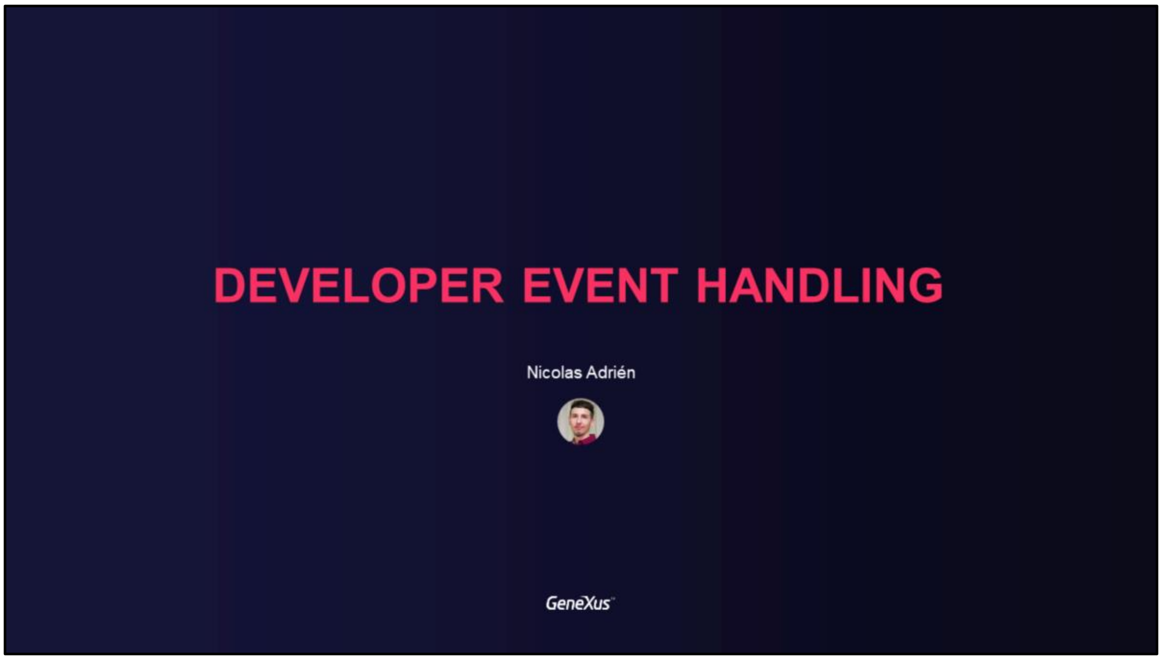
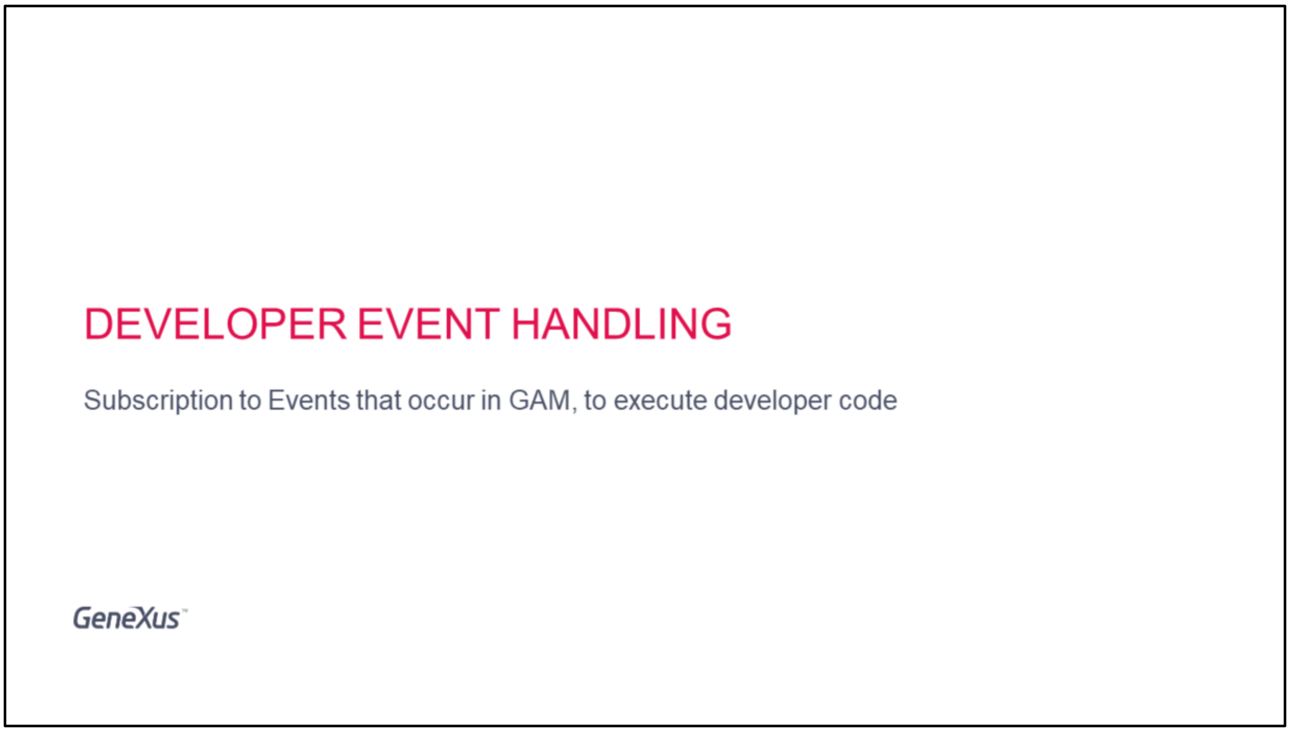
GeneXus™ by Globant



# DEVELOPER EVENT HANDLING

Nicolas Adrién

GeneXus

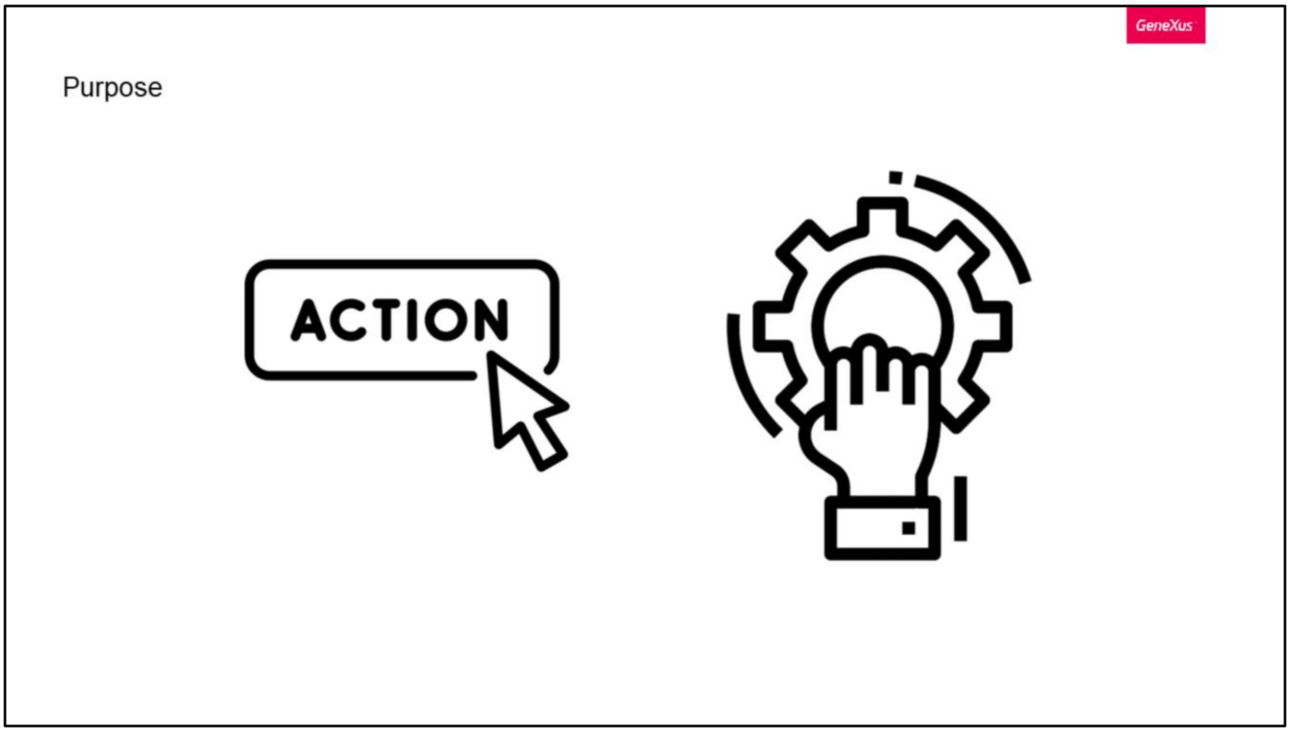


## DEVELOPER EVENT HANDLING

Subscription to Events that occur in GAM, to execute developer code

GeneXus™

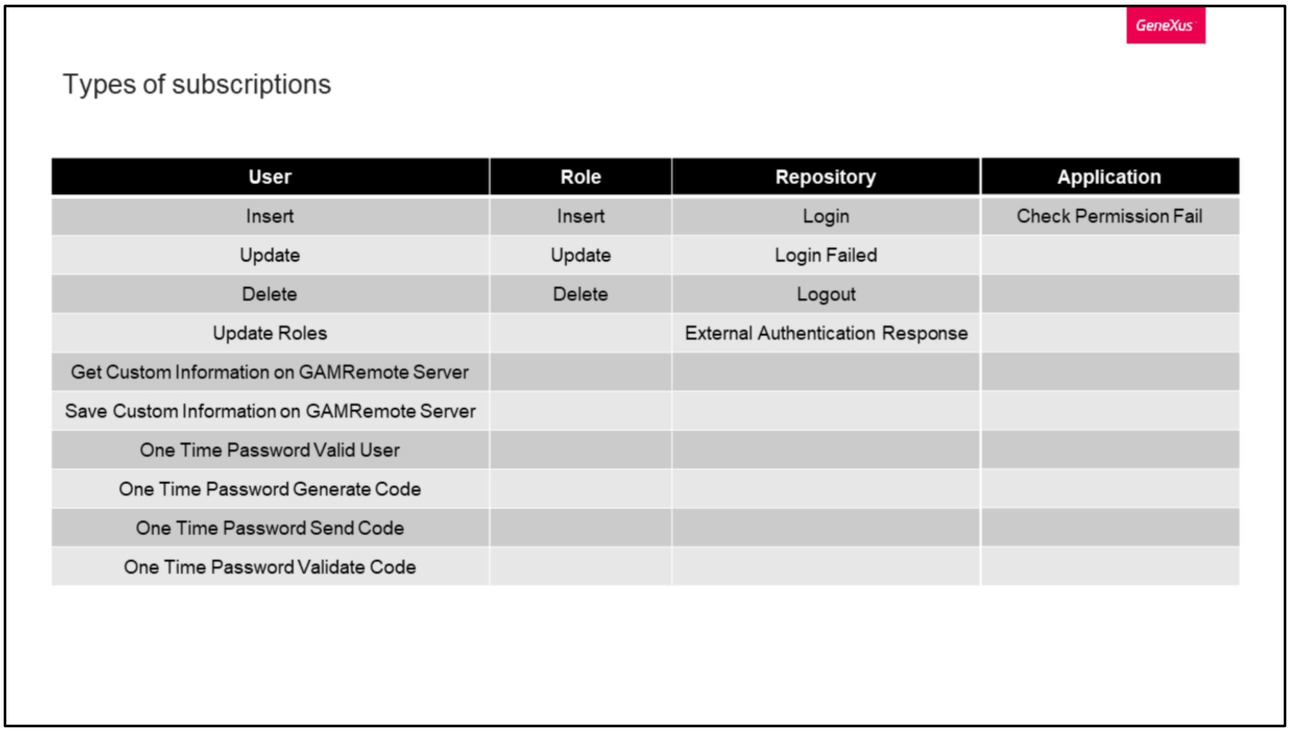
In this video, we will talk about the possibility of subscribing to events that occur in GAM, where it is possible to run code managed by a developer at the time these events occur.



## Purpose

GAM allows us to subscribe to different events that occur in applications, either when caused by a user action or triggered by some other action.

The purpose of this is to run additional code implemented by the GeneXus developer in predefined events provided by GAM.



## Types of subscriptions

| User | Role | Repository | Application |
|---|---|---|---|
| Insert | Insert | Login | Check Permission Fail |
| Update | Update | Login Failed | |
| Delete | Delete | Logout | |
| Update Roles | | External Authentication Response | |
| Get Custom Information on GAMRemote Server | | | |
| Save Custom Information on GAMRemote Server | | | |
| One Time Password Valid User | | | |
| One Time Password Generate Code | | | |
| One Time Password Send Code | | | |
| One Time Password Validate Code | | | |

There are four categories of events that we can subscribe to.
The first one is by User, which includes:

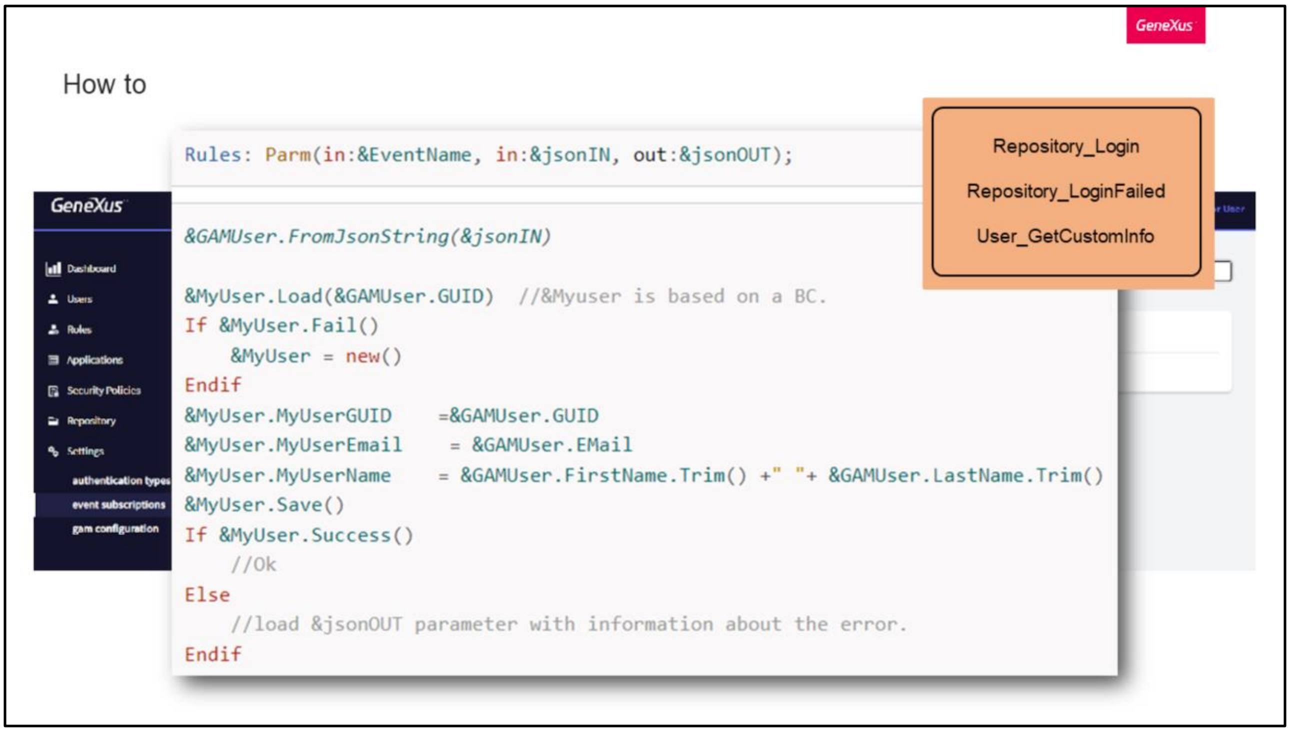
- Insert: Triggered by the insertion of a GAM User.
- Update: Triggered by the update of a GAM User.
- Delete: Triggered by the deletion of a GAM User.
- UpdateRoles: Occurs when the roles of a GAM user are changed.
- GetCustomInfo: Occurs in an IDP Server, and allows running code to obtain customized information from the user to send it to the GAMRemote client.
- SaveCustomInfo: Allows reading the customized information sent from the IDP server and running code to process that information in the Client and, for example, to save it in the tables that the system needs.
- OneTimePasswordValidUser: Allows including code for the developer to validate the user who requested an OTP code.
- OneTimePasswordGenerateCode: An event where the developer can customize how the OTP code to be sent to the user is generated.
- OneTimePasswordSendCode: An event that allows customizing how the OTP code is sent, which can be by SMS, notification, email, etc. By default, GAM sends it by email.
- OneTimePasswordValidateCode: A developer event used to validate the OTP code.

Then there are Role events:
Here we simply have the typical Insert, Update and Delete.

There are the events of a Repository:

- Login: Occurs when a GAM user login takes place, regardless of the authentication type.
- LoginFailed: Occurs when the user login fails, only due to an incorrect username and/or password.
- Logout: Triggered on GAM logout.
- External Authentication Response: An event to customize how to process the response from an external IDP. This event must interact with the external IDP and end with the local login.

Lastly, we have Application events; in this case, there is only Check Permission Fail, which is triggered when a permission is denied. This could be used, for example, to record a log of permissions that are checked and denied.



Repository_Login

Repository_LoginFailed

User_GetCustomInfo

```
Rules: Parm(in:&EventName, in:&jsonIN, out:&jsonOUT);

&GAMUser.FromJsonString(&jsonIN)

&MyUser.Load(&GAMUser.GUID)  //&Myuser is based on a BC.
If &MyUser.Fail()
    &MyUser = new()
Endif
&MyUser.MyUserGUID    =&GAMUser.GUID
&MyUser.MyUserEmail    = &GAMUser.EMail
&MyUser.MyUserName    = &GAMUser.FirstName.Trim() +" "+ &GAMUser.LastName.Trim()
&MyUser.Save()
If &MyUser.Success()
    //Ok
Else
    //load &jsonOUT parameter with information about the error.
Endif
```

Let's see how to subscribe to an event through the GAM web back office.

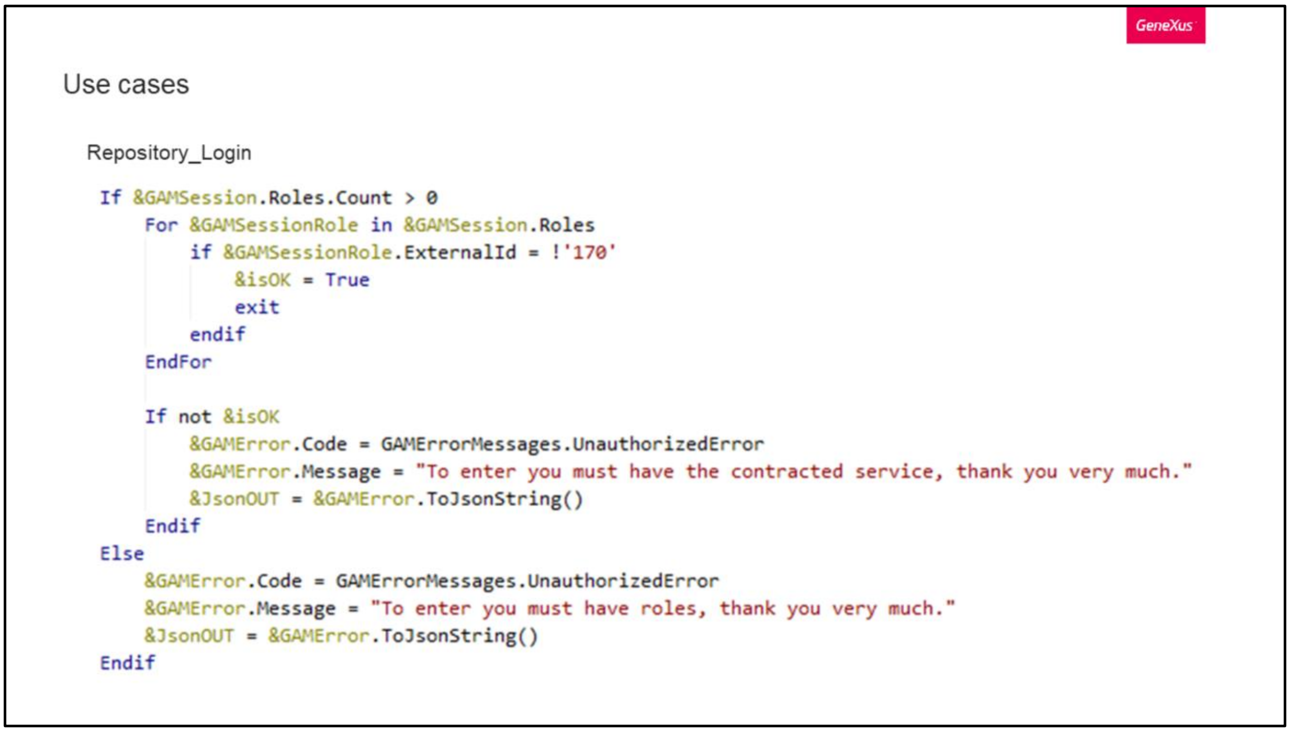
To do so, we go to Settings/Event Subscription, and click on Add.

There we can enter a description, and select which event we want to subscribe to, as well as the file name (this is the name of the .dll or .class file that will listen to the event execution), the class name (this is the name of the program including its package in the case of Java), and finally the method's name (in GeneXus, the method of the program is always “execute”).

One possible implementation of a procedure that notifies about the notification event to the user could be the following, which if the Success method fails, loads the output JSON with the error information.

Pay special attention to the rules that must be in the procedures that we make, since they must receive the name of the event and input JSON that will have information about the event.
The output JSON is only used by certain events:

- Repository_Login and Repository_LoginFailed, where it must return empty if OK, and the GAMError object if there is an error.
- User_GetCustomInfo where the output parameter must have the JSON to send to the client.



## Use cases

Repository_Login

```
If &GAMSession.Roles.Count > 0
    For &GAMSessionRole in &GAMSession.Roles
        if &GAMSessionRole.ExternalId = !'170'
            &isOK = True
            exit
        endif
    EndFor

    If not &isOK
        &GAMError.Code = GAMErrorMessages.UnauthorizedError
        &GAMError.Message = "To enter you must have the contracted service, thank you very much."
        &JsonOUT = &GAMError.ToJsonString()
    Endif
Else
    &GAMError.Code = GAMErrorMessages.UnauthorizedError
    &GAMError.Message = "To enter you must have roles, thank you very much."
    &JsonOUT = &GAMError.ToJsonString()
Endif
```

Let's see some event subscription use cases.

Repository_Login.

As we said before, this event occurs when a GAM user login occurs, regardless of the type of authentication, which allows canceling the login and displaying a message to the end user.
For example, if the user must have a certain role to log in to an application, it could be validated and an error could be returned.

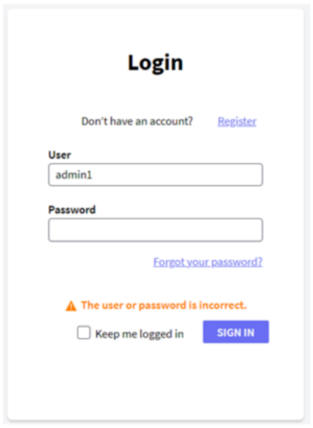
A possible implementation for this could be the following.

We check the session roles by comparing to role 170, where that role is the one that the user must have, and if the user has it, we do nothing.

On the other hand, if they don't have it, we set the GAMError with the information we want; we will have to stop the login. To do this, a JSON must be returned from the GAMError object.
We also do this when there are no roles, since it would be the same case but with a different error message.

## Use cases

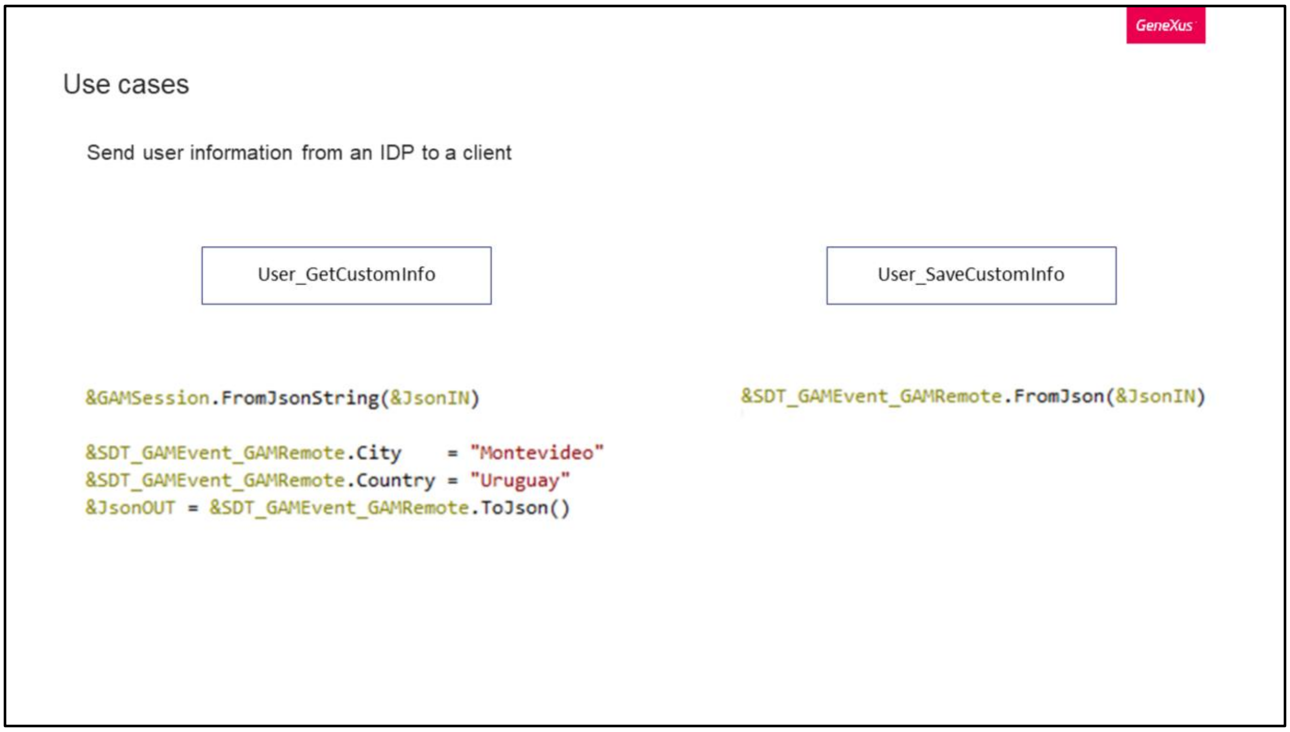
Repository_LoginFailed



```
&GAMSession.FromJsonString(&JsonIN)
Log.Error("User " + &GAMSession.User + " login failed.", &GAMSession.ApplicationId)
```

Let's continue with the Repository events, but this time with LoginFailed.
As we said before, this event only occurs when a user's login fails with GAM error 11 or 18, which stand for an incorrect password or username.

Our goal is to log all failed user attempts.
To do so, we must subscribe to the LoginFailed event, and in the procedure work as we wish with the JSON we will receive.
In this type of event, the JSON will correspond to the GAMSession External Object, so we can include all its properties in the error message that we will log.



## Use cases

Send user information from an IDP to a client

User_GetCustomInfo

User_SaveCustomInfo

```
&GAMSession.FromJsonString(&JsonIN)

&SDT_GAMEvent_GAMRemote.City    = "Montevideo"
&SDT_GAMEvent_GAMRemote.Country = "Uruguay"
&JsonOUT = &SDT_GAMEvent_GAMRemote.ToJson()
```

```
&SDT_GAMEvent_GAMRemote.FromJson(&JsonIN)
```

Another important use case could be how to send any type of data to a client from an IDP made with GAM. For this, we will use User events.

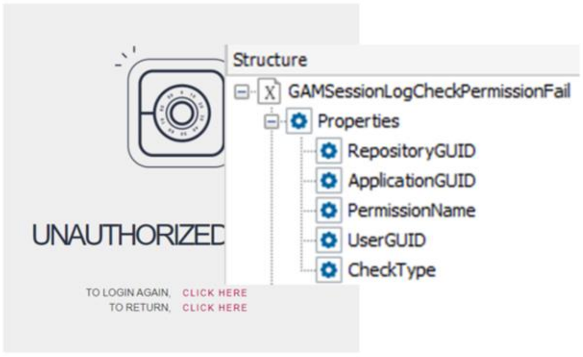
In the IDP we must have the GetCustomInfo event of the User section activated. Here it is advisable to load the data that we want to send, and send it as a JSON.

Then, from the client side, we will have to subscribe to the SaveCustomInfo event, also of User, and there receive the JSON sent to be able to query the information that we wanted to send and receive.
Of course, the information received must be processed by the developer and saved in the tables that required it.

That’s all.

## Use cases

Application_CheckPermissionFail



```
&GAMLog.FromJsonString(&JsonIN)
Log.Error("User " + &GAMLog.UserGUID + " without permission " + &GAMLog.PermissionName, &GAMLog.ApplicationGUID)
```

Now let's see a case associated with applications.
Our objective is the same as in the LoginFailed case we saw: to log all the access attempts to objects that the user doesn't have permission to access.
To do so, we must subscribe to the CheckPermissionFail event, and in the procedure work as we wish with the JSON that we will receive.
In this type of event, the JSON will correspond to the External Object GAMSessionLogCheckPermissionFail, so we can include the following properties in the error message that we will log.

GeneXus™ by Globant

training.genexus.com
wiki.genexus.com