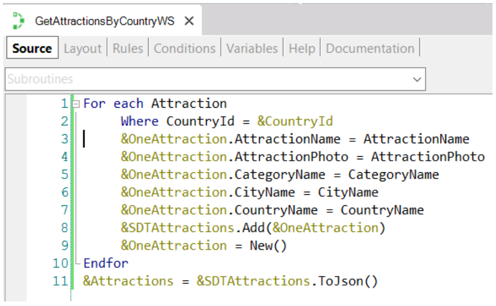
This is particularly useful when, for example, we are running a service on a development server and we want to run it on a production server. Although the code that is executed is the same, when the host address and other properties are changed, it is considered a different service from the first one.

We can change the location of a web service at generation time (to change the default values assigned by GeneXus) or at runtime, by code, when we consume the web service.

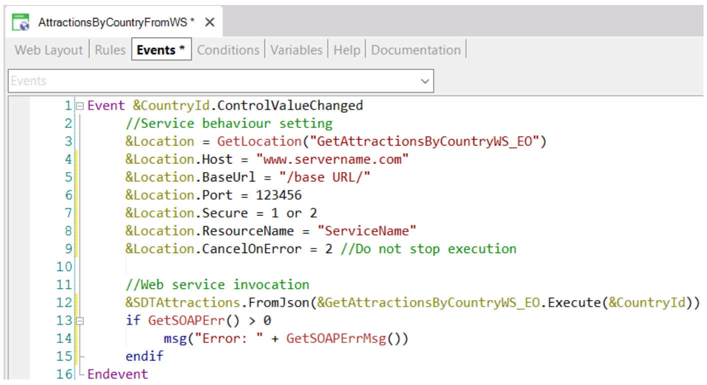
Location of a SOAP web service


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

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


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

```
For each Attraction
     Where CountryId = &CountryId
     &OneAttraction.AttractionName = AttractionName
     &OneAttraction.AttractionPhoto = AttractionPhoto
     &OneAttraction.CategoryName = CategoryName
     &OneAttraction.CityName = CityName
     &OneAttraction.CountryName = CountryName
     &SDTAttractions.Add(&OneAttraction)
     &OneAttraction = New()
Endfor
&Attractions = &SDTAttractions.ToJson()
```


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

```
Event &CountryId.ControlValueChanged
     //Service behaviour setting
     &Location = GetLocation("GetAttractionsByCountryWS_EO")
     &Location.Host = "www.servername.com"
     &Location.BaseUrl = "/base URL/"
     &Location.Port = 123456
     &Location.Secure = 1 or 2
     &Location.ResourceName = "ServiceName"
     &Location.CancelOnError = 2 //Do not stop execution

     //Web service invocation
     &SDTAttractions.FromJson(&GetAttractionsByCountryWS_EO.Execute(&CountryId))
     if GetSOAPErr() > 0
          msg("Error: " + GetSOAPErrMsg())
     endif
Endevent
```

Let's go back to the example of the GetAttractionsByCountryWS procedure that we published as a SOAP web service to obtain data of attractions by country and then consumed from the AttractionsByCountryFromWS web panel, to test it.

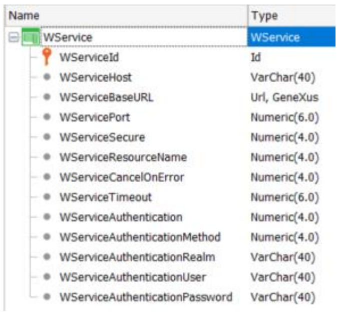
In the web panel events where we show the data returned by the service, we can obtain the location of the invoked web service by assigning a Location variable, of Location type, with the data returned by the GetLocation method to which we passed the name of the External Object as a parameter.

This allows us to assign different values to the location properties, such as changing the host, the base URL, the port, the resource name and other service data. This is very useful when we have a service running on a server and then we want the service to run on another server. For example, when we move from the development server to the production server, or when we want to reuse a service in another application and we want it to run on another server.

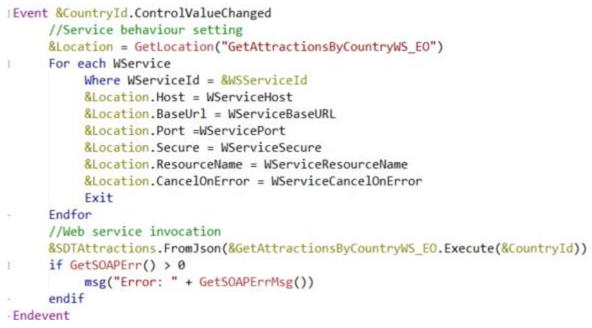
Note that after executing the web service, we use the GetSoapErr() and GetSoapErrMsg methods to obtain information about the result of the execution. We will look at these concepts in more detail later.

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>
```

The data that we assign to the location of a service can be parameterized in the database. In the example, we create a WService transaction and then, with a For Each command, we access the table containing the values of the attributes to assign the location properties.

In addition to defining a variable of the Location data type and changing the values of its properties by code as we have just seen, another way to assign the values of a location is by using a Location.xml file that we place in the web folder of our target environment; for example, the Web folder in a .NET model or web-inf in Java.

When GeneXus compiles the application, if it finds a file named location.xml in the folder mentioned above, it reads it and assigns the location values dynamically at runtime.

The format of the location.xml file is as shown on the screen.

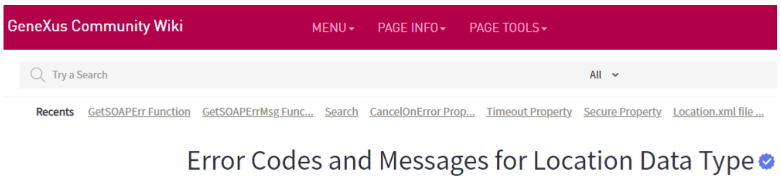
In GXLocation Name we assign the name of the external object associated with our web service.
Host is the name of the server that will host the service, without including http or https; that is, only www.servername.com, for example.
The port is a numeric value that depends on the type of server; it is recommended that the base URL be between slashes, for example /services/.
Secure with the value 1 indicates that a secure server will be used; that is, with HTTPS protocol and the value 0 for HTTP.
Proxyserverhost and Proxyserverport are used when there is a proxy server

between the client and the server providing the service.
Timeout is assigned with the maximum time in seconds that the system must wait for a response to be sent to the server after each request. If the value 0 is assigned, it means that the waiting time is indefinite.
Then, if required, the authentication characteristics of the HTTP communication can be defined.

The properties assigned at runtime to a variable of the Location data type will take precedence over those assigned at runtime through a location.xml file; in turn, the latter will take precedence over those assigned at generation time. This allows for more dynamism in the configuration of locations.

Error handling in 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

When we invoke a SOAP service and assign values to a location variable, we can assign a value to the CancelOnError property, which allows us to obtain the result of the operation and handle errors. Here is a typical example of use.

The behavior of the service will depend on the value assigned to the CancelOnError property.

For example, if we assign the value 0, the program that invokes the web service will always cancel the execution at the end of the invocation. This is the default value.
If we assign the value 1, the calling program will cancel the execution if an error occurs.
If we assign the value 2, the calling program will NOT cancel the execution if an error occurs; to obtain information about the error we can use the GetSOAPErr() and GetSOAPErrMsg() functions.

The GetSOAPErr() function returns a numeric value corresponding to the error code of the last SOAP operation.
If the function returns 0, no error occurred. Otherwise, the code depends on the type of error and it varies if the error occurred in the client or in the server.

The GetSOAPErrMsg() function returns a description of the error of the last SOAP operation, which allows us to give a more user-friendly message.

A table of possible error codes, both in the client and in the server, can be found in the Wiki article “Error Codes and Messages for Location Data Type.”

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