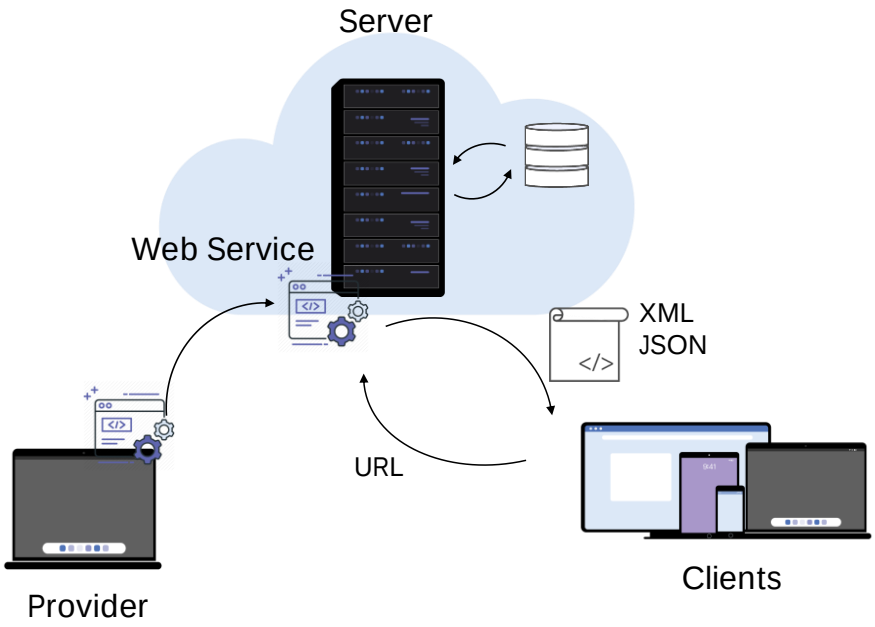
# Web Services in GeneXus

## Introduction

GeneXus™

Next, we'll see what web services are and how they can be used in a GeneXus application.

## Definition




Web Services are programs that provide useful functionalities to other programs and are stored in web servers so that they can be located and invoked over a network, usually the Internet.

When we publish the backend services of our application on the web server so that they can be accessed by other systems, they become web services. To facilitate access to these services, standards are used that define how to interact with them and the format of the information received.

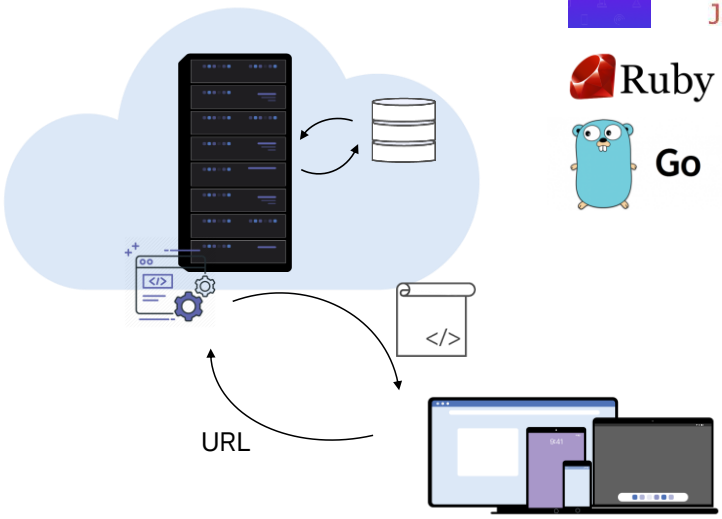
The service provider "publishes" a Web Service on a server and the client applications "consume" the Web Service published.

To access the service, the client application uses its location (URL) to invoke it and sends it the required parameters.

Then, it receives the returned information, usually as a structure in XML or JSON format.

# Web Service Protocols

| Protocol / Standard | |
|---|---|
| SOAP | WSDL |
| REST | OpenAPI |



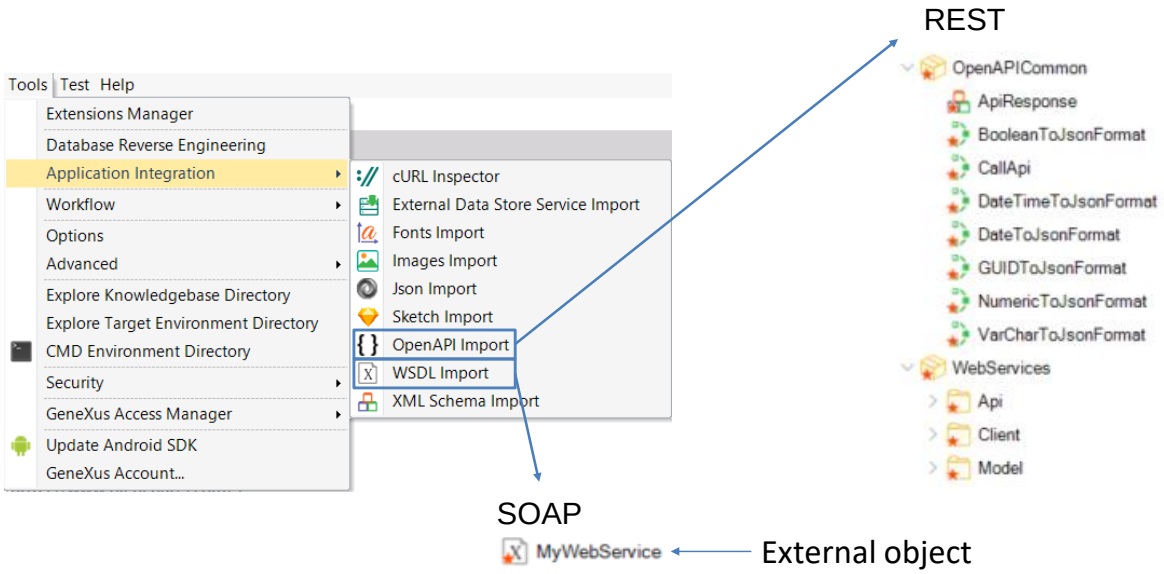

Web Services can be developed by following different standards; the most common ones in the industry are SOAP and REST. Each standard defines how to publish the information about the functions available in the web service.

SOAP web services use a definition written in WSDL (Web Services Description Language), while REST services use the OpenAPI standard.

To access a published Web Service, we must know its location and import its definition in order to access the functions available in the web service.

GeneXus allows consuming Web Services that have been developed in any programming tool or platform, with SOAP or REST protocols.

## How to consume a Web Service in GeneXus

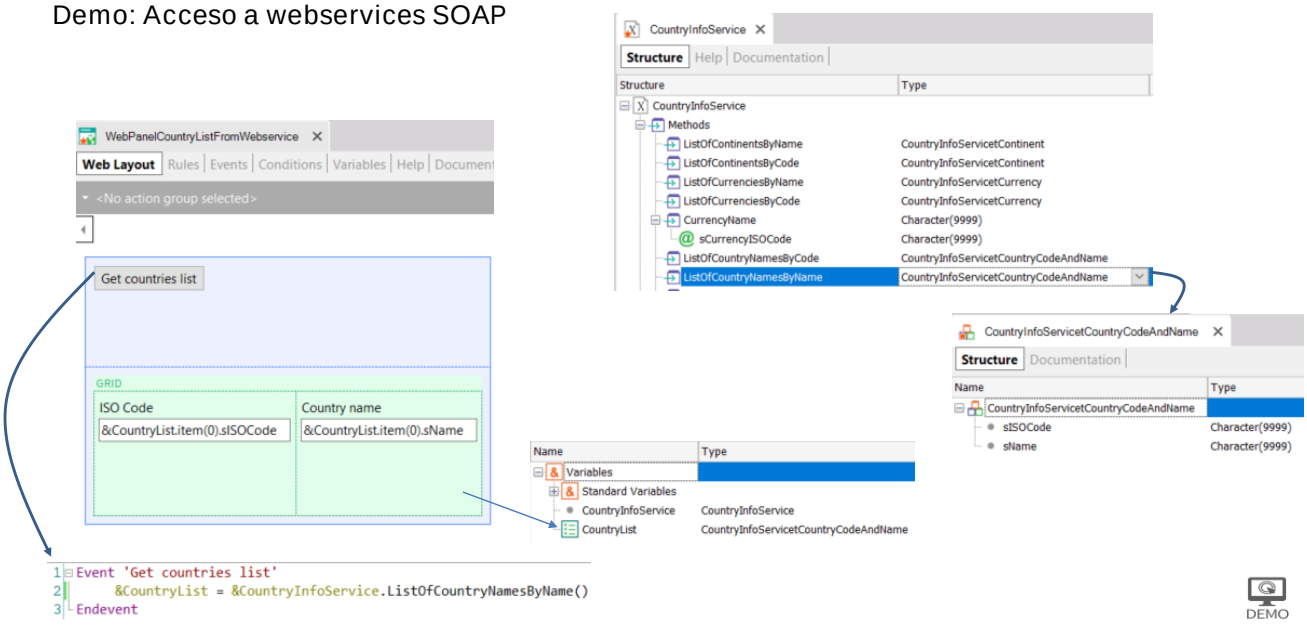



To integrate a Web Service into a GeneXus application, go to Tools/Application Integration; select WSDL Import if the Web Service follows the SOAP protocol, and OpenAPI Import if the Web Service has a REST architecture.

This will trigger a wizard that will vary depending on the type of Web Service selected. Finally, if the Web service is SOAP, GeneXus will automatically create an External Object associated with the Web Service and the structured data types required for managing its data.

Otherwise, if it was REST, several GeneXus objects will be automatically created (and we will usually include them in a module, for example, WebServices). Also, they will allow us to automatically run the service through these objects in our KB. The wizard will leave in an API folder the programs to invoke, and in a Model folder the SDTs to manage the data.

Demo: Acceso a webservices SOAP




Let's start by importing a list of countries from a SOAP web service into our application.
To do so, we access the menu options Tools/Application Integration/WSDL import and type the URL displayed on screen (http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL), and click on Next.
We see that a service called CountryInfoService was found. To order the imported objects we are going to save them in the SOAPWebService folder, leave the suggested prefix, and click on Next.
We click on the "+" sign and on the Service Description node. Note that a list of the functions offered by the web service opens, such as: ListOfCountryNamesByName to get the list of country names, CountryCurrency to get a country's currency, CountryFlag to get an image of its flag, etc.

We click on Import so that GeneXus imports the web service definition and we see that in the output window we are informed that a series of structured data types and other components are being imported. At the end, we confirm that the SOAPWebService folder appears in the KBNavigator, and in its contents we find the external object CountryInfoService and a series of SDTs that will allow us to store the information received from each method of the service.

If we double click on CountryInfoService, we can see that all the methods offered by the CountryInfoService web service have been incorporated to the external object, detailing the necessary parameters of each method

and what type of data it returns. In particular, we're interested in the ListfOfCountryNamesByName method, which will return a list of countries ordered by name.

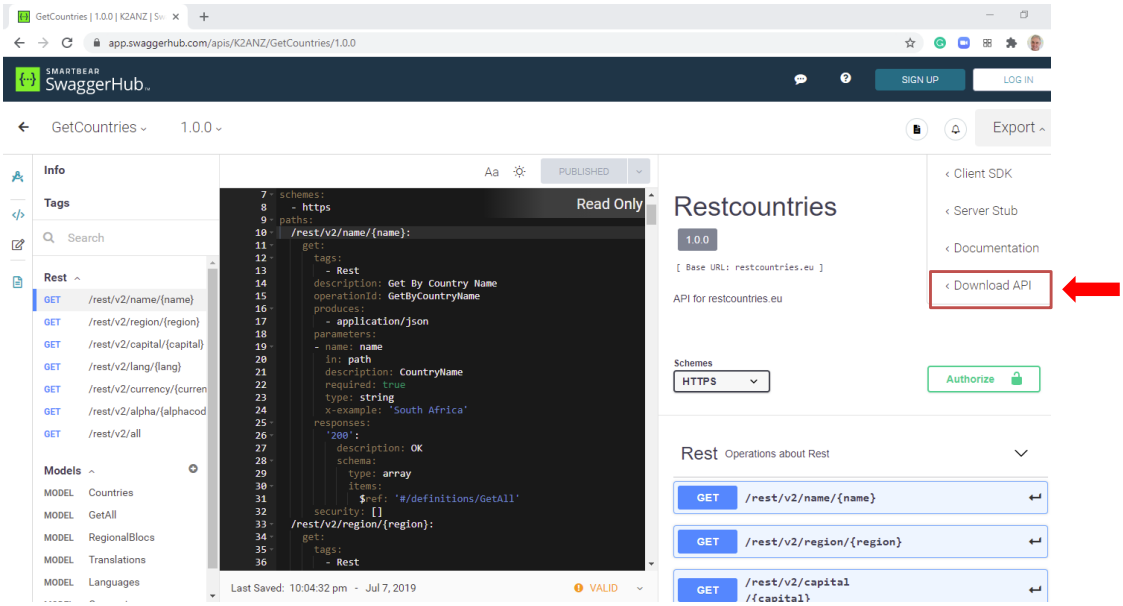
Now we create a web panel called WebPanelCountryListFromWebService. It its variables, we create a variable of CountryInfoService type that automatically takes the data type of the external object.
If we return to the definition of the external object, we see that the data type returned by the ListfOfCountryNamesByName method is an SDT called CountryInfoServicetCountryCodeAndName. We open it and see that it stores the ISO Code of the country and its name.
Let's return to the web panel, create a CountryList variable of the data type SDT CountryInfoServicetCountryCodeAndName and set it as a collection.
Now, in the form, we drag a button with the event name: Get countries list, double click on it and in the event we insert the variable &CountryList and assign it the variable &CountryInfoService. If we type a period, we see that we can access all the methods of the web service, so we choose ListOfCountryNamesByName.
We return to the form and drag the CountryList variable based on the SDT and press OK.
We run it.

If we open the web panel WebPanelCountryListFromWebService and press the button Get countries list, we obtain the list of countries in alphabetical order with the ISO Code of each one of them, as we expected.

This data can then be used in our travel agency application as a selection list to filter by a country, or in different uses.

# Demo: Acceso a webservices REST con protocolo OpenAPI

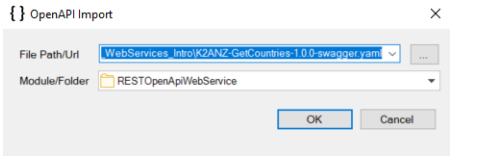
GeneXus actualmente puede importar OpenAPI versión 2.0

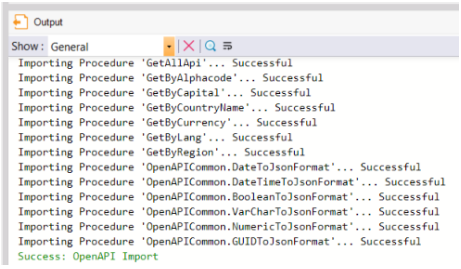


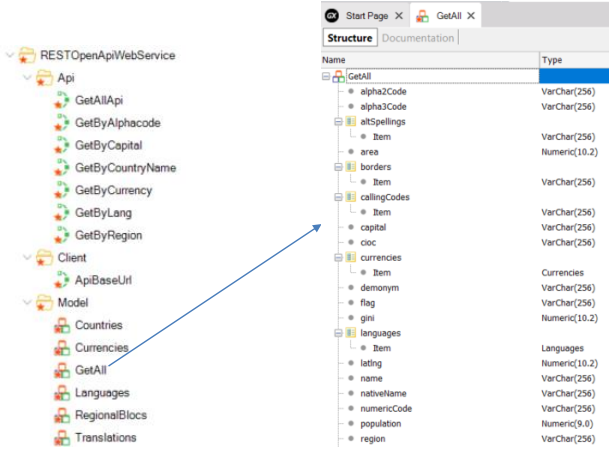

DEMO

Let's look at the case of importing a REST web service that complies with the OpenAPI specification, also known as the Swagger specification.
The example is obtained from the web page: https://app.swaggerhub.com, an API called GetCountries. We choose the Download API option and download the .yaml file.

## Demo: Acceso a webservices REST con protocolo OpenAPI




```
GetAllApi(&GetAllOUT, &HttpMessage, &IsSuccess)
```

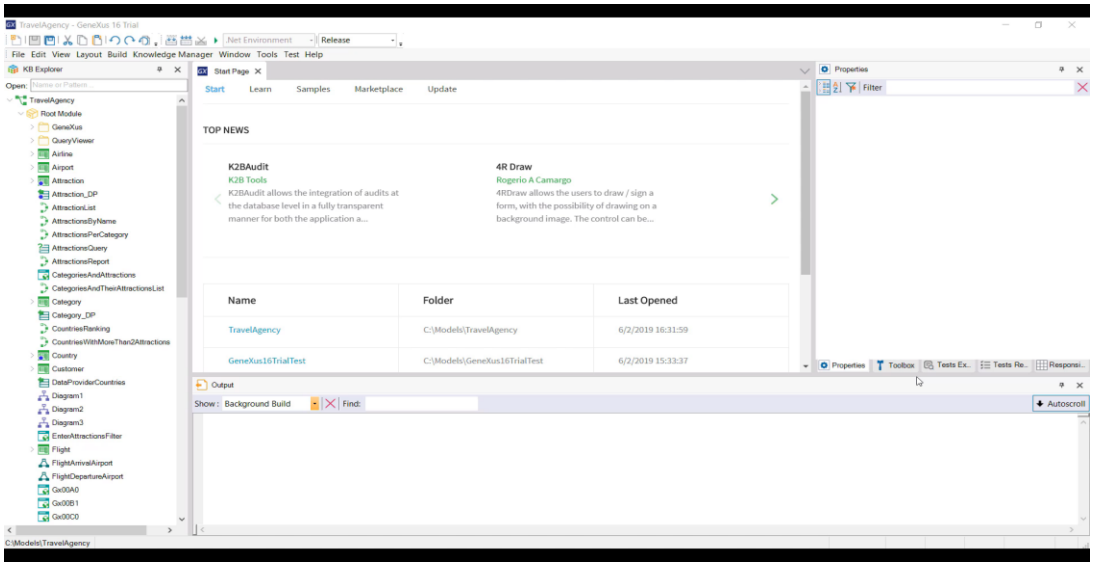
DEMO

Next, we select Tools/Application Integration/OpenApi import and choose the .yaml file we had downloaded. As a folder we type RESTOpenApiWebService and click on Next.
We see in the Output window that several elements are imported and when finished we open the destination folder. We can see that the API folders were created containing the methods that we can invoke, the Client folder with the ApiBaseURL method and the Model folder containing several SDTs that are the data types returned by the previous methods.

If we open the SDT called GetAll we can see all the data we can retrieve from the countries. And if we want to get the list of countries, we can invoke the GetAllApi method that will return the data in the variables &GetAllOut (collection of GetAll elements), &HttpMessage and the Boolean variable &IsSuccess.

Next, we can run through the &GetAllOut collection to get the country data.

## Example of use of a REST Web Service




DEMO

[ DEMO: https://youtu.be/bvmt0Gjcxpw ]

To import the Web Service, we go to Tools/Application Integration and select OpenAPI Import. In the File Path / URL we type: C:\Models\TravelAgency\CSharpModel\Web\default.yaml because it is where the Rest API documentation was generated, and select the WebServices module as target. We confirm that everything has been imported correctly; if we open the WebServices module, the imported procedure is located in the API folder. Also, we see that in the Model folder there is an SDT called CreateNewAirlineWSInput, and if we open it we see that the parameters we need to pass to the service are available here.

To test whether the Web Service works correctly, we created a web panel called CreateNewAirlineUsingWS. Next, we drag a button to the form and call the event Create airline. We open the variables and create a variable &CreateNewAirlineWSInput which is automatically set as SDT type.
In order to receive feedback about the result of the Web Service execution, we also created a variable &IsSuccess and another one &HttpMessage. Note that the types are assigned by default.

We double-click on the button and in the event we load the members of the SDT; next, we invoke the Web Service passing the SDT as a parameter, and finally we write the following code to show messages on screen.

We press F5... We see the lines that we have entered... And we execute the webpanel that we have just created.
We press the button and see that the service informs us that the airline was created correctly.
To confirm it we select the Airline transaction… And see that the airline we wanted has been actually added.

## Example of use of a SOAP Web Service

https://training-legacy.genexus.com/en/training/global/courses/genexus-en/genexus-15-course-analyst#web-services-gx15

Here we saw an example of use of a Web Service of REST type. To see how to use a SOAP service, you can watch this video.

## Example of use of an ODATA Web Service

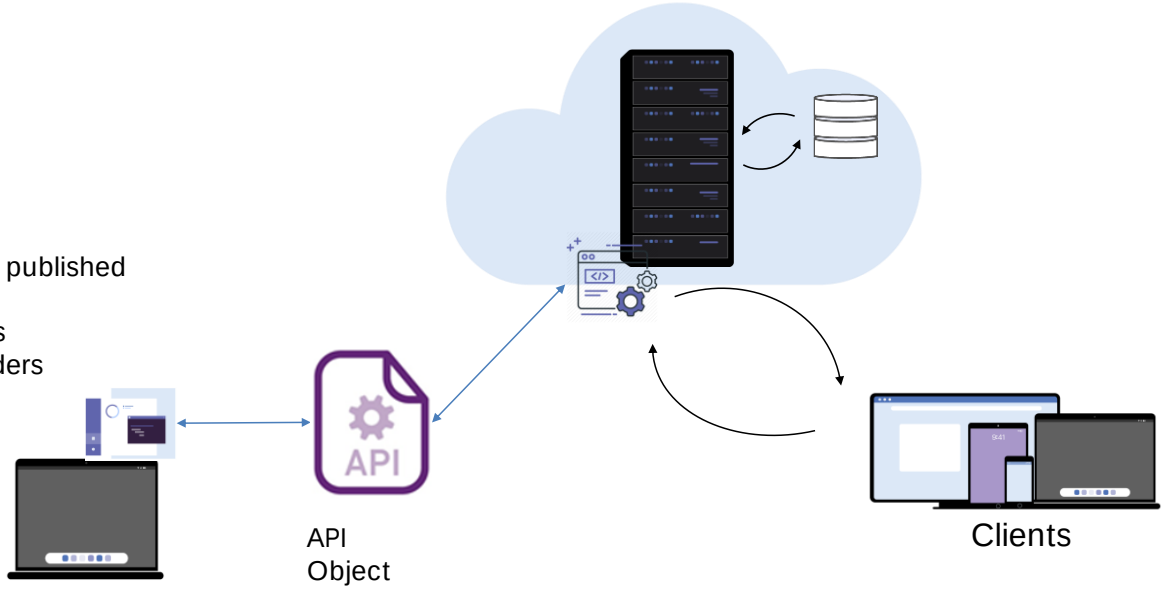
https://training-legacy.genexus.com/en/training/esp-en/main/ampliacion-general/web-services-that-access-the-providers-database-pdf

And if you want to see an example of use of a Web Service with OData protocol, read the following document.

## API Object

Objects to be published as services:

- Procedures
- Data Providers




We will now see another way to publish objects as services, using the API object.

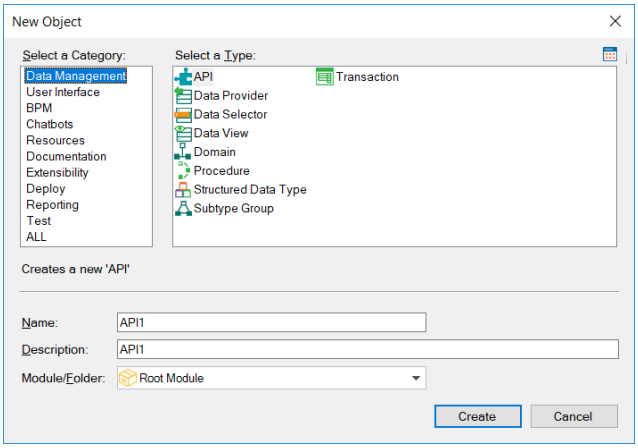
The API object (Application Programming Interface) is a GeneXus object that allows us to programmatically define an access interface to objects of our application, such as procedures or data providers.
This means that an external application will be able to access these objects published as web services.

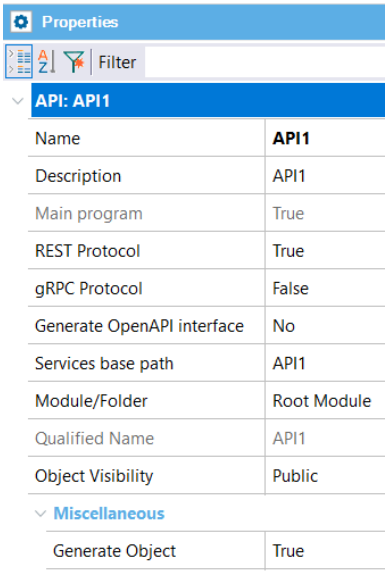
The API object adds an intermediate layer that separates the interface from the implementation details, so that future programming changes to objects do not affect the way they are invoked by external applications.

For example, a mapping is made between the internal name of the object in the KB and the name with which it is exposed as a service. Also, you can change the type and name of the parameters or change the access path, without this affecting the way the service is invoked.

This abstraction provides significant flexibility because we can evolve our services without forcing the applications that use them to change their code to accommodate those changes.

# API Object (continued)

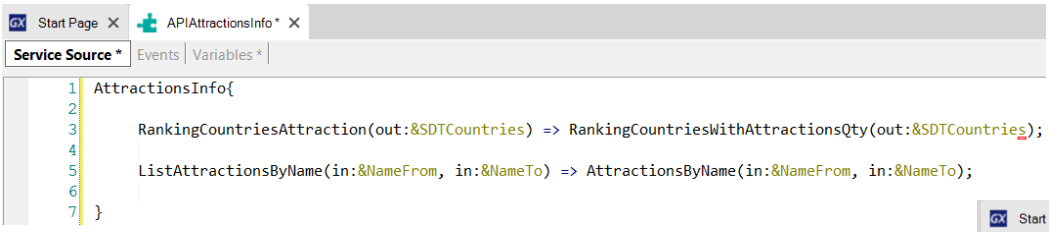


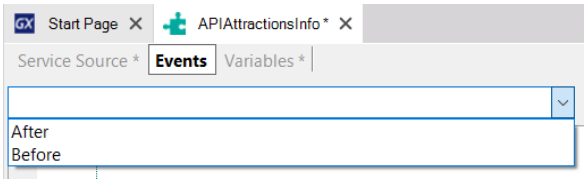



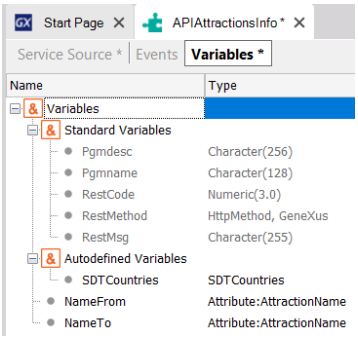

To create an API object, go to the Data Management category and select the API type. In its properties we can define at design time its name, address, and protocol to be used, among other things.

The gPRC protocol is a modern open source framework developed by Google, which allows calls to remote procedures with high performance and in a bidirectional way.

# API Object (continued)




The API object has three sections: Service Source, Events, and Variables.

The name of each service to be published is defined in Service Source, through a declarative syntax, with the corresponding parameters and the name of the GeneXus object in our KB that we are exposing as a service, with its corresponding parameters.

In the example, we see that the data provider that we previously created under the name RankingCountriesWithAttractionsQty is being published as a service under the name RankingCountriesAttractions, and the parameters that we expose are the same as those of the object.
The same is true for the AttractionsByName list that we publish under the name ListAttractionsByName.

In the future, this definition will allow us to change the name or parameters of the GeneXus object without changing the definition of how this object is published.

In the events we have the Before and After event, with which we can perform actions that are executed before or after the invocation of the object as a service.

In the variables there are some of the standard type, which allow us to define or change characteristics of the API object at runtime.

# More information about Web Services

WSDL: https://wiki.genexus.com/commwiki/servlet/wiki?6181
OpenAPI: https://wiki.genexus.com/commwiki/servlet/wiki?31864
OData: https://wiki.genexus.com/commwiki/servlet/wiki?40713
API Object: https://wiki.genexus.com/commwiki/servlet/wiki?46151

For more information about Web Services in GeneXus, click on these wiki links.

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