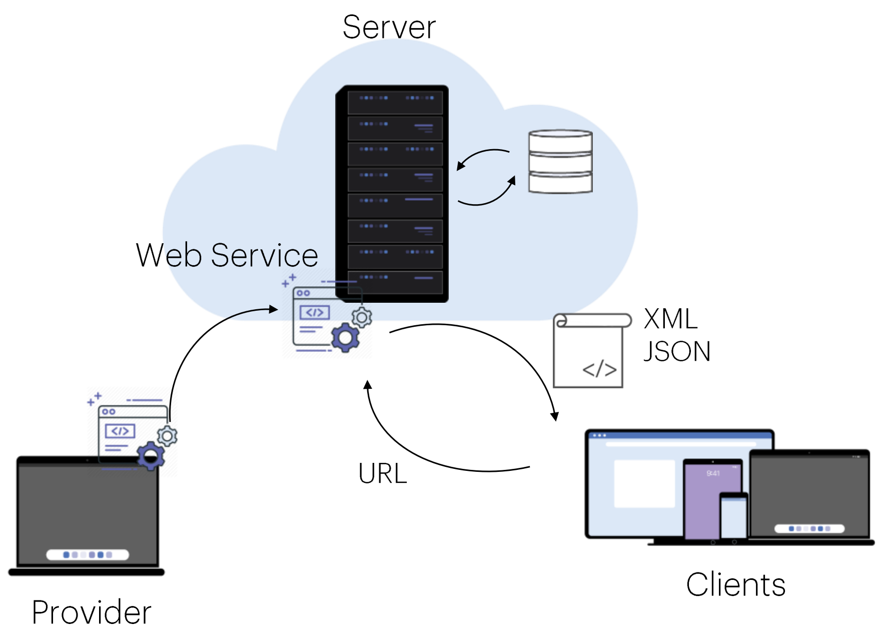
Web Services in GeneXus

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) in order to invoke it and sends it the required parameters.

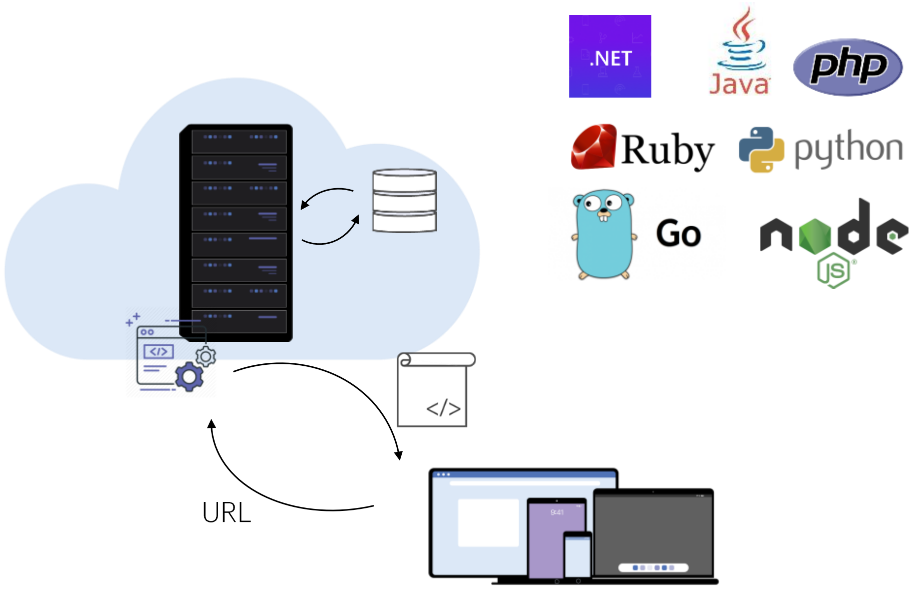
Then, it receives the returned information, usually as a structure in XML or JSON format.

Web Service Protocols

Protocol / Standard

SOAP ⟶ WSDL

REST ⟶ OpenAPI



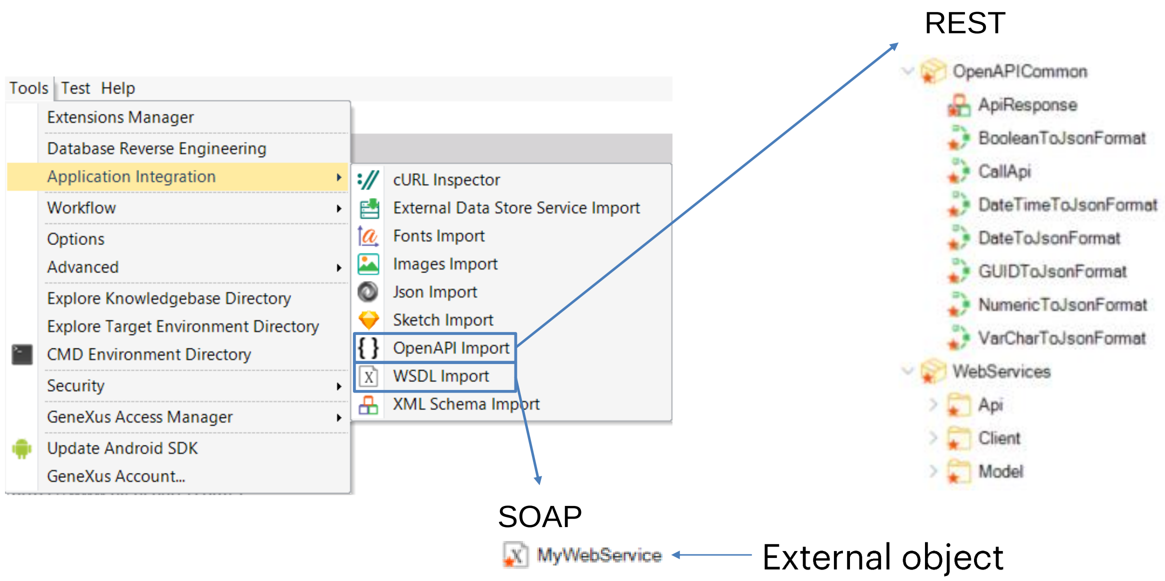
Web Services can be developed by following different standards; the most common ones in the industry are SOAP and REST. Each standard defines how to publish the information about the functions available in the web service.

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

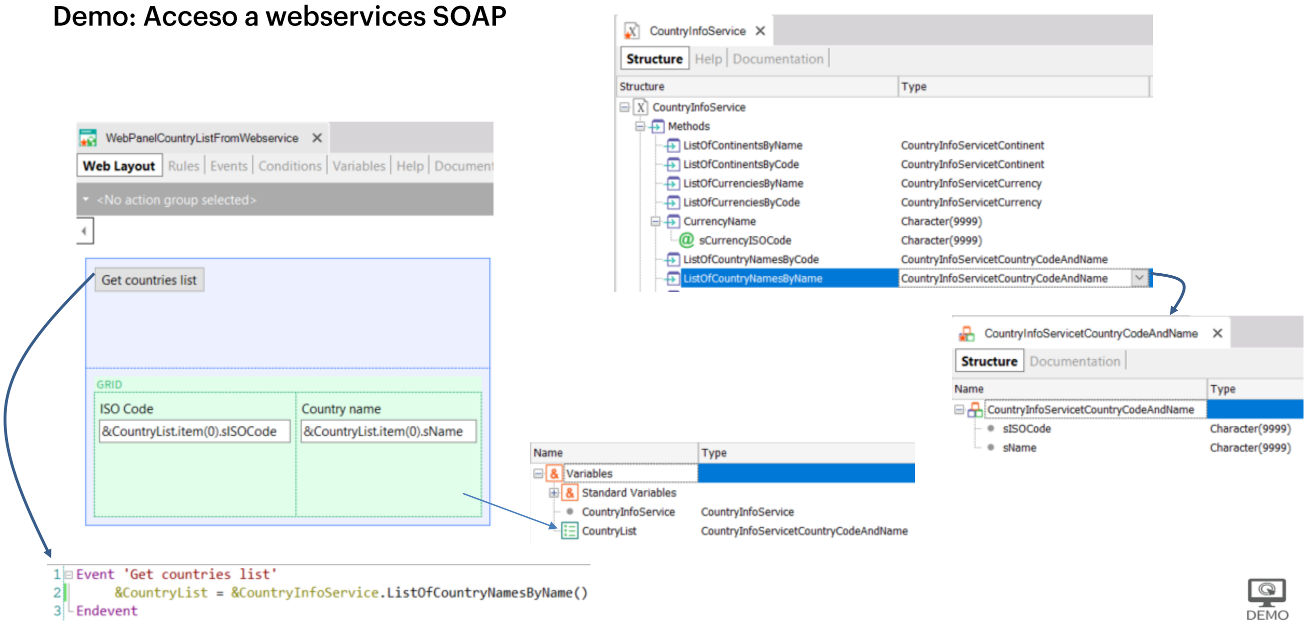


To integrate a Web Service into a GeneXus application, go to Tools/Application Integration; select WSDL Import if the Web Service follows the SOAP protocol, and OpenAPI Import if the Web Service has a REST architecture.

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'll 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


WebPanelCountryListFromWebservice
Web Layout | Rules | Events | Conditions | Variables | Help | Document
<No action group selected>
Get countries list
GRID
DEMO

ISO Code	Country name
&CountryList.item(0).sISOCode	&CountryList.item(0).sName

CountryInfoService — Structure | Help | Documentation

Structure	Type
CountryInfoService	
Methods	
ListOfContinentsByName	CountryInfoServicetContinent
ListOfContinentsByCode	CountryInfoServicetContinent
ListOfCurrenciesByName	CountryInfoServicetCurrency
ListOfCurrenciesByCode	CountryInfoServicetCurrency
CurrencyName	Character(9999)
sCurrencyISOCode	Character(9999)
ListOfCountryNamesByCode	CountryInfoServicetCountryCodeAndName
ListOfCountryNamesByName	CountryInfoServicetCountryCodeAndName

CountryInfoServicetCountryCodeAndName — Structure | Documentation

Name	Type
CountryInfoServicetCountryCodeAndName	
sISOCode	Character(9999)
sName	Character(9999)

Name	Type
Variables	
Standard Variables	
CountryInfoService	CountryInfoService
CountryList	CountryInfoServicetCountryCodeAndName

```
Event 'Get countries list'
      &CountryList = &CountryInfoService.ListOfCountryNamesByName()
Endevent
```

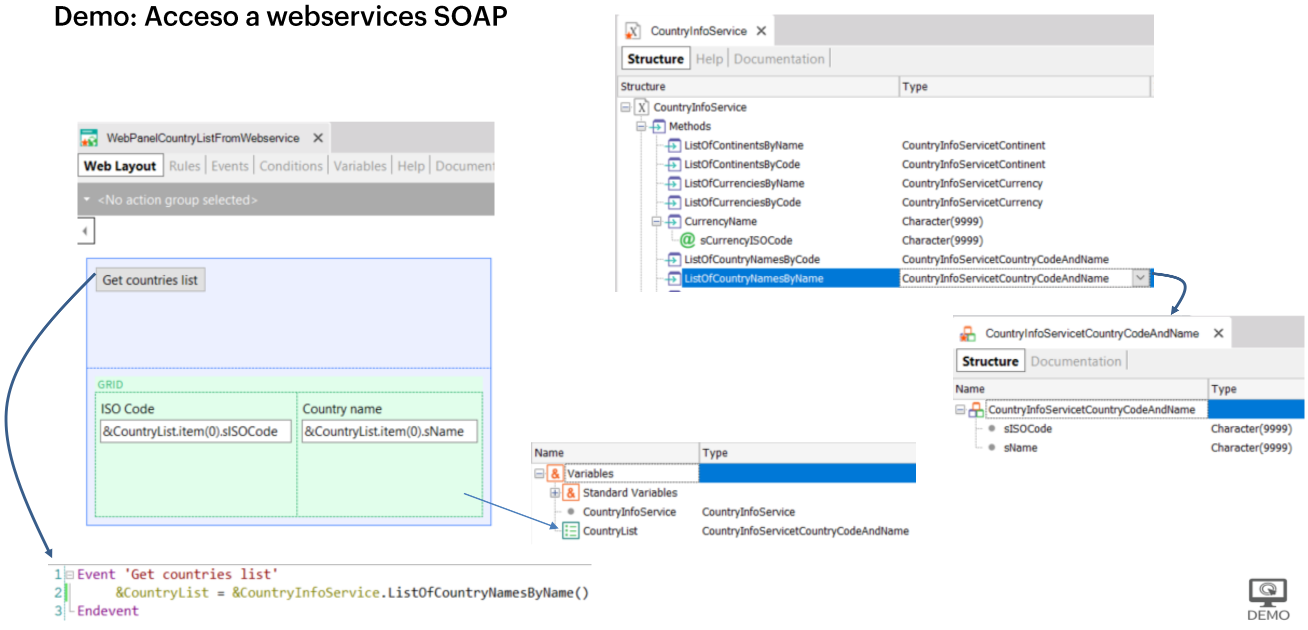
Let's start by importing a list of countries from a SOAP web service into our application.
To do so, we access the menu options Tools/Application Integration/WSDL import and type the URL displayed on screen (http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL), and click 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 LisfOfCountryNamesByName method, which will return a list of countries ordered by name.

GeneXus

Demo: Acceso a webservices SOAP



Structure	Type
CountryInfoService	
Methods	
ListOfContinentsByName	CountryInfoServicetContinent
ListOfContinentsByCode	CountryInfoServicetContinent
ListOfCurrenciesByName	CountryInfoServicetCurrency
ListOfCurrenciesByCode	CountryInfoServicetCurrency
CurrencyName	Character(9999)
sCurrencyISOCode	Character(9999)
ListOfCountryNamesByCode	CountryInfoServicetCountryCodeAndName
ListOfCountryNamesByName	CountryInfoServicetCountryCodeAndName

Name	Type
CountryInfoServicetCountryCodeAndName	
sISOCode	Character(9999)
sName	Character(9999)

Name	Type
Variables	
Standard Variables	
CountryInfoService	CountryInfoService
CountryList	CountryInfoServicetCountryCodeAndName

```
Event 'Get countries list'
	&CountryList = &CountryInfoService.ListOfCountryNamesByName()
Endevent
```

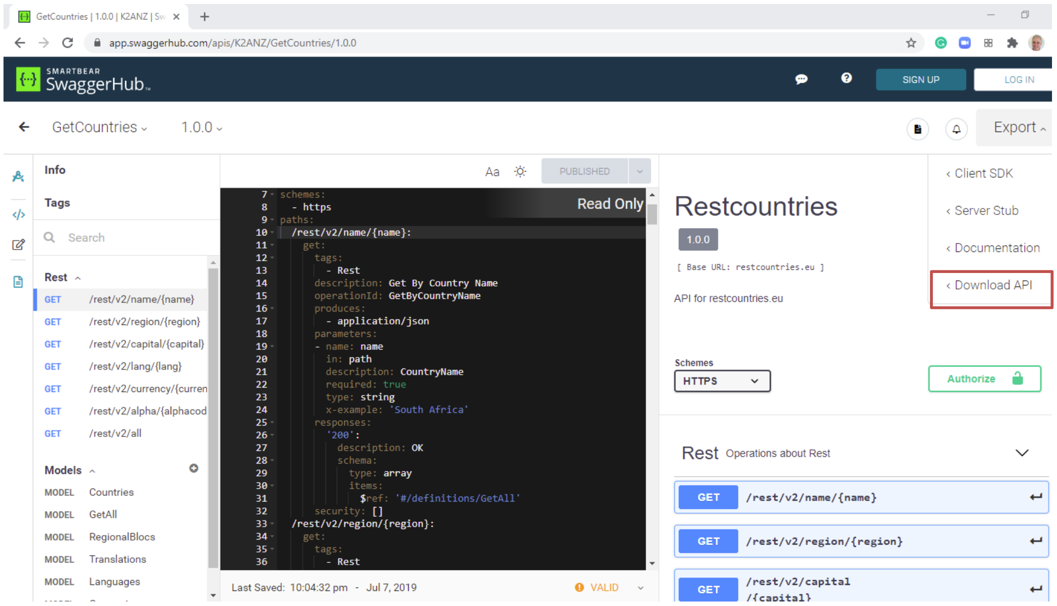
Now we create a web panel called WebPanelCountryListFromWebService. It its variables, we create a variable of CountryInfoService type that automatically takes the data type of the external object.
If we return to the definition of the external object, we see that the data type returned by the LisfOfCountryNamesByName 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



```
schemes:
  - https
paths:
  /rest/v2/name/{name}:
    get:
      tags:
        - Rest
      description: Get By Country Name
      operationId: GetByCountryName
      produces:
        - application/json
      parameters:
      - name: name
        in: path
        description: CountryName
        required: true
        type: string
        x-example: 'South Africa'
      responses:
        '200':
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/GetAll'
      security: []
  /rest/v2/region/{region}:
    get:
      tags:
        - Rest
```

Last Saved: 10:04:32 pm - Jul 7, 2019 VALID

Restcountries

1.0.0

[Base URL: restcountries.eu]

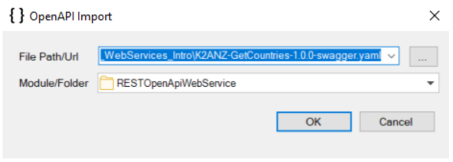
API for restcountries.eu

Rest Operations about Rest

DEMO

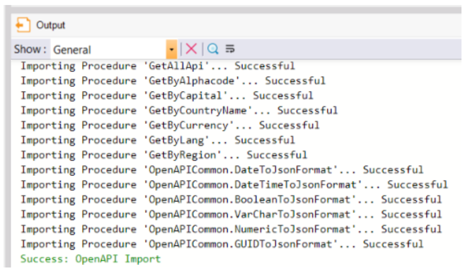
Let's look at the case of importing a REST web service that complies with the OpenAPI specification, also known as the Swagger specification.
The example is obtained from the web page: https://app.swaggerhub.com, an API called GetCountries. We choose the Download API option and download the .yaml file.

Demo: Acceso a webservices REST con protocolo OpenAPI

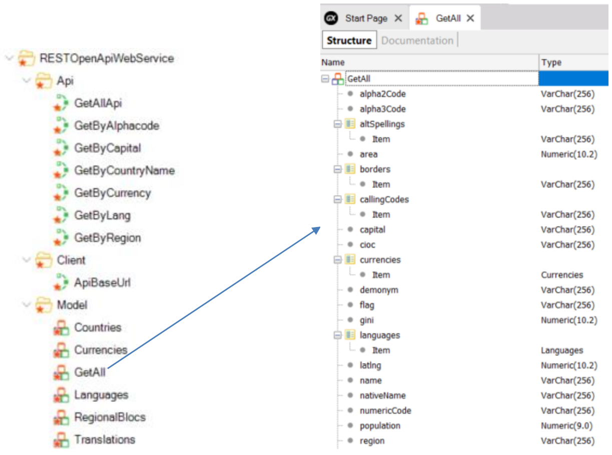


{ } OpenAPI Import

File Path/Url	WebServices_Intro\K2ANZ-GetCountries-1.0.0-swagger.yaml
Module/Folder	RESTOpenApiWebService



```
Importing Procedure 'GetAllApi'... Successful
Importing Procedure 'GetByAlphacode'... Successful
Importing Procedure 'GetByCapital'... Successful
Importing Procedure 'GetByCountryName'... Successful
Importing Procedure 'GetByCurrency'... Successful
Importing Procedure 'GetByLang'... Successful
Importing Procedure 'GetByRegion'... Successful
Importing Procedure 'OpenAPICommon.DateToJsonFormat'... Successful
Importing Procedure 'OpenAPICommon.DateTimeToJsonFormat'... Successful
Importing Procedure 'OpenAPICommon.BooleanToJsonFormat'... Successful
Importing Procedure 'OpenAPICommon.VarCharToJsonFormat'... Successful
Importing Procedure 'OpenAPICommon.NumericToJsonFormat'... Successful
Importing Procedure 'OpenAPICommon.GUIDToJsonFormat'... Successful
Success: OpenAPI Import
```



```
RESTOpenApiWebService
  Api
    GetAllApi
    GetByAlphacode
    GetByCapital
    GetByCountryName
    GetByCurrency
    GetByLang
    GetByRegion
  Client
    ApiBaseUrl
  Model
    Countries
    Currencies
    GetAll
    Languages
    RegionalBlocs
    Translations
```

Name	Type
GetAll	
alpha2Code	VarChar(256)
alpha3Code	VarChar(256)
altSpellings	
Item	VarChar(256)
area	Numeric(10.2)
borders	
Item	VarChar(256)
callingCodes	
Item	VarChar(256)
capital	VarChar(256)
cioc	VarChar(256)
currencies	
Item	Currencies
demonym	VarChar(256)
flag	VarChar(256)
gini	Numeric(10.2)
languages	
Item	Languages
latlng	Numeric(10.2)
name	VarChar(256)
nativeName	VarChar(256)
numericCode	VarChar(256)
population	Numeric(9.0)
region	VarChar(256)

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

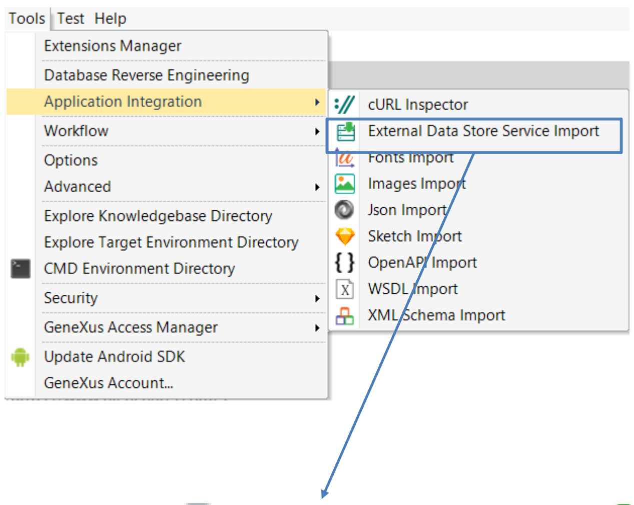
DEMO

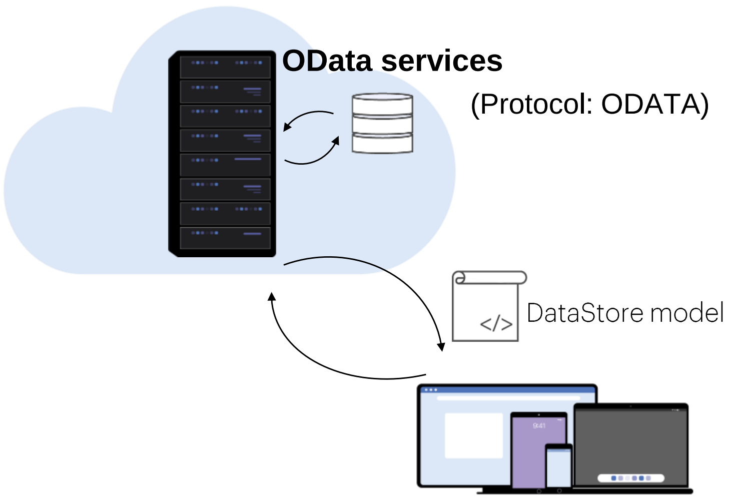
Next, we select Tools/Application Integration/OpenApi import and choose the .yaml file we had downloaded. As a folder we type RESTOpenApiWebService and click on Next.
We see in the Output window that several elements are imported and when finished we open the destination folder. We can see that the API 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.

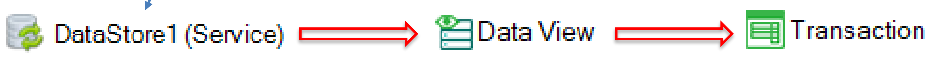
If we open the SDT called GetAll we can see all the data we can retrieve from the countries. And if we want to get the list of countries, we can invoke the GetAllApi method that will return the data in the variables &GetAllOut (collection of GetAll elements), &HttpMessage and the Boolean variable &IsSuccess.

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

How to consume a Web Service in GeneXus (continued)







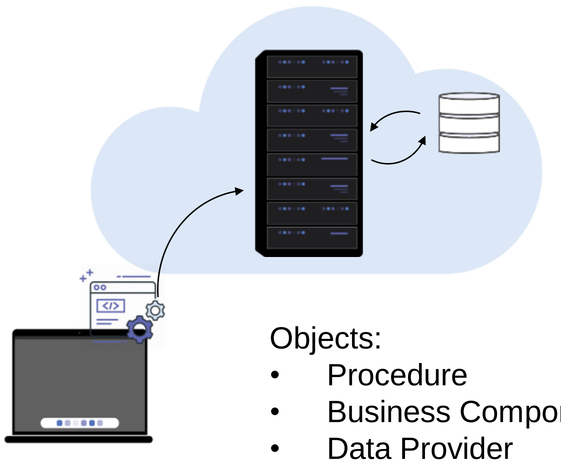
OData services are a special type of web services. These services use the Open Data Protocol (OData) designed to provide operations that insert, modify, or delete records in a database through a website.

Just as for a Soap or Rest Web Service its definition had to be imported first, in order to consume an OData service we must first import the model with the database entities included in the service.

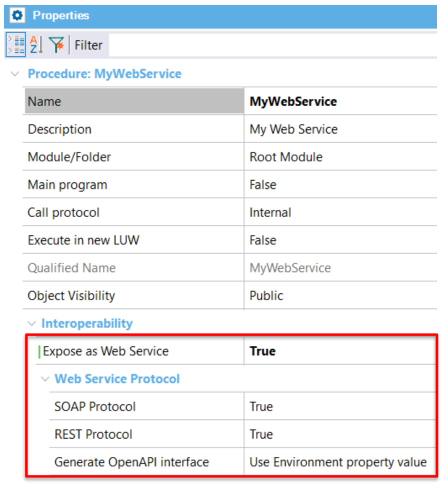
To do so, go to Tools/Application Integration and select External DataStore Service Import. Since the process implies creating a DataStore of Service type to associate it with the external datastore, we can only use this feature in GeneXus Full.

After executing the wizard, as many transaction objects will be created as entities were included in the model, and we will be able to work with them as with any other transaction in our application.

How to publish a Web Service with GeneXus



Objects:
- Procedure
- Business Component
- Data Provider



Properties

Filter

Procedure: MyWebService

Name	MyWebService
Description	My Web Service
Module/Folder	Root Module
Main program	False
Call protocol	Internal
Execute in new LUW	False
Qualified Name	MyWebService
Object Visibility	Public
Interoperability	
Expose as Web Service	True
Web Service Protocol	
SOAP Protocol	True
REST Protocol	True
Generate OpenAPI interface	Use Environment property value

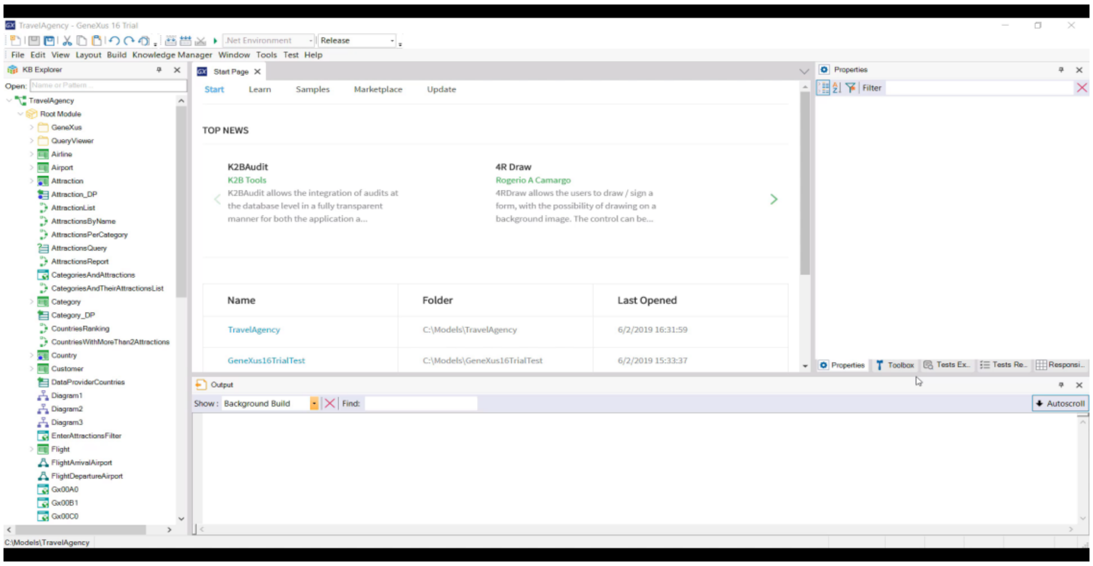
GeneXus also allows creating web services and publishing them on a web server.

The GeneXus objects that can be exposed as web services are procedures, business components, and data providers.

To expose one of these objects as a Web Service, we set the property Expose as Web Service to True, and indicate whether to use SOAP protocol, REST or both.

If you want the service to provide CRUD operations on a database, you can also generate the necessary information following the OData protocol.

Example of use of a REST Web Service



TOP NEWS

K2BAudit
K2B Tools
K2BAudit allows the integration of audits at the database level in a fully transparent manner for both the application a...

4R Draw
Rogerio A Camargo
4RDraw allows the users to draw / sign a form, with the possibility of drawing on a background image. The control can be...

Name	Folder	Last Opened
TravelAgency	C:\Models\TravelAgency	6/2/2019 16:31:59
GeneXus16TrialTest	C:\Models\GeneXus16TrialTest	6/2/2019 15:33:37

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

As an example, we will create a REST Web Service that inserts a new airline in the database and we will invoke it from our application.

To do so, first we open the Airline transaction and set its Business Component property to True. Then we are going to create a procedure object that receives by parameter the data of an airline and inserts a new record in the Airline table.

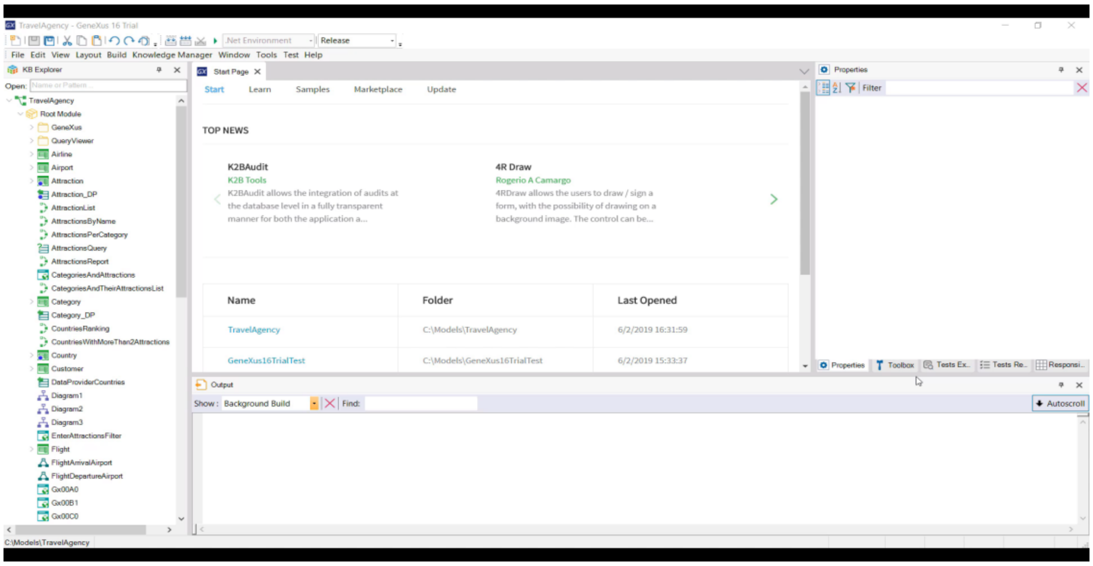
We call the procedure CreateNewAirlineWS and create a Parm rule with 2 input variables: &AirlineName and &AirlineDiscountPercentage. We create these two variables using the Add Variable, and then in the Variables section we create another variable called Airline, of Airline business component type.

In the source, we write the code to insert an airline with the data received by parameter. We don't have to enter a value for AirlineId because it is autonumbered.

Now, we open the properties of the procedure object and set Expose as Web Service to True, SOAP Protocol to False; we leave REST Protocol set to True and set Generate OpenAPI interface to Yes. The latter will allow the definition of our Web Service to be generated in the OpenAPI standard.
We right-click on the procedure and select Build With This Only, so that the service will be published on our web server, in this case on our local machine. In the Ouput window we see a message confirming that the documentation of the Rest API of the Web Service was generated, with its definition.

Before importing the Web Service, we'll create a new module called WebServices, so everything is stored in that module.

Example of use of a REST Web Service



TOP NEWS

K2BAudit
K2B Tools
K2BAudit allows the integration of audits at the database level in a fully transparent manner for both the application a...

4R Draw
Rogerio A Camargo
4RDraw allows the users to draw / sign a form, with the possibility of drawing on a background image. The control can be...

Name	Folder	Last Opened
TravelAgency	C:\Models\TravelAgency	6/2/2019 16:31:59
GeneXus16TrialTest	C:\Models\GeneXus16TrialTest	6/2/2019 15:33:37

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.

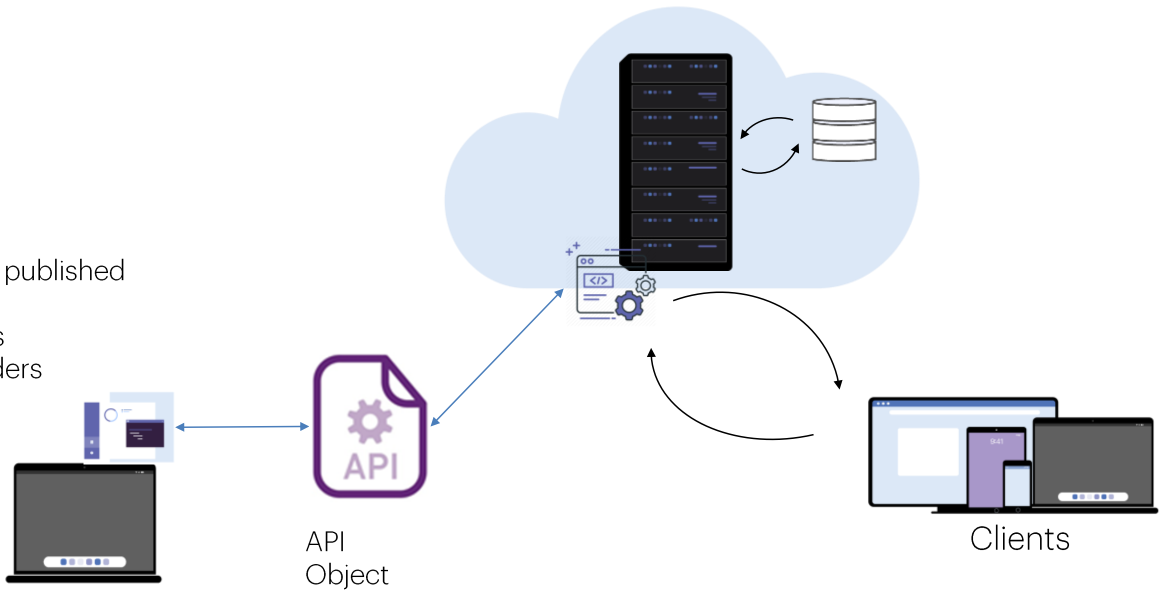
We double-click on the button and in the event we load the members of the SDT; next, we invoke the Web Service passing the SDT as a parameter, and finally we write the following code to show messages on screen.

We press F5… We see the lines that we've entered… And we execute the webpanel that we've 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.

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.

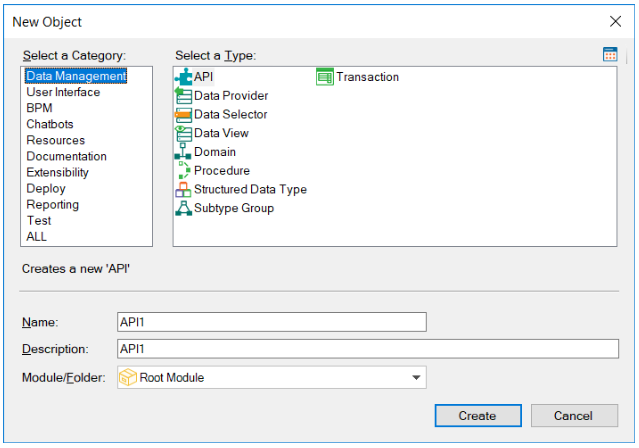
The API object (Application Programming Interface) is a GeneXus object that allows us to programmatically define an access interface to objects of our application, such as procedures or data providers.
This means that an external application will be able to access these objects published as web services.

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 these changes.

API Object (continued)



New Object

Select a Category: Data Management, User Interface, BPM, Chatbots, Resources, Documentation, Extensibility, Deploy, Reporting, Test, ALL

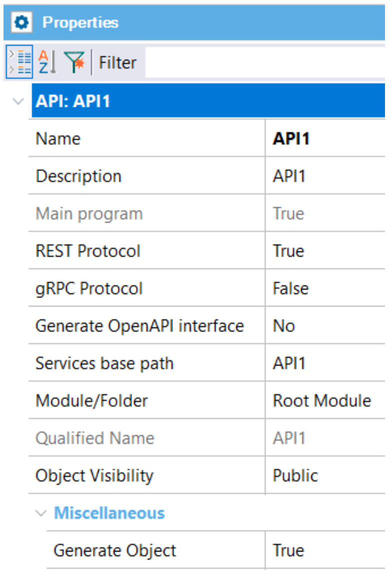
Select a Type: API, Data Provider, Data Selector, Data View, Domain, Procedure, Structured Data Type, Subtype Group, Transaction

Creates a new 'API'

Name: API1

Description: API1

Module/Folder: Root Module



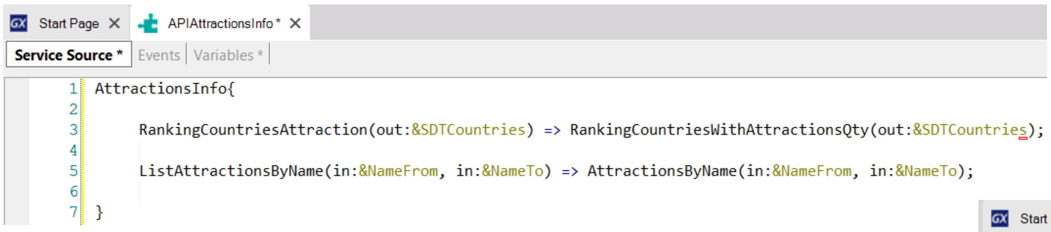
API: API1

Name	API1
Description	API1
Main program	True
REST Protocol	True
gRPC Protocol	False
Generate OpenAPI interface	No
Services base path	API1
Module/Folder	Root Module
Qualified Name	API1
Object Visibility	Public
Miscellaneous	
Generate Object	True

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)

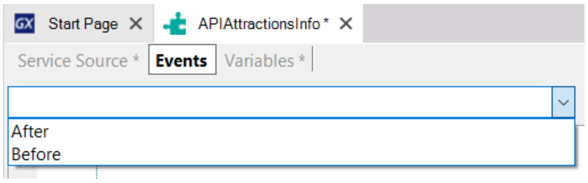


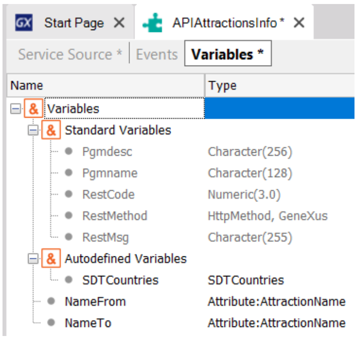
```
AttractionsInfo{

    RankingCountriesAttraction(out:&SDTCountries) => RankingCountriesWithAttractionsQty(out:&SDTCountries);

    ListAttractionsByName(in:&NameFrom, in:&NameTo) => AttractionsByName(in:&NameFrom, in:&NameTo);

}
```





Name	Type
Variables	
Standard Variables	
Pgmdesc	Character(256)
Pgmname	Character(128)
RestCode	Numeric(3.0)
RestMethod	HttpMethod, GeneXus
RestMsg	Character(255)
Autodefined Variables	
SDTCountries	SDTCountries
NameFrom	Attribute:AttractionName
NameTo	Attribute:AttractionName

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 events Before and After, 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