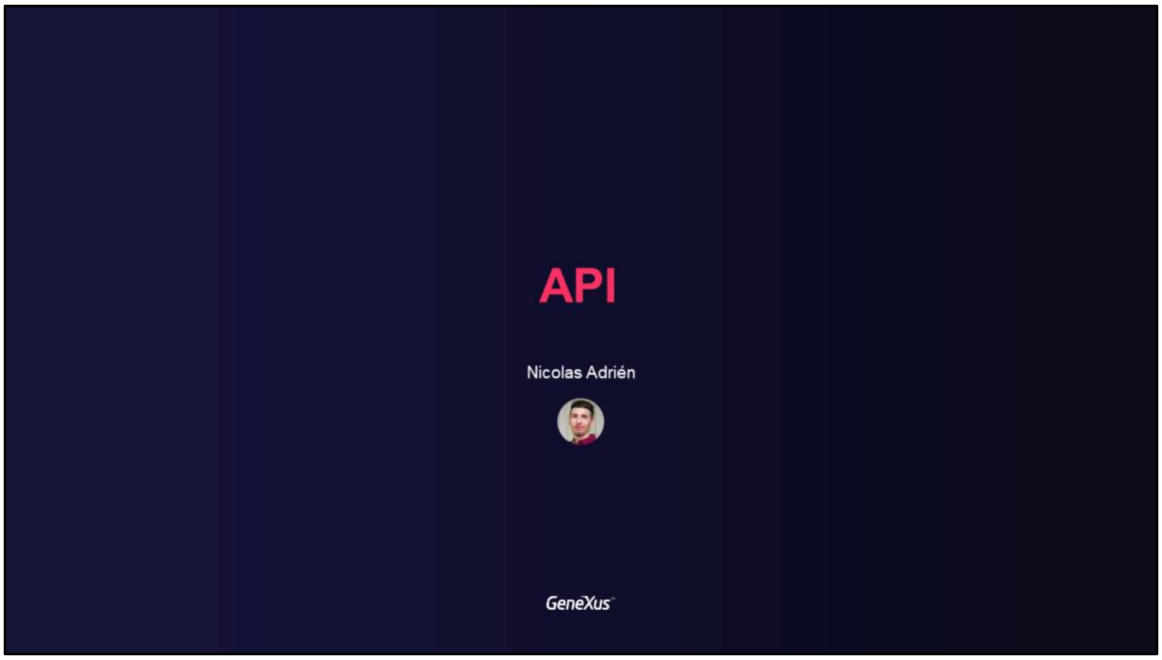
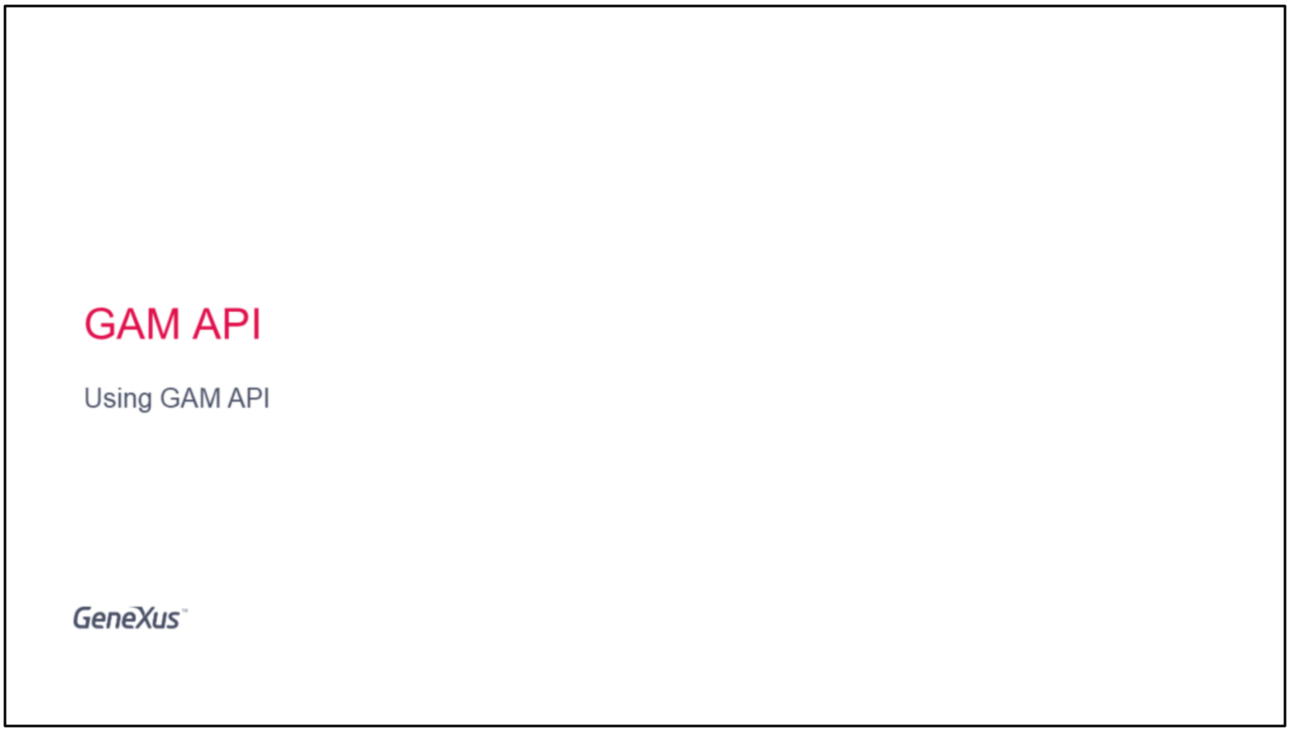
GeneXus™
by Globant



API

Nicolas Adrién

GeneXus

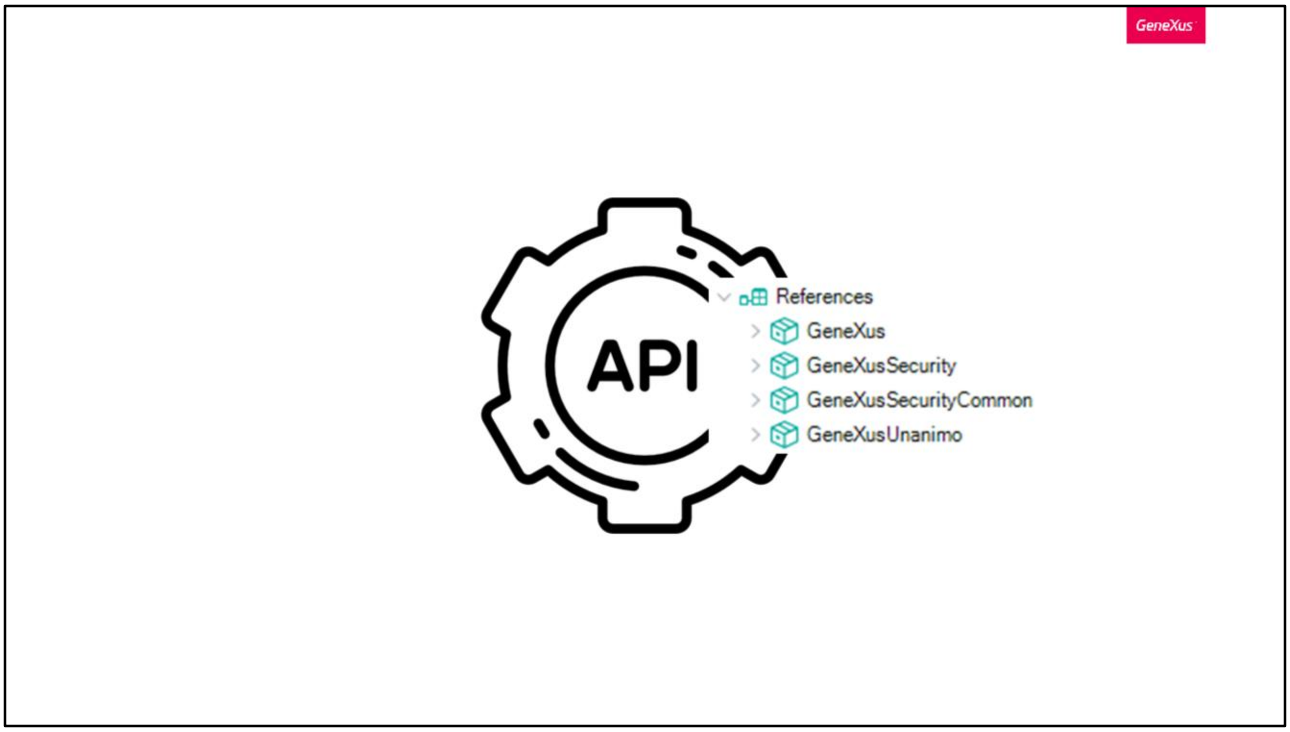


GAM API

Using GAM API

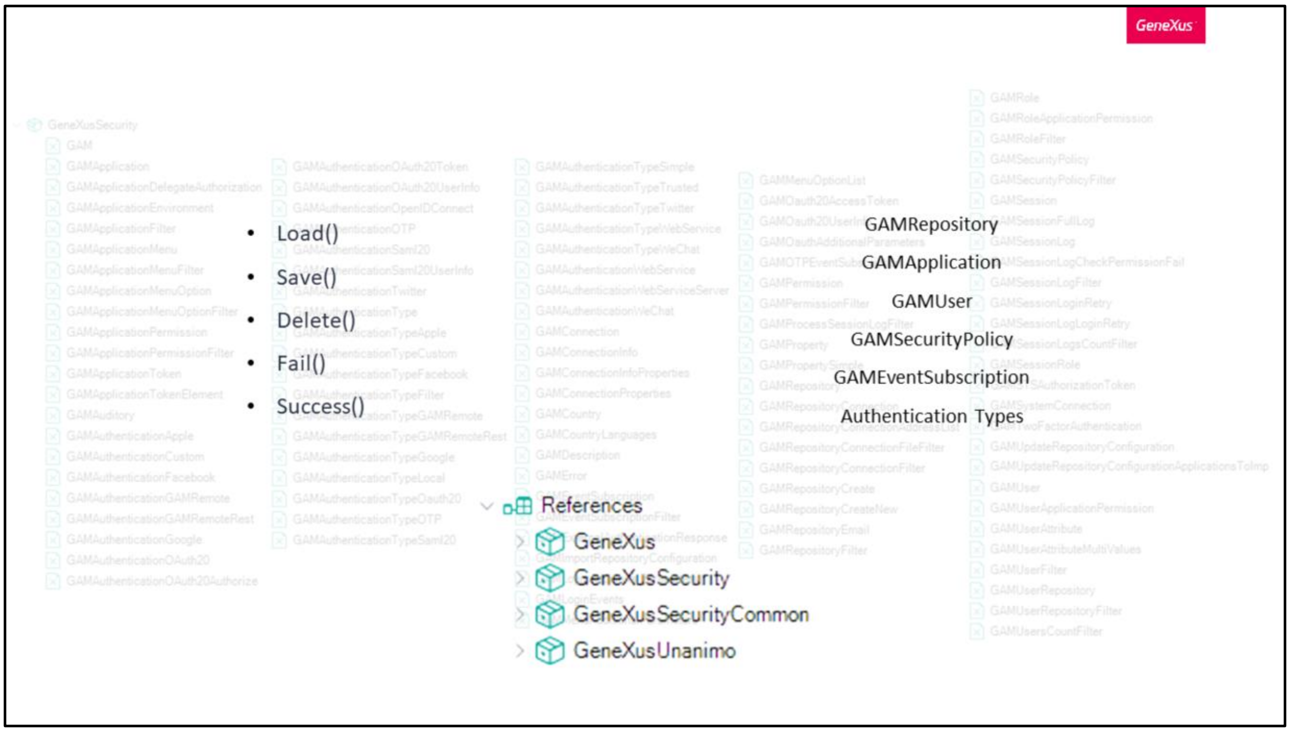
GeneXus™

In this video, we will explain what the GAM API is all about, with its reference methods and implementations, authentication types, and REST services, among others.



GAM provides an API that allows users to handle different types of data and methods to add security to GeneXus applications, both web and mobile.

When integrated security is enabled in the KB, certain External Objects are incorporated to allow the user to interact with the GAM API, where these objects are included.
They can all be found below the module called GeneXusSecurity. Their domains are distributed in the GeneXusSecurityCommon module.

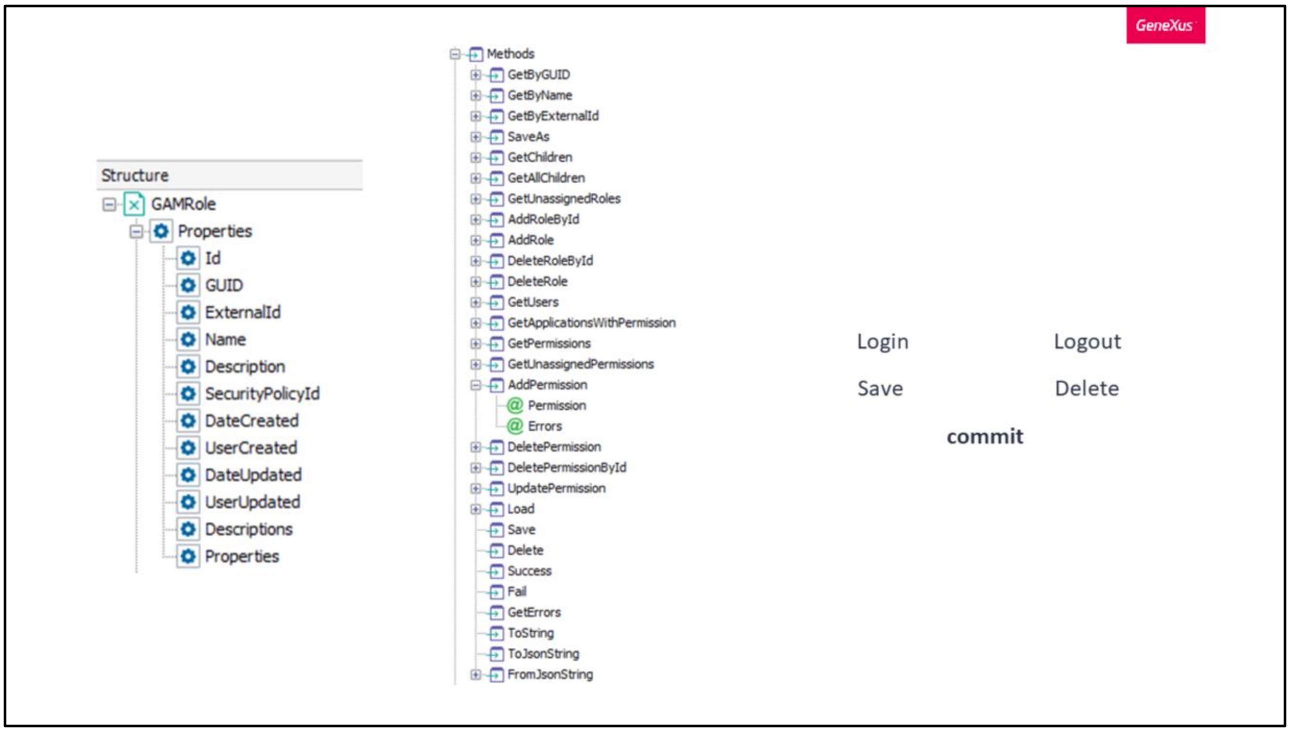


In the introductory course, we saw the list of these External Objects provided by GAM, and showed an example of use for one of them. Now we'll discuss this in more detail.

These objects have properties and methods and, in turn, some of them implement the same methods as Business Components, such as Load, Save, Delete, Fail, and Success.
If a property of one of those GAM objects is changed, it is necessary to call the Save() method and run the commit command.

The objects imported as Business Components are as follows: GAMRepository, GAMApplication, GAMUser, GAMSecurityPolicy, GAMEventSubscription and all authentication types.

As we've said before, all these objects can be found in the GeneXusSecurity module.



```
Structure
GAMRole
  Properties
    Id
    GUID
    ExternalId
    Name
    Description
    SecurityPolicyId
    DateCreated
    UserCreated
    DateUpdated
    UserUpdated
    Descriptions
    Properties
```

```
Methods
  GetByGUID
  GetByName
  GetByExternalId
  SaveAs
  GetChildren
  GetAllChildren
  GetUnassignedRoles
  AddRoleById
  AddRole
  DeleteRoleById
  DeleteRole
  GetUsers
  GetApplicationsWithPermission
  GetPermissions
  GetUnassignedPermissions
  AddPermission
    Permission
    Errors
  DeletePermission
  DeletePermissionById
  UpdatePermission
  Load
  Save
  Delete
  Success
  Fail
  GetErrors
  ToString
  ToJsonString
  FromJsonString
```

Login Logout

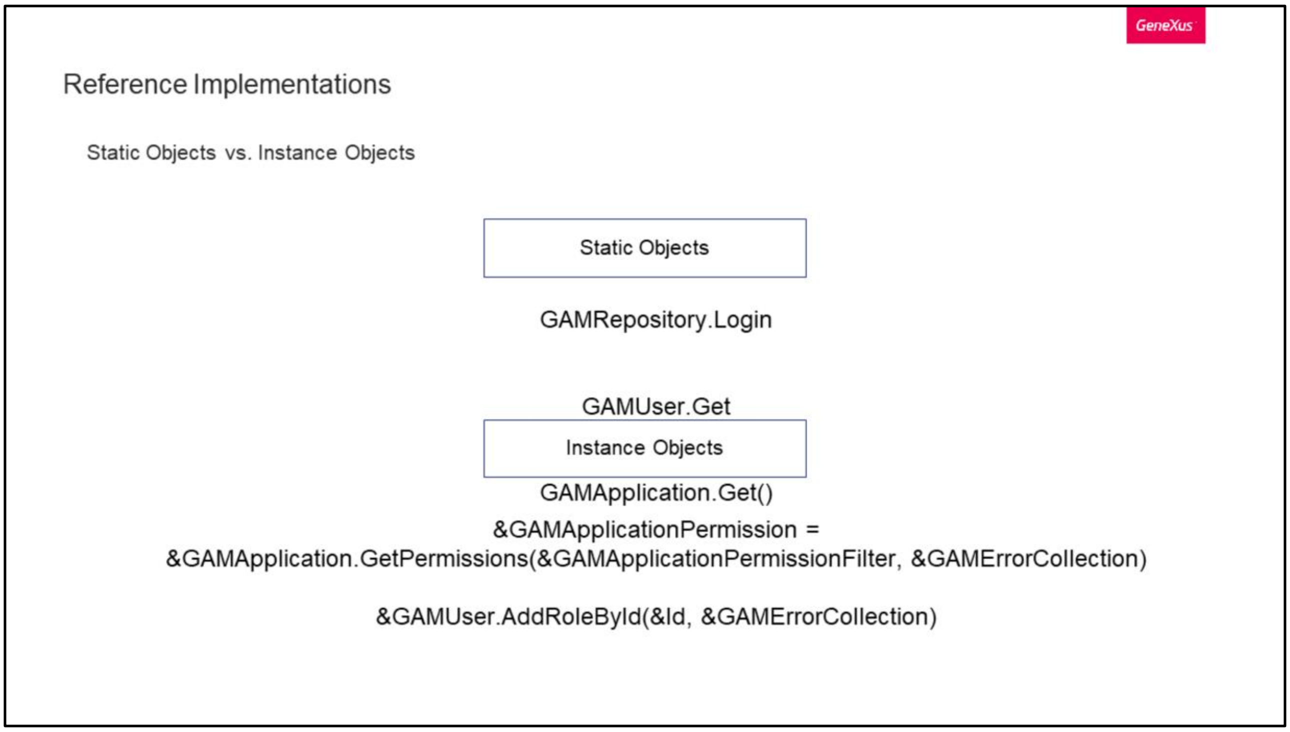
Save Delete

commit

Here is the GAMRole object as an example.
Remember that these objects are composed of properties and methods.

In turn, GAM objects also have other methods implemented to create, update or delete objects.
When using them, the Commit command must be used after the method has been correctly executed. This command, just like in Business Components, is also used after a Save or Delete.

The only GAM methods that execute an implicit commit are those related to login and logout, and they are executed in a new logical unit of work, where it is not necessary to do so since GAM internally generates the various inserts and commits them.



Reference Implementations

Static Objects vs. Instance Objects

Static Objects

GAMRepository.Login

GAMUser.Get

Instance Objects

GAMApplication.Get()

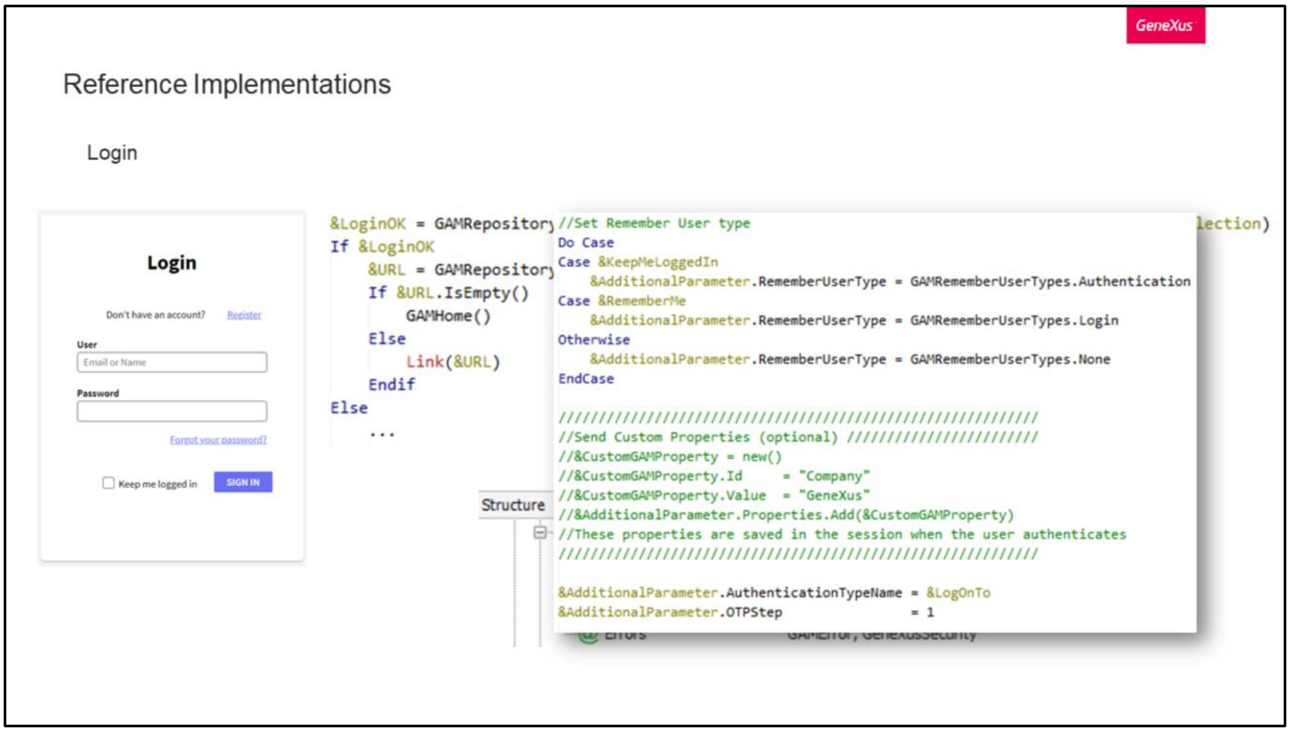
&GAMApplicationPermission =
&GAMApplication.GetPermissions(&GAMApplicationPermissionFilter, &GAMErrorCollection)

&GAMUser.AddRoleById(&Id, &GAMErrorCollection)

Within the reference implementations, we can find uses of static External Objects and instance objects.
The difference between them is that the static ones are methods that apply when the object does not have the &.

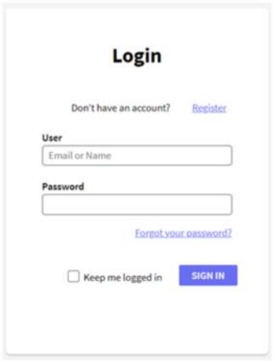
Some examples of static objects can be as follows:
GAMRepository.Login ==> With this we can log into the current repository
GAMUser.Get ==> Obtain information about the user currently logged in
GAMApplication.Get() ==> Obtain information about the current application

Examples of instance objects include:
&GAMApplication.GetPermissions ==> Where the & indicates that the method will be applied on the loaded object instance, which in this case is GAMApplication. With this we obtain an application's permissions.
&GAMUser.AddRoleById, where it receives an identifier and a collection of errors. With this we add a Role to the user previously loaded (which was instantiated).



Reference Implementations

Login



```
&LoginOK = GAMRepository          ...lection)
If &LoginOK
    &URL = GAMRepository
    If &URL.IsEmpty()
        GAMHome()
    Else
        Link(&URL)
    Endif
Else
    ...
```

Structure

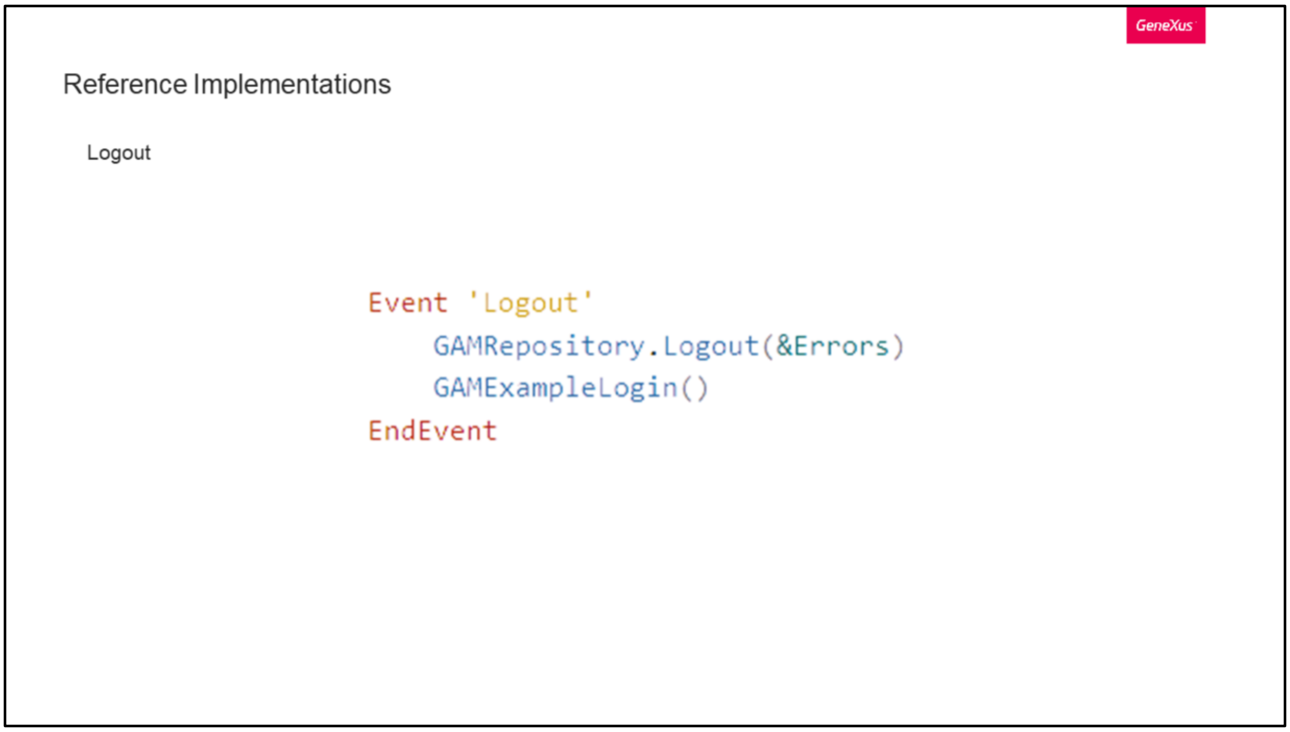
```
//Set Remember User type
Do Case
Case &KeepMeLoggedIn
    &AdditionalParameter.RememberUserType = GAMRememberUserTypes.Authentication
Case &RememberMe
    &AdditionalParameter.RememberUserType = GAMRememberUserTypes.Login
Otherwise
    &AdditionalParameter.RememberUserType = GAMRememberUserTypes.None
EndCase

/////////////////////////////////////////////////////////////
//Send Custom Properties (optional) ////////////////////////
//&CustomGAMProperty = new()
//&CustomGAMProperty.Id      = "Company"
//&CustomGAMProperty.Value   = "GeneXus"
//&AdditionalParameter.Properties.Add(&CustomGAMProperty)
//These properties are saved in the session when the user authenticates
/////////////////////////////////////////////////////////////

&AdditionalParameter.AuthenticationTypeName = &LogOnTo
&AdditionalParameter.OTPStep                = 1
```

Now let's look at two reference implementations of our own, such as Login and Logout. Let's start with Login. For this, we have the GAMExampleLogin object.

When pressing the login button, the implementation is based on executing the Login method of the External Objects: GAMRepository.
As we can see, the Login method of this object already provided by GAM is being used. The parameters included in the call are:
Username and password, obtained from the WebPanel.
Additional parameters: They can be the option to keep the session active, remember the user, or another case. In addition, we can create custom properties and associate them with the user's session as shown on the screen. We can also send it the type of authentication used, OTP-related information, etc.
Error collection: If there are any errors, they are loaded into this variable for later review.

The result of the operation is stored in the Boolean variable LoginOK. The action to be performed depends on this value.
If it is true, the resulting value of the GetLastErrorsURL method is stored in the URL variable. This returns the URL where the last GAM error occurred and once it is executed, its value is removed. This is useful when users try to access an object with authentication and they can't because they are not logged in, so they are first redirected to the login and from there they are directed back to the object in question,

now with a valid session.
Otherwise, they are simply redirected to the Home defined in the application.

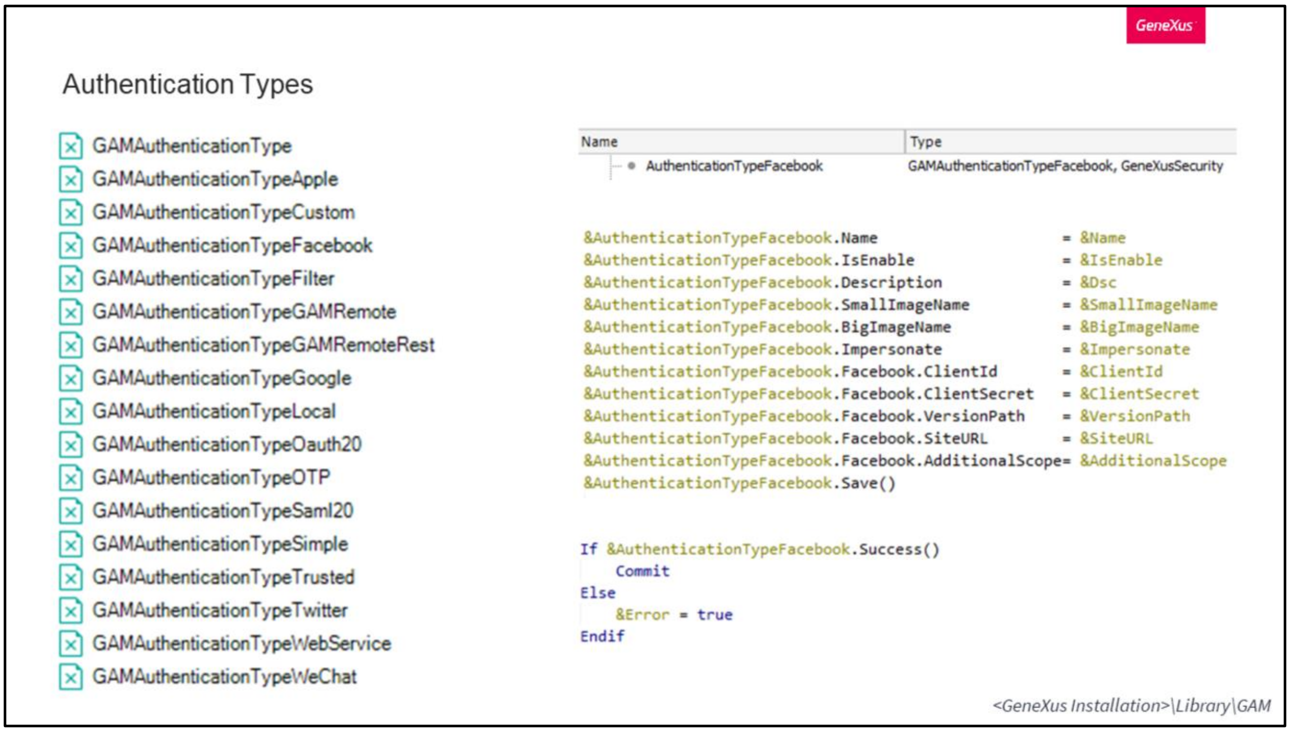


Reference Implementations

Logout

```
Event 'Logout'
    GAMRepository.Logout(&Errors)
    GAMExampleLogin()
EndEvent
```

Logout.
This can be implemented in a very simple way, as shown below.
With the first instruction, the current session is ended and if there are any errors, they are saved in the &Errors SDT.
The second instruction ends the flow by taking the user to the GAMExampleLogin Web Panel or, when it is not available, to the Web Panel used for login.
This last instruction should only be performed in cases where this application is NOT an IDP client. In the event that it is, there is no need to call anything after logout to close the IDP session.



Authentication Types

- GAMAuthenticationType
- GAMAuthenticationTypeApple
- GAMAuthenticationTypeCustom
- GAMAuthenticationTypeFacebook
- GAMAuthenticationTypeFilter
- GAMAuthenticationTypeGAMRemote
- GAMAuthenticationTypeGAMRemoteRest
- GAMAuthenticationTypeGoogle
- GAMAuthenticationTypeLocal
- GAMAuthenticationTypeOauth20
- GAMAuthenticationTypeOTP
- GAMAuthenticationTypeSaml20
- GAMAuthenticationTypeSimple
- GAMAuthenticationTypeTrusted
- GAMAuthenticationTypeTwitter
- GAMAuthenticationTypeWebService
- GAMAuthenticationTypeWeChat

Name	Type
AuthenticationTypeFacebook	GAMAuthenticationTypeFacebook, GeneXusSecurity

```
&AuthenticationTypeFacebook.Name                    = &Name
&AuthenticationTypeFacebook.IsEnable                = &IsEnable
&AuthenticationTypeFacebook.Description             = &Dsc
&AuthenticationTypeFacebook.SmallImageName          = &SmallImageName
&AuthenticationTypeFacebook.BigImageName            = &BigImageName
&AuthenticationTypeFacebook.Impersonate             = &Impersonate
&AuthenticationTypeFacebook.Facebook.ClientId       = &ClientId
&AuthenticationTypeFacebook.Facebook.ClientSecret   = &ClientSecret
&AuthenticationTypeFacebook.Facebook.VersionPath    = &VersionPath
&AuthenticationTypeFacebook.Facebook.SiteURL        = &SiteURL
&AuthenticationTypeFacebook.Facebook.AdditionalScope= &AdditionalScope
&AuthenticationTypeFacebook.Save()


If &AuthenticationTypeFacebook.Success()
    Commit
Else
    &Error = true
Endif
```

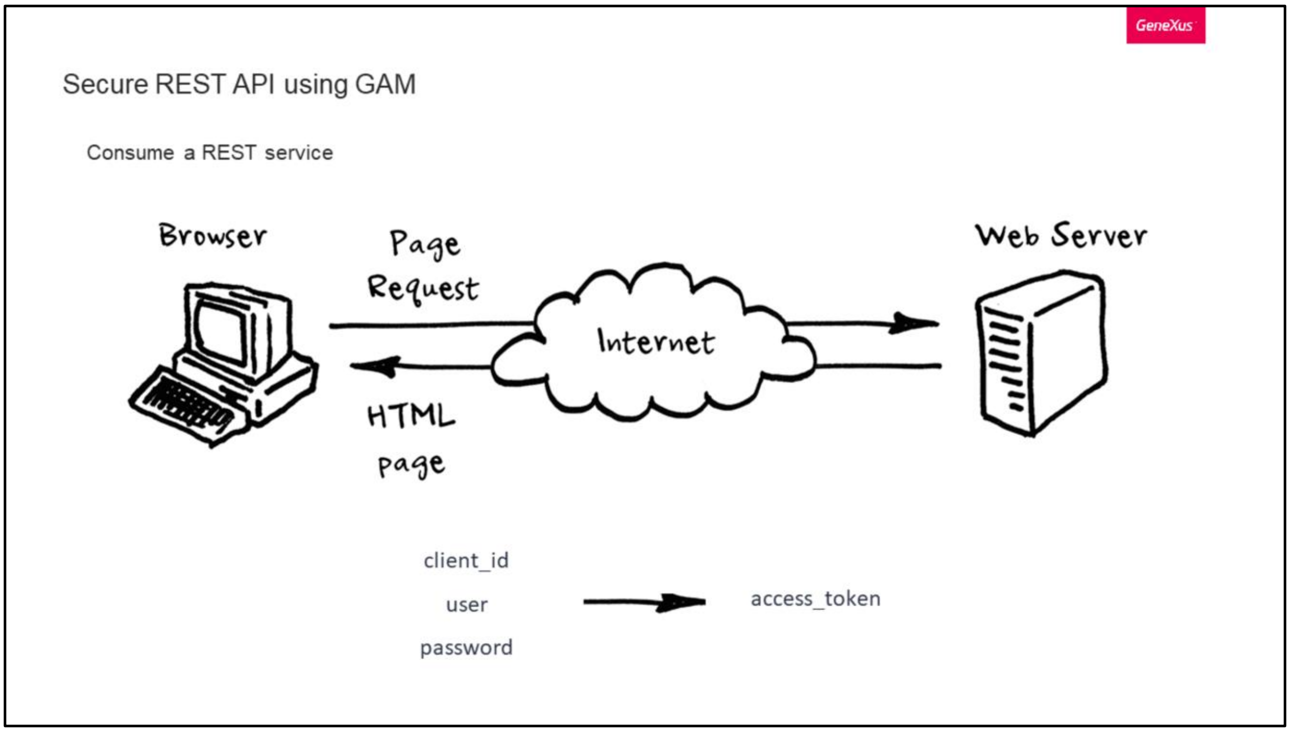
<GeneXus Installation>\Library\GAM

In the introductory course, we also addressed the types of authentication from the GAM web back office, but we mentioned that this is also possible through the GAM API.
To do so, we have the External Objects shown on the screen, one for each authentication type supported by GAM.

Let's take the example of the Facebook authentication type.
If we wanted to create one of these, first we should create a variable of this type.
After that, we start filling in the different attributes of the authentication type, according to our data.
If the authentication type is successful, we perform a commit to persist the data.

That’s all.

In the GAM API examples, you can find this and all the other examples for each authentication type, including insertion, modification and deletion.
These examples are located in the GeneXus installation folder inside the Library/GAM directory.



Secure REST API using GAM

Consume a REST service

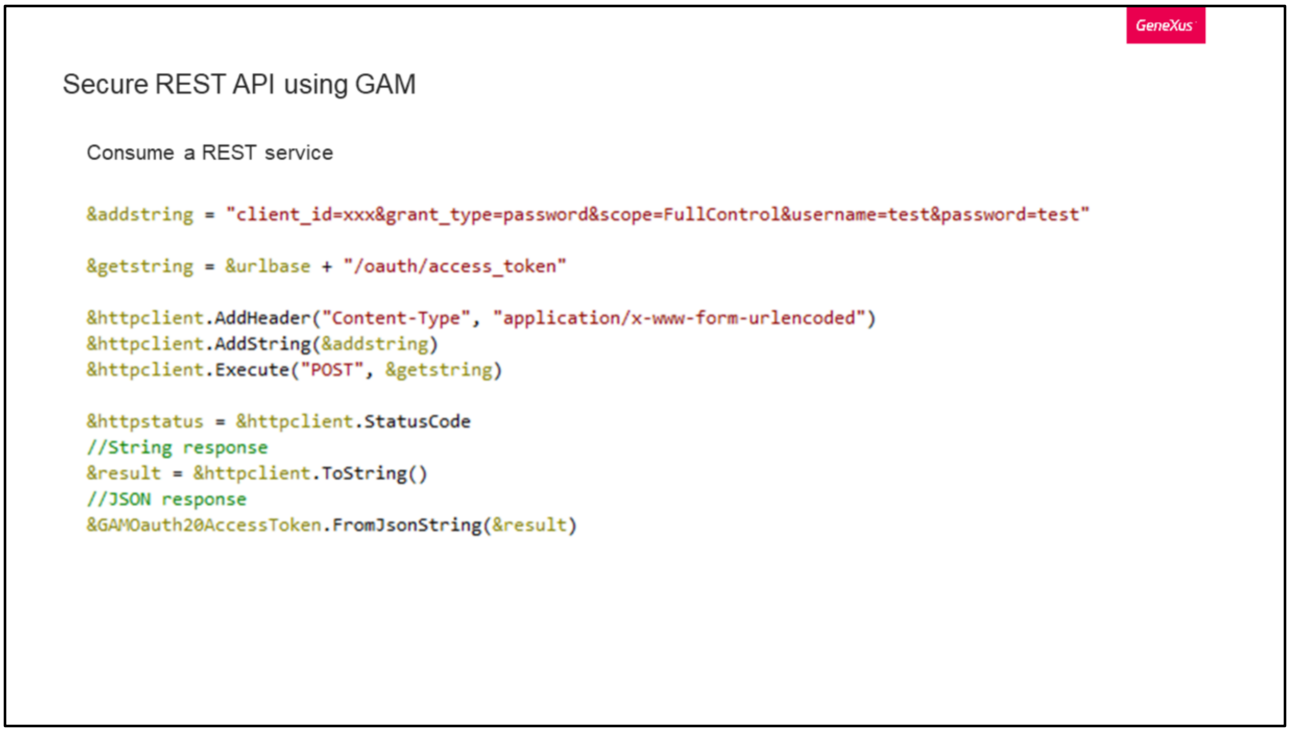
REST Services.

It is very common for applications to expose REST services, so security is very important when data privacy is a concern.
In GeneXus, as we have already said, the solution to this is to use GAM, where OAuth technology is used in the background, making it easier for the user to deal with the complexity it contains.

The GAM API provides a way to restrict user access to REST services defined in the application.

To consume a REST service in GeneXus, it is necessary to have the client_Id of the Application, and the username and password with access permissions to this application.
Before consuming the web service, you must first obtain an access_token.

Let's see a possible implementation of a procedure that consumes it.



Secure REST API using GAM

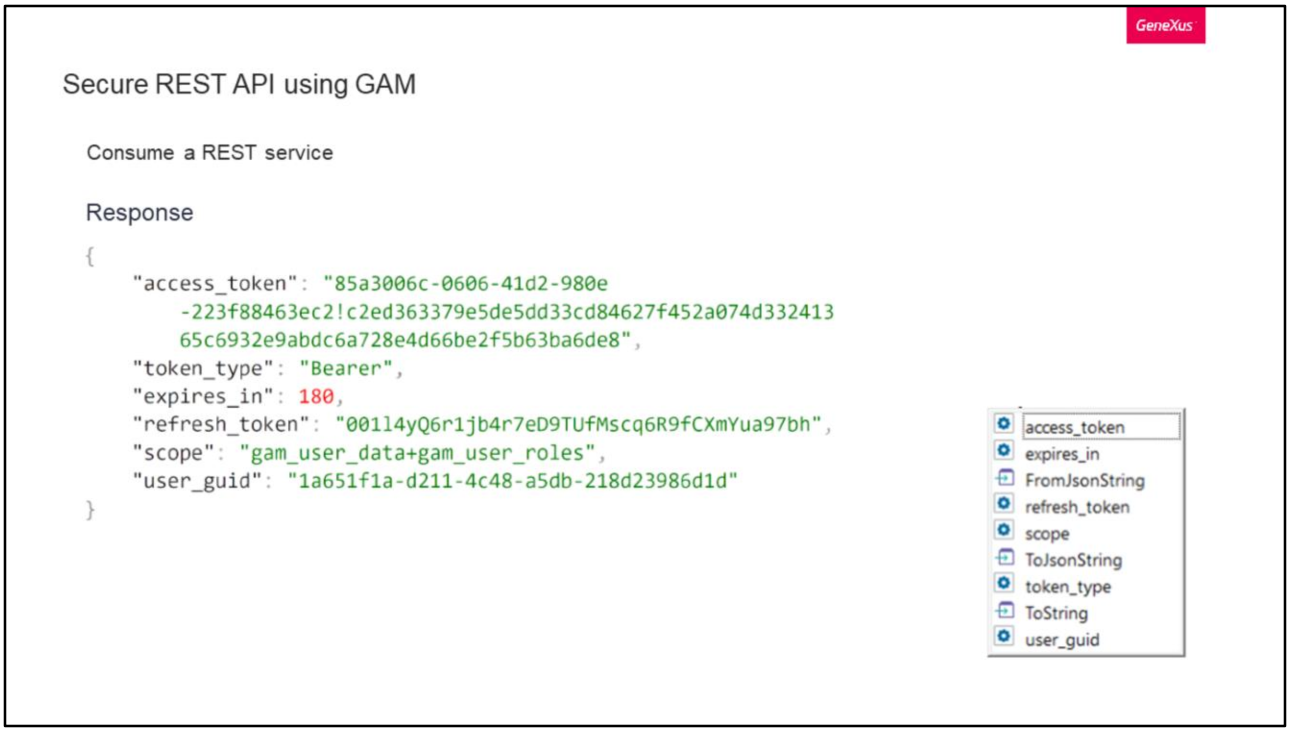
Consume a REST service

```
&addstring = "client_id=xxx&grant_type=password&scope=FullControl&username=test&password=test"

&getstring = &urlbase + "/oauth/access_token"

&httpclient.AddHeader("Content-Type", "application/x-www-form-urlencoded")
&httpclient.AddString(&addstring)
&httpclient.Execute("POST", &getstring)

&httpstatus = &httpclient.StatusCode
//String response
&result = &httpclient.ToString()
//JSON response
&GAMOauth20AccessToken.FromJsonString(&result)
```

The first string is composed of:

- Client_id of the application
- Grant_type of the flow
- Scope of the user account you want to access
- Username and password of the account you want to access

The URL to be consumed will be the application's base, followed by /oauth/access_token.
The Content-Type header must be added.
Also, we execute a POST request with this information through the httpclient.



Secure REST API using GAM

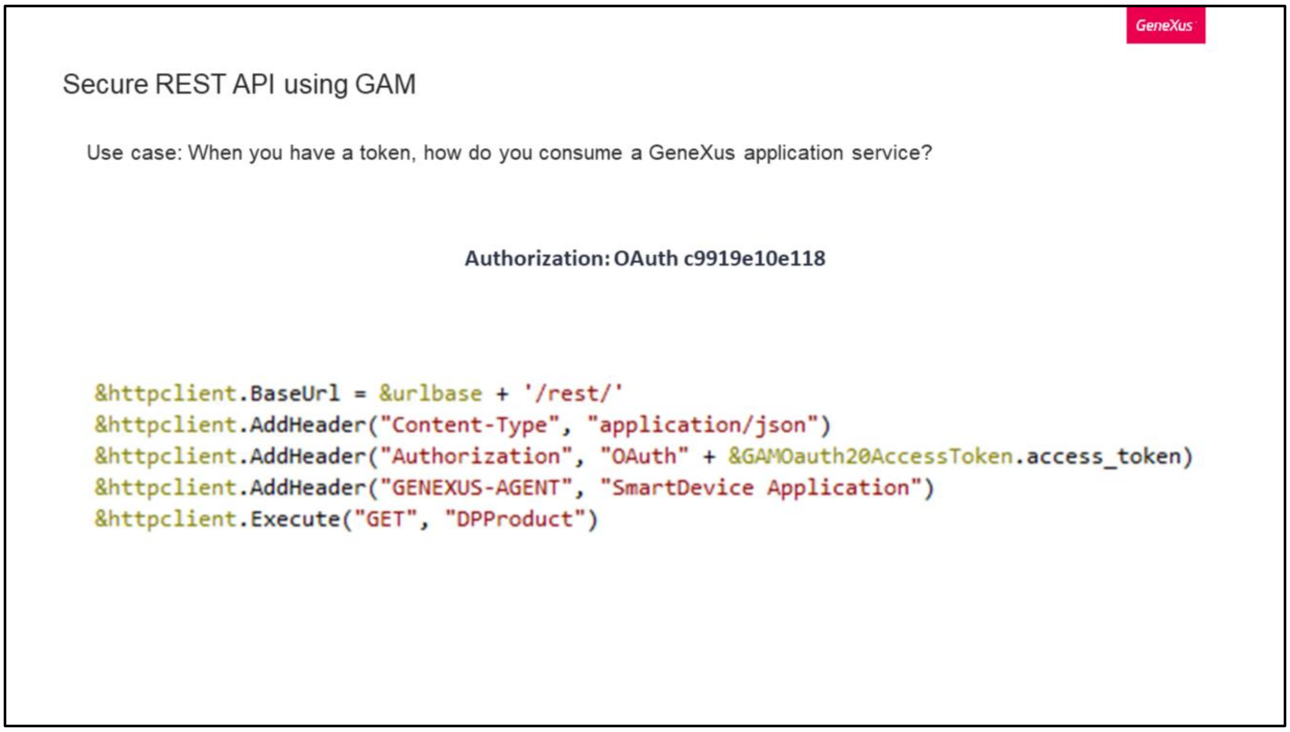
Consume a REST service

Response

```
{
    "access_token": "85a3006c-0606-41d2-980e
        -223f88463ec2!c2ed363379e5de5dd33cd84627f452a074d332413
        65c6932e9abdc6a728e4d66be2f5b63ba6de8",
    "token_type": "Bearer",
    "expires_in": 180,
    "refresh_token": "001l4yQ6r1jb4r7eD9TUfMscq6R9fCXmYua97bh",
    "scope": "gam_user_data+gam_user_roles",
    "user_guid": "1a651f1a-d211-4c48-a5db-218d23986d1d"
}
```

To read the response, we can do it either in a string or in a JSON. When using the latter, we have the following attributes to be able to read from the response programmatically.

If the information is correct, the response of the request should look as follows, providing an access_token in a JSON and informing about the scope.



Secure REST API using GAM

Use case: When you have a token, how do you consume a GeneXus application service?

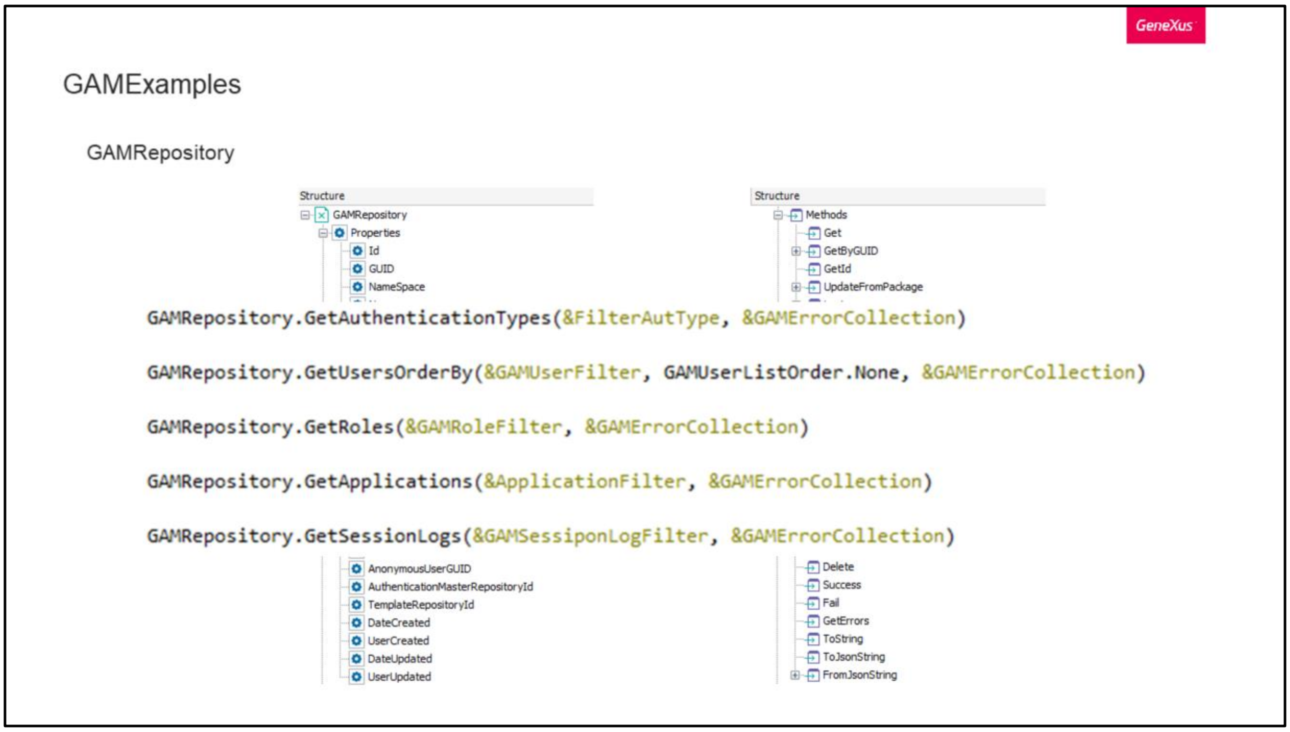
Authorization: OAuth c9919e10e118

```
&httpclient.BaseUrl = &urlbase + '/rest/'
&httpclient.AddHeader("Content-Type", "application/json")
&httpclient.AddHeader("Authorization", "OAuth" + &GAMOauth20AccessToken.access_token)
&httpclient.AddHeader("GENEXUS-AGENT", "SmartDevice Application")
&httpclient.Execute("GET", "DPProduct")
```

Now suppose we already have the access_token we saw earlier; how do we proceed to actually consume the service we want?

To do this, each call must contain the access_token included in the Authorization header of our requests, after the OAuth word as shown on the screen.

An example of code can be the following, where we show how to get a list of products.
We indicate the URL, and add the headers which include the one we said before with the access_token.
Finally, we execute the request, which in this case is a GET to DPProduct. As we said, it represents a list of products exposed as a REST web service, where this is a secure service because GAM would be enabled in its KB.



GAMExamples

GAMRepository

```
GAMRepository.GetAuthenticationTypes(&FilterAutType, &GAMErrorCollection)

GAMRepository.GetUsersOrderBy(&GAMUserFilter, GAMUserListOrder.None, &GAMErrorCollection)

GAMRepository.GetRoles(&GAMRoleFilter, &GAMErrorCollection)

GAMRepository.GetApplications(&ApplicationFilter, &GAMErrorCollection)

GAMRepository.GetSessionLogs(&GAMSessiponLogFilter, &GAMErrorCollection)
```

To close the theoretical part of this topic, we will see the most common External Objects used in GeneXus applications.
Let's start with GAMRepository.

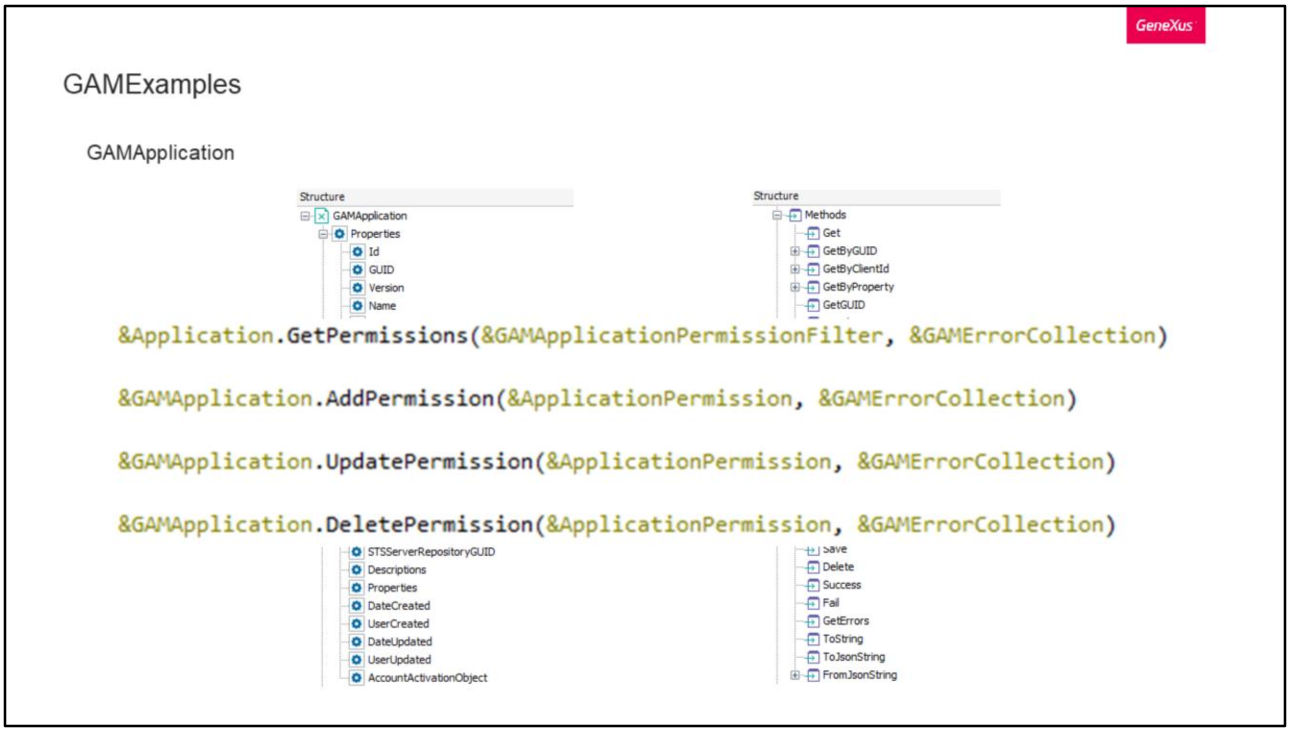
This object is quite large, both in terms of its properties and methods. The most common options it provides are everything directly related to a repository, login, users, sessions, roles, applications, etc.

Let's see some of the most common methods.

First we have GetAuthenticationTypes. This method is used to obtain the authentication types of a repository.
Then we have GetUsersOrderBy, which obtains all the users according to the preferences passed as filters, where we can also define an order for the return list (corresponding to the domain GAMUserListOrder).
GetRoles and GetApplications correspond to methods to obtain the roles and applications of a repository.
Lastly, we have GetSessionLogs which can be used to obtain a log of all the sessions of a repository, with the possibility to filter, for example, by the active sessions.

As we can see, all methods always include the same parameters:
Those ending in Filter are useful to pass filters to the query to be performed on the

database. Each one corresponds to another External Object with its specification. They can be reviewed before using them to define filters that meet our needs.
The other parameter corresponds to the various errors that the method may cause when used, which can then be read to verify the result of the operation.



GAMExamples

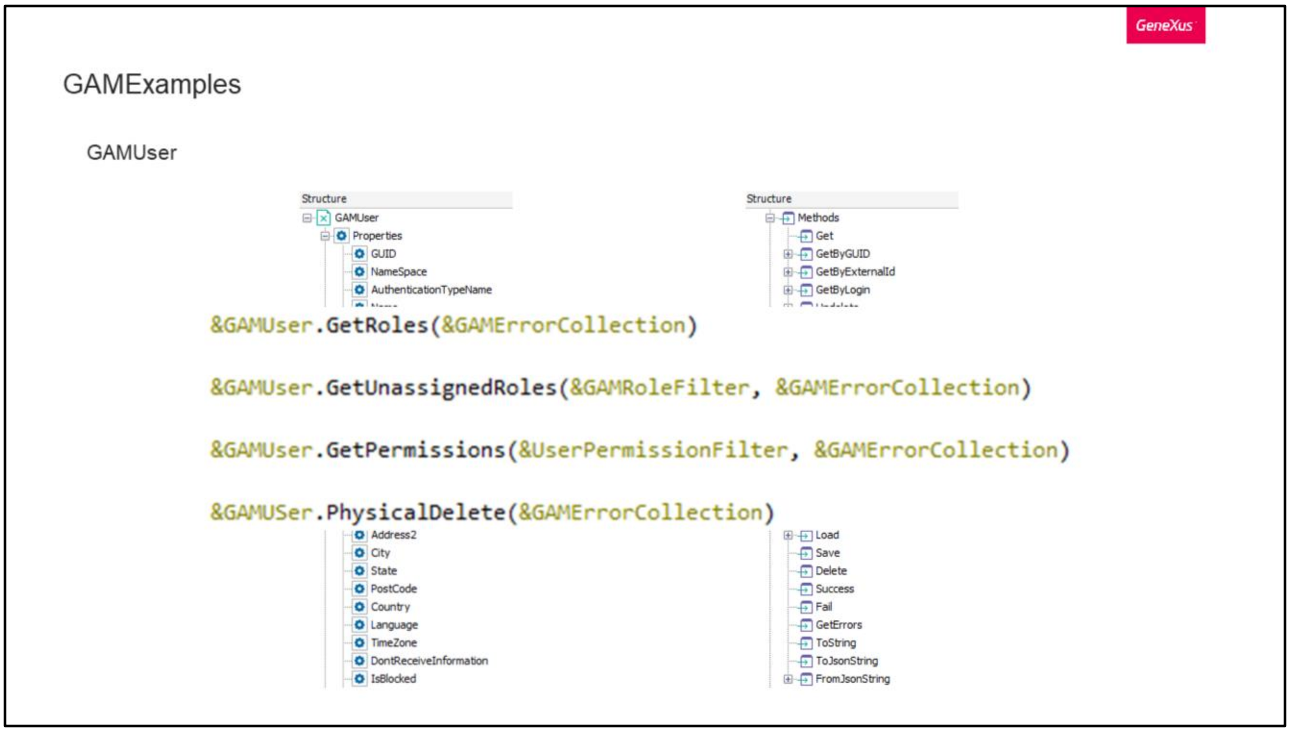
GAMApplication

```
&Application.GetPermissions(&GAMApplicationPermissionFilter, &GAMErrorCollection)

&GAMApplication.AddPermission(&ApplicationPermission, &GAMErrorCollection)

&GAMApplication.UpdatePermission(&ApplicationPermission, &GAMErrorCollection)

&GAMApplication.DeletePermission(&ApplicationPermission, &GAMErrorCollection)
```

GAMApplication.

With this object we will have control over the applications defined in GAM.

Let's see four of its most common methods.
First we have GetPermissions, which allows us to obtain the list of permissions defined in an application. In its first parameter, we will indicate the filtering we want to do in the result list.
The other three parameters are used to add, modify or delete permissions of an application. In this method, through the first parameter we must indicate the data of the permission to which we will apply one of these functions.



GAMExamples

GAMUser

```
Structure
GAMUser
  Properties
    GUID
    NameSpace
    AuthenticationTypeName
    Address2
    City
    State
    PostCode
    Country
    Language
    TimeZone
    DontReceiveInformation
    IsBlocked
```

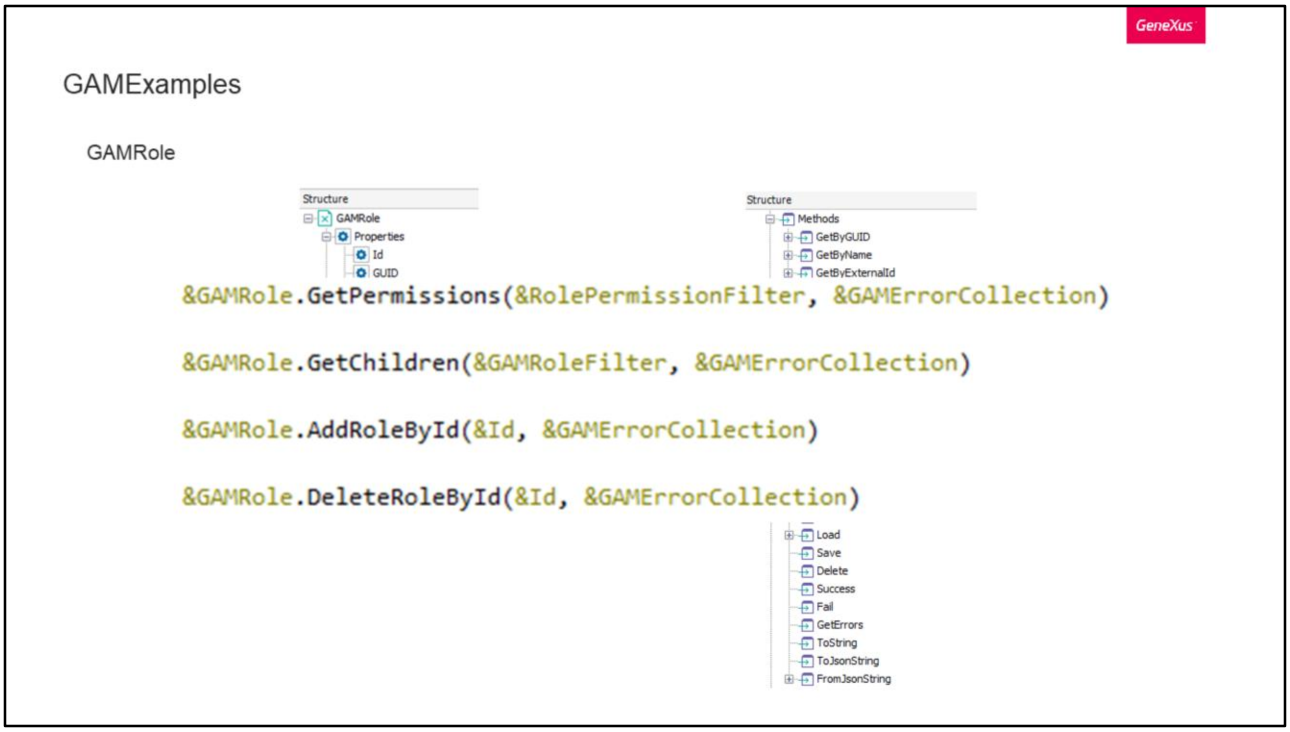
```
Structure
Methods
  Get
  GetByGUID
  GetByExternalId
  GetByLogin
  Load
  Save
  Delete
  Success
  Fail
  GetErrors
  ToString
  ToJsonString
  FromJsonString
```

```
&GAMUser.GetRoles(&GAMErrorCollection)

&GAMUser.GetUnassignedRoles(&GAMRoleFilter, &GAMErrorCollection)

&GAMUser.GetPermissions(&UserPermissionFilter, &GAMErrorCollection)

&GAMUSer.PhysicalDelete(&GAMErrorCollection)
```

GAMUser.

With this object, we have full control of a GAM user. Its methods include the ability to obtain all the information of each one and to control their roles, permissions, and sessions.

Among the most common ones we find the typical GetRoles to obtain the roles of a user.
Then we have GetUnassignedRoles, which would be the opposite of the previous method. It returns the roles that the user is not assigned.
GetPermissions returns the permissions associated with the user. As in the previous method, we have the filter variable.
Finally, PhysicalDelete physically deletes users and all their related data.



GAMExamples

GAMRole

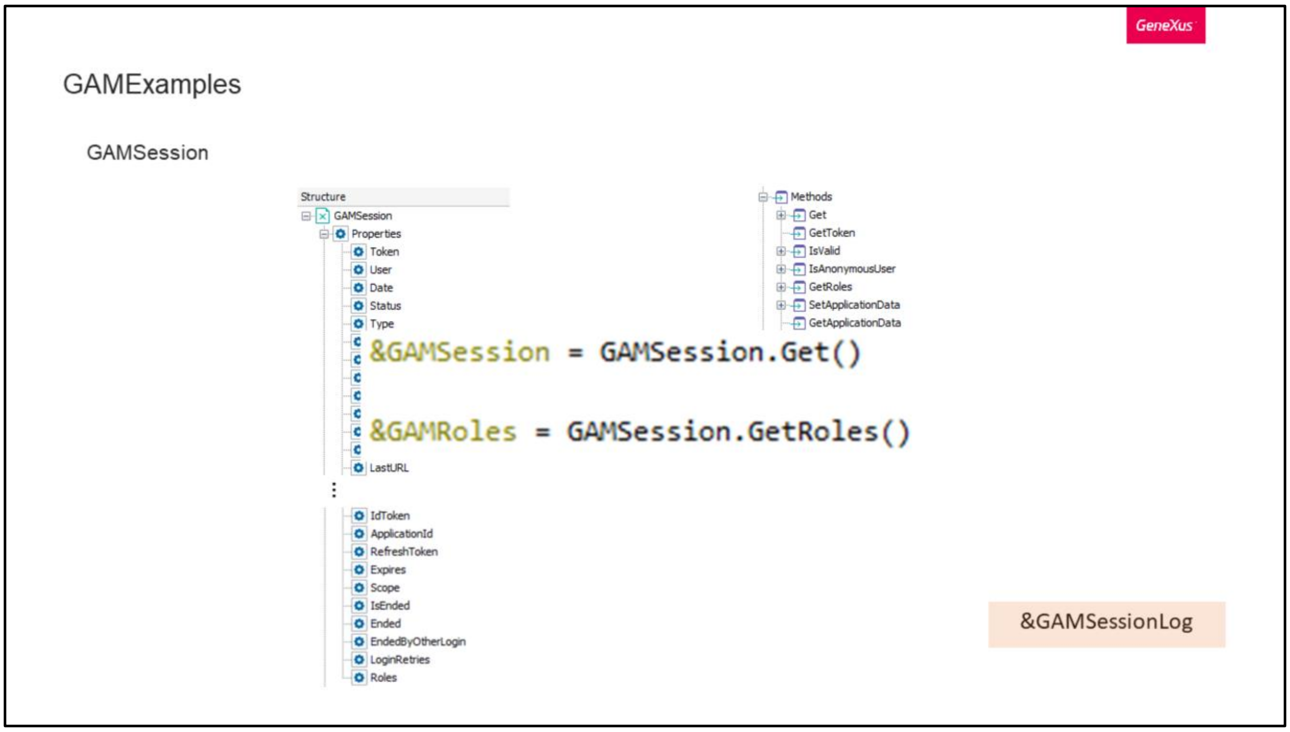
```
&GAMRole.GetPermissions(&RolePermissionFilter, &GAMErrorCollection)

&GAMRole.GetChildren(&GAMRoleFilter, &GAMErrorCollection)

&GAMRole.AddRoleById(&Id, &GAMErrorCollection)

&GAMRole.DeleteRoleById(&Id, &GAMErrorCollection)
```

GAMRole.

With this object, we can manage GAM roles. Its most outstanding methods include the ability to obtain, add and delete them, as well as to control their permissions, among others.

The most common methods are as follows:
With GetPermissions we will obtain all the permissions associated with the role we have in the GAMRole variable.
With GetChildren we will obtain all the children roles contained in the identified role.
In both methods we can use the filtering that we have mentioned for all the other methods.

Next, we have Add and Delete roles by identifier, which basically do what the name of the methods say, except that they work with the child roles of the role to which they are instantiated. In the Id parameter we send the identifier of the role to add or delete.



GAMExamples

GAMSession

```
&GAMSession = GAMSession.Get()

&GAMRoles = GAMSession.GetRoles()
```

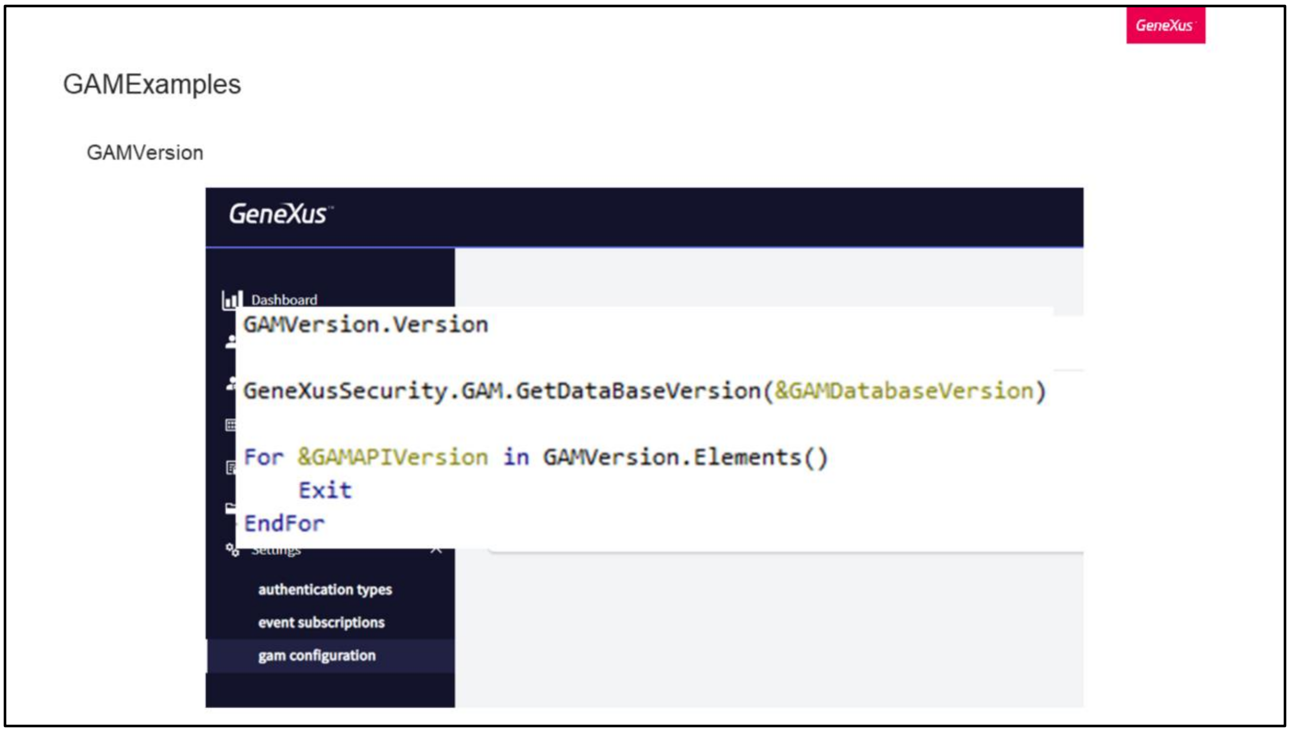
&GAMSessionLog

Let's continue with GAMSession.

With this object we can control all sessions. We can obtain all the attributes of a session and its roles, among other things.

Let's see two of its methods:
To load the data of the current session, we do it as follows by using GAMSession.Get() and storing the result in the &GAMSession variable. This variable is only for the current session.
If, for example, we wanted to see the history of all sessions, we must use the &GAMSessionLog object.

In the same way we load the data of the current session, we can load the roles of a session using the GetRoles method.


GAMExamples

GAMVersion

```
GAMVersion.Version

GeneXusSecurity.GAM.GetDataBaseVersion(&GAMDatabaseVersion)

For &GAMAPIVersion in GAMVersion.Elements()
    Exit
EndFor
```

Finally we have GAMVersion; although it is a domain and not an object, it can be useful to know it.

There are two ways to obtain the GAM version:
The first one is through the web back office, as we see on screen, in the GAM Configuration option within Settings.
The second way is through GeneXus code, and for this we have different options:

- To find out the API version, we can do the following:
    1. First we can query it with the domain, as shown below.
    2. Another option is to perform the following For, where the GAMAPIVersion variable of GAMVersion type will have the version we are looking for.

- To find out the database structure version, we can make a query to the database like the one shown on the screen; this returns a Boolean indicating the result, and the version number is recorded in the variable GAMDatabaseVersion.

Something to keep in mind is that the database version may be newer than the GAM API version.

GeneXus™ by Globant

training.genexus.com
wiki.genexus.com