

Web Services. Advanced topics

Publishing REST services with GeneXus

GeneXus[™]

So far, we have only published SOAP services. Now we will see how to publish REST services with GeneXus, using the usual mechanism and also the API object, with all the advantages provided by this object.

Publishing a REST web service

- Procedure
- Business Component
- Data Provider

+

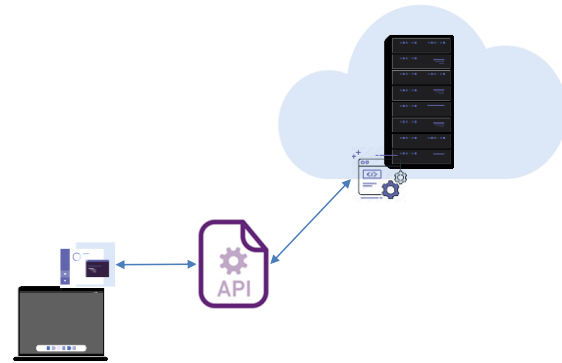
Expose as Web Service	True
Web Service Protocol	
SOAP Protocol	False
REST Protocol	True
Generate OpenAPI interface	Use Environment property value

- Procedure
- Data Provider

+



API Object



In GeneXus, there are two ways to publish a REST service.

One that we have already seen, which is to expose procedure objects, data providers, or business components using the Expose as Web Services property set to True and setting the REST Protocol property to True.

Another way is using the API object, which is useful to expose procedure objects or data providers. This API object allows exposing services with REST or gRPC protocols, making it possible to group several services that are semantically or functionally related.

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.

This allows for a mapping between the internal name of the object in the KB and the name used to expose it as a service, as well as changing the type and name of the parameters or changing the access path without affecting the way in which 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 adapt to the changes.

For that reason, to expose Procedures and Data Providers as REST services, it is strongly recommended to always use the API object, instead of the Expose as Web

Service and Rest Protocol properties.

Publishing a service with the API object

The image displays four screenshots from the GeneXus IDE illustrating the steps to create and configure an API object:

- New Object Dialog:** Shows the 'Select a Type' dropdown set to 'API'. The 'Name' field is 'GetAttractionsInfo', the 'Description' is 'Get Attractions Info', and the 'ModuleFolder' is 'Root Module'.
- Service Source Editor:** Shows the declarative definition of the API object:


```

1 GetAttractionsInfo{
2
3   GetAttractionsByCountry(in:&CountryId, out:&Attractions)
4     => GetAttractionsByCountryWS(in:&CountryId, out:&Attractions);
5
6   GetAttractionsByCountryAndTrips(in:&CountryId, in:&TripsQty, out:&Attractions)
7     => GetAttractionsByCountryWithTripsWS(in:&CountryId, in:&TripsQty, out:&Attractions);
8 }
9
      
```
- Variables Explorer:** Shows the 'Autodefined Variables' section with the following table:

Name	Type
CountryId	Attribute:CountryId
Attractions	LongVarChar(2M)
TripsQty	Numeric(4,0)
- Events Editor:** Shows the 'Events' tab with two events defined for the 'GetAttractionsByCountry' service:

Controls	Events
GetAttractionsByCountry	Before
GetAttractionsByCountry	After

To test what we have seen, let's create an API object to publish the 2 methods we had to get attraction data by country, as REST services. We call the API object `GetAttractionsInfo`.

The API object has three parts: Service Source, Events, and Variables. In the Service Source section we define, using a declarative syntax, the name of each service to be published, with the corresponding parameters, specifying whether each parameter is input or output and the name of the GeneXus object in our KB that we are exposing as a service, with its corresponding parameters.

Here we are exposing the `GetAttractionByCountry` service based on the `GetAttractionsByCountryWS` procedure. The parameters match.

Similarly, below we are exposing the `GetAttractionsByCountryWithTripsWS` procedure as a service with the name `GetAttractionsByCountryAndTrips`, and the parameters we expose are the same as those of the procedure object.

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. We also have Before and After events for each exposed service. In these events it is not possible to access the database. If it is required, it must be included in the code of the procedure or data provider exposed as a service.

In the variables there are some under the group of standard variables, which allow us to know or change characteristics of the API object at runtime. The variable &Pgmname contains the name of the object, &Pgmdesc contains the description of the object, and &RestMethod contains the HTTP method with which the object was called. This variable is empty when the object is called using a protocol other than REST. Also, the &RestCode variable is used to set the HTTP status code, depending on what the service invocation returns. In our example, if we do not find any attraction for the given country, we could assign the &RestCode variable the value 404 (Not found).

Customizing a REST web service with the API object

1) Parameter customization

Service Source | Events | Variables | Help | Documentation

```

1 GetAttractionsInfo{
2
3   GetAttractionsByCountry(in:&CountryId, out:&Attractions)
4   => GetAttractionsByCountryWS(in:&CountryId, out:&Attractions);

```

GetAttractionsInfo X

Service Source | Events | **Variables** | Help | Documentation

Name	Type
& Variables	
& Standard Variables	
● Pgmdesc	Character(256)
● Pgmname	Character(128)
● RestCode	Numeric(3.0)
● RestMethod	HttpMethod, GeneXus
● Attractions	LongVarChar(2M)
● CountryId	Attribute:CountryId
● TripsQty	Numeric(4.0)

Interface Information

External Name	Id
---------------	----

URL: ...GetAttractionsInfo/GetAttractionsByCountry?CountryId=3

↓

URL: ...GetAttractionsInfo/GetAttractionsByCountry?Id=3

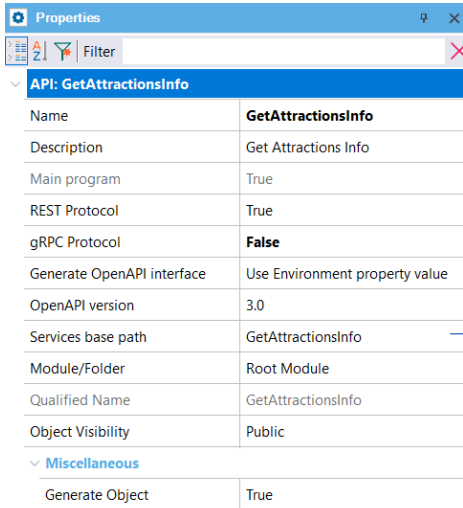
As we said before, the API object allows us to change the name of a parameter. To do so, we must assign the External Name property of the variables used in the parameters of the exposed service.

In the example that we saw, if we go to the variables of the API object GetAttractionsInfo, we select the CountryId variable and in the ExternalName property we write Id.

The URL with which the service will be invoked, instead of using the name CountryId for the parameter, will use the name that we defined as external name; the word Id will appear instead of CountryId.

Customizing a REST web service with the API object

2) Service URL customization



API: GetAttractionsInfo	
Name	GetAttractionsInfo
Description	Get Attractions Info
Main program	True
REST Protocol	True
gRPC Protocol	False
Generate OpenAPI interface	Use Environment property value
OpenAPI version	3.0
Services base path	GetAttractionsInfo
Module/Folder	Root Module
Qualified Name	GetAttractionsInfo
Object Visibility	Public
Miscellaneous	
Generate Object	True

URL:

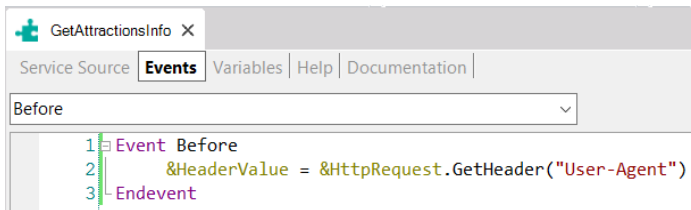
...GetAttractionsInfo/GetAttractionsByCountry?CountryId=3

To change the default URL of an API object, we can do it by changing the value of its Services base path property.

By default, this property has the name of the API object. If we enter another text, that text will appear in the URL instead of the name of the object.

Customizing a REST web service with the API object

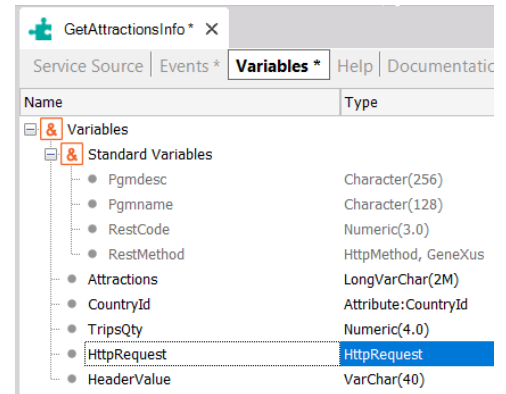
3) Reading parameters from HTTP header Request



```

1 Event Before
2   &HeaderValue = &HttpRequest.GetHeader("User-Agent")
3 Endevent

```



Name	Type
Variables	
Standard Variables	
Pgmdesc	Character(256)
Pgmname	Character(128)
RestCode	Numeric(3.0)
RestMethod	HttpMethod, GeneXus
Attractions	LongVarChar(2M)
CountryId	Attribute:CountryId
TripsQty	Numeric(4.0)
HttpRequest	HttpRequest
HeaderValue	VarChar(40)

Another thing we can do with the API object is to read the data from the header of the invocation to our REST service.

If any parameter is received in the HTTP header of the service, using the Before event, we can read the header that is sent by the application that invokes the service when it makes the Request.

The text expression used between the brackets of the GetHeader method determines the name of the HTTP header. These headers, which are invisible to the end user, define how information is sent or received from the service. For example, who is calling the service, host name, access credentials, connection type, cookies, etc.

In this case, the header is named "User-Agent," and the value returned is a text string that identifies the user agent to the server; in other words, the name of the client application that is invoking the service.

Customizing a REST web service with the API object

4) HTTP access methods customization

```
AttractionsInfo {
  [RestMethod(GET)]
  GetAttraction(in:&AttractionId, out:&AttractionInfo)
    => GetAttractionInfoWS(in:&AttractionId, out:&AttractionInfo);

  [RestMethod(POST)]
  InsertAttraction(in:&AttractionName, in:&AttractionPhoto, in:&CountryId, in:&CityId, in:&CategoryId)
    => CreateAttractionWS(&AttractionName,&AttractionPhoto,&CountryId,&CityId,&CategoryId);
}
```

With the API object, if we include an annotation prior to the definition of the service, we can specify the HTTP access method.

Let's suppose that we have two services, one named `GetAttraction` to obtain data from a specific tourist attraction through a data provider, and another one named `InsertAttraction`, which allows us to create a new attraction to the database through a procedure.

For example, in the case of the `GetAttraction` service, we are specifying that the GET method will be used to invoke it, since we are obtaining information through the service. The annotation goes between straight brackets; it will contain `RestMethod` and between brackets the HTTP method to use—in this case, GET. In the example, the internal method to be executed is that of the data provider `GetAttractionInfoWS`. Since Data Providers return information, they are invoked by default as GET, and with the annotation we are making this explicit.

The `InsertAttraction` method that inserts an attraction into the database is invoked with a POST, using the `RestMethod(POST)` annotation. The `CreateAttractionWS` object, since it is a procedure, can be invoked as GET or as POST; in this case, we invoke it explicitly as POST because it will be saving information in the database.

Customizing a REST web service with the API object

5) Service response customization

```
1 Event GetAttraction.Before
2     if &AttractionId <= 0
3         &RestCode = 412 // HTTP Status Code: Precondition Failed
4         return
5     endif
6 Endevent
7
8 Event GetAttraction.After
9     if &AttractionInfo.&AttractionId = 0
10        &Message.Type = MessageTypes.Error
11        &Message.Description = format("Attraction %1 was not found",&AttractionId)
12        &Messages.Add(&Message)
13        &RestCode = 404 // HTTP Status Code: Not Found
14    endif
15 Endevent
```

By using the standard `&RestCode` variable of the API object, we can change the HTTP status code.

We do this in the events, and we can do it before invoking the service (in a Before event) to set a value in case it is already determined that the service cannot be invoked because the received data is not correct.

In the code, we are assigning the HTTP Status Code 412 Precondition Failed, if the `AttractionId` received as parameter is not valid.

And we can also do it after executing the service, to establish the result of the operation; for example, if the required information was not found.

In the example, the information received has the `AttractionId` field empty, so no information was found for an attraction with the `AttractionId` passed as parameter. Therefore, we assign the `&RestCode` variable with the value 404 (Not Found).

In this video, we saw the flexibility of the API object to publish REST services. To learn more about this object, you can visit the wiki.

GeneXus™

training.genexus.com

wiki.genexus.com

training.genexus.com/certifications