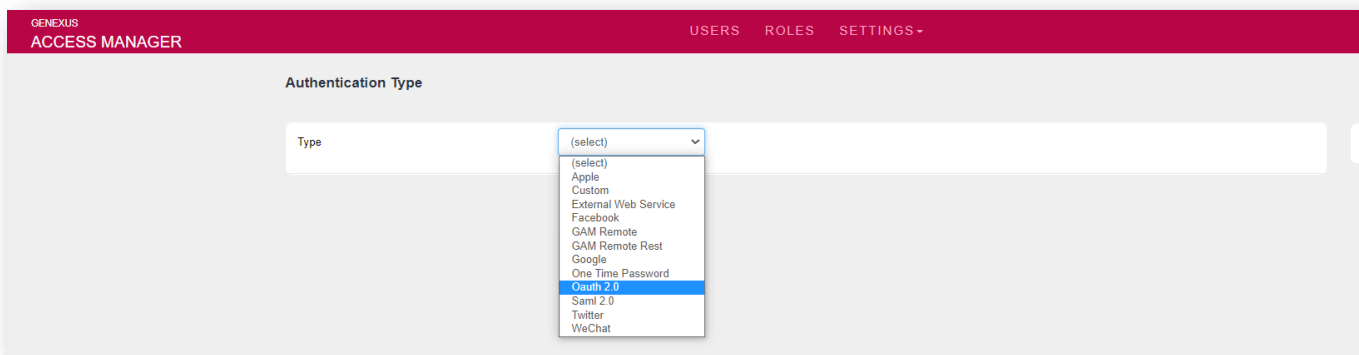
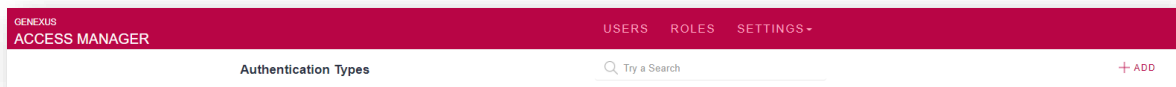


# DEMO: OpenID Connect

First demo: OpenID Connect



For this demo, we will use the Oauth 2.0 protocol in GAM. Our identity provider will be Azure Active Directory from Microsoft.

We will assume that configuration on the Azure side has already been made correctly and we will not go into detail about it. In the GeneXus Wiki, you will find a detailed article on how to do so.

First, we create a new GAM Oauth 2.0 Authentication Type and define the basic concepts, such as Name, Description, etc.

General	Authorization	Token	User Information	
Client Id	Tag	<input type="text" value="client_id"/>	Value	<input type="text" value="2d55e4aa-22c6-476e-acc6-8f3728018bc9"/>
Client Secret	Tag	<input type="text" value="client_secret"/>	Value	<input type="text" value=""/>
Redirect URL	Tag	<input type="text" value="redirect_uri"/>	Value	<input type="text" value="http://localhost:8080/GAMCourse.JavaEnvironment/"/>
Custom Redirect URL?		<input type="checkbox"/>		
Redirect to authenticate?		<input type="checkbox"/>		

In the General tab, the following must be defined:  
First, we set the Client ID and Client Secret obtained from Azure.  
The redirection URL must be the Base URL of our application's back end.

As we said in the previous video, we will not select the Redirect option to authenticate because we want to log in from the GAM itself.

General	Authorization	Token	User Information		
			https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize		
URL	https://login.microsoftonline.com/.../oauth2/v2.0/authorize				
<a href="#">Advanced configuration</a>					
ResponseType	<input checked="" type="checkbox"/>	Tag	response_type	Value	code
Scope	<input checked="" type="checkbox"/>	Tag	scope	Value	https://graph.microsoft.com/user.read
State	<input checked="" type="checkbox"/>	Tag	state		
Include Client Id	<input checked="" type="checkbox"/>				
<ul style="list-style-type: none"> <li>■</li> <li>■</li> <li>■</li> </ul>					
<b>Response</b>					
Access Code Tag	<input type="text" value="code"/>				
Error Description Tag	<input type="text" value="error_description"/>				

Now, we go to the Authorization tab.

There, we set the Azure URL obtained from its portal, which looks as follows.

Next, we modify the Scope, which must contain the URL shown on the screen.

The rest is left with the default values.

General Authorization Token **User Information** <https://login.microsoftonline.com/{tenat}/oauth2/v2.0/token>

URL

Advanced configuration

Token Method

Header

Tag	Value
Content-type	application/x-www-form-urlencoded

Include Authentication header?

Method  Realm

Include Authorization header with Basic value?

Grant Type

Tag	Value
grant_type	password

⋮

Additional Parameters

⋮

In the Token tab, once again we set the Azure URL obtained from the portal, which looks as follows.

The rest is left with the default values, except for the Grant Type and Additional Parameters fields, which should be set as shown on the screen.

Note that the latter should only be changed when you do not want to redirect when logging in, so that it is done from the GAM login. Otherwise, the Grant Type should be left with the default value (which is `authorization_code`) and without additional parameters.

The rest of the options are left with the default values.

General	Authorization	Token	User In
URL			
User Info Method			
Header			
Include Access Token			
Include Client Id			
Include Client Secret			
Include User Id			
User Email Tag			mail
User Verified Email Tag			
User External Id Tag			id
User Name Tag			userPrincipalName
User First Name	Tag		givenName
User Last Name	Tag		surname
User Gender	Tag		gender
User Birthday Tag			birthday
User URL Image Tag			picture
User URL Profile Tag			link
User Language Tag			locale
User Time Zone Tag			timezone
Error Description Tag			message

[Advanced configuration](#)

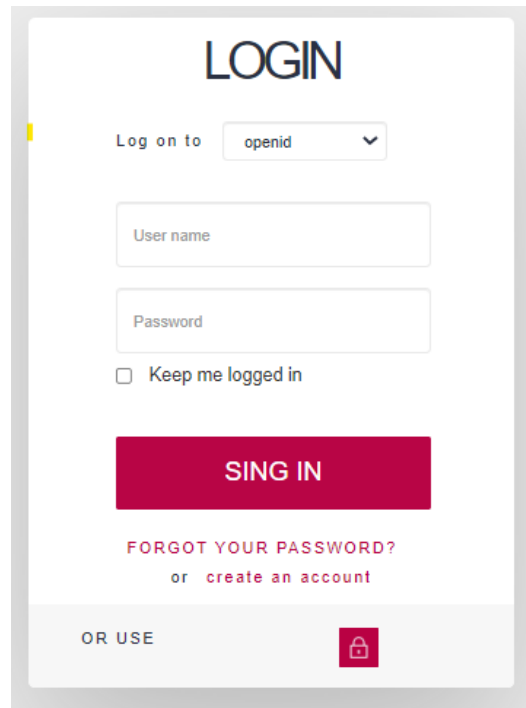
charset=utf-8

Lastly, in the User Information tab, we simply set the URL shown on the screen (also obtained from Azure) and do not include anything.


This is how the user is created in the local GAM, and it is where the user information is mapped according to the configured IDP. The IDP must return a unique user identifier, which must be configured in "User External Id Tag" to ensure that in subsequent GAM logins the same user is being authenticated.

We set the rest of the parameters as shown on the screen.

The configuration is now complete.



The image shows a login form with the following elements:

- LOGIN**: The main title of the form.
- Log on to**: A label for a dropdown menu.
- openid**: The selected option in the dropdown menu.
- User name**: A text input field.
- Password**: A text input field.
- Keep me logged in**: A checkbox with a label.
- SING IN**: A large red button.
- FORGOT YOUR PASSWORD?**: A link in red text.
- or create an account**: A link in red text.
- OR USE**: A label for a social login option.
- : A red square icon with a white padlock symbol.

Now, we go to Login, select the authentication type we have just created, and enter the credentials of a user defined in Azure.

That's all.

DEMO: IDP

Second demo: IDP.



Application

General Rem

Client ID

Client secret

Oauth single user

WEB (Identity Provider, SSO)

Allow authentication?

Can get user roles?

Can get user additional data?

Can get session initial properties?

Image URL

Local login URL

Callback URLs

For this demo, we will use the Oauth 2.0 protocol again. The GAM, through it, will be our identity provider.

First, we configure our GAM application defined on the IDP server that will act as the provider.

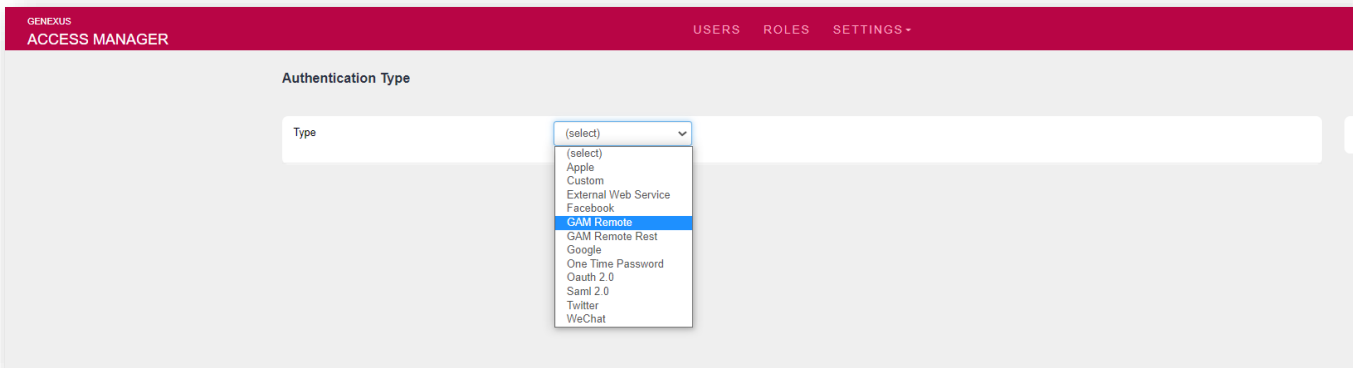
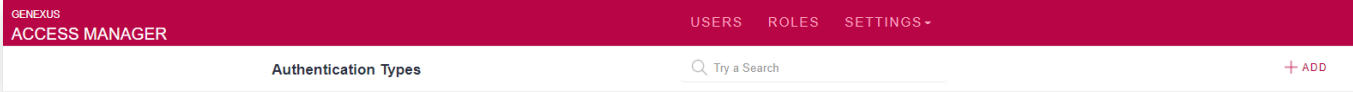
To do so, we select the “Remote Authentication” tab in the application settings from the GAM back end.

We save the Client ID and Client Secret to set them later in the client application. Next, we select the option to allow authentication in the WEB section (Identity Provider, SSO). There, you can indicate if you want to share with the Client the users' roles, additional information, etc.

It is important to show the local login and callback URLs in the demo. The first one must correspond to the Web Panel GAMRemoteLogin of the KB, which will perform the login process in the IDP. The second one must be the path of the client application from where the IDP will be invoked, which will be called after the login process is completed. This last parameter can be composed of more than one URL, which must be separated by semicolons.

Of course, GAM is where the users that will be used to log in to the IDP must be defined.


With this configuration and a user created, we have finished the process from the IDP side.



Let's look at the Client side.

The first step is to go to Authentication Types from the GAM back-end menu, and create a GAM Remote type.

## Authentication Type

Type	GAM Remote
Name*	gamremote
Function	Only Authentication ▾
Enabled?	<input checked="" type="checkbox"/>
Description	gamremote
Small image name	
Big image name	
Impersonate	(none) ▾
Client Id.*	b62c8a436ca34614821066e7d1c94ff8
Client Secret*	

It is important to configure the following:

Set the Function property to Only Authentication since on the IDP server side we do not indicate that the user roles are shared. If the other option (Authentication and roles) is set, we will get an error when logging in.

The next thing to configure is the Client Id and Client Secret saved from the IDP.

Local site URL*	<input type="text" value="http://localhost:8080/App1.JavaEnvironment"/>		
Custom callback URL?	<input type="checkbox"/>		
Add gam_user_additional_data scope?	<input type="checkbox"/>	Add gam_session_initial_prop scope?	<input type="checkbox"/>
Additional Scope	<input type="text"/>		
Remote server URL*	<input type="text" value="http://localhost:8080/AppIDP.JavaEnvironment"/>		
Private encryption key	<input type="text"/>		
Repository GUID	<input type="text"/>		
Validate external token	<input type="checkbox"/>		

\$Server/<Base\_URL>

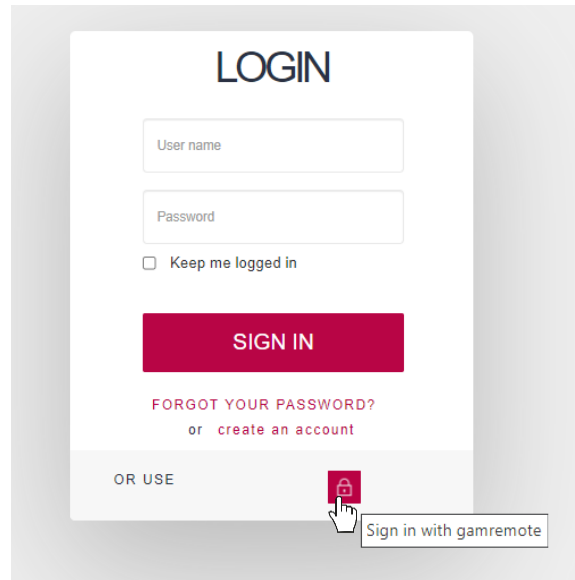
Later on, we will configure the “Local site URL” property with the address of our client application, the same that we already specified in the Callback URL in the server; also, the “Remote server URL” property with the address of the IDP, following the format shown in red.

Additional comments:

The property “Add gam\_user\_additional\_data scope?” must be activated when we want to send additional user data. On the server side, the Allow authentication property must be selected, in the Web section (Identity provider, SSO).

The “Add gam\_session\_initial\_prop scope?” property involves asking the IDP to return the initial properties dynamically set at login to the client. Of course, the IDP must also be configured to send this information.

Finally, the “Validate External Token” property validates the expiration of the session according to the expiration of the token and renews it automatically without having to do it manually.



LOGIN

User name

Password

Keep me logged in

**SIGN IN**

FORGOT YOUR PASSWORD?  
or create an account

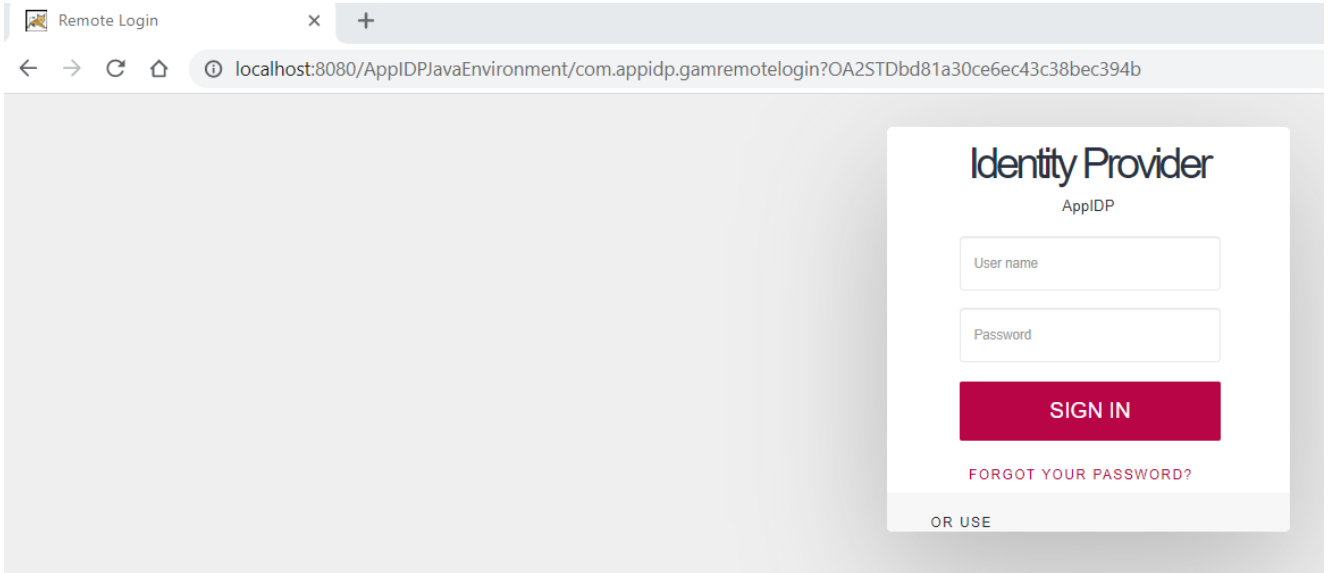
OR USE

Sign in with gamremote

For the purposes of the demo, we create a Web Panel in the Client application, where it shows us the data of the logged in user. Of course, this object has integrated security activated with the Authentication value.

When we want to access it, since we are not logged in, we are redirected to the login.

Note that since we have the Oauth authentication type defined, from the login we have the option to access through it.



When clicking on this option, we see that it redirects us to the IDP and its remote login.

We log in with the user that we had defined in the IDP.

## App Client

User

GENEXUS  
ACCESS MANAGER

USERS ROLES SETTINGS -

A Administrator

✓ SHOW FILTERS Users  + ADD

User Name	First Name	Last Name	Authentication	
<a href="#">nadrien@mail.com</a>	Nicolas	Adrién	idp	<a href="#">EDIT</a>
<a href="#">admin</a>	Administrator	User	local	<a href="#">EDIT</a>

FIRST / PREV / NEXT

Username	<a href="#">nadrien@mail.com</a>
E-Mail	<a href="#">nadrien@mail.com</a>
First Name	Nicolas
Last Name	Adrién

We are now redirected to our Web Panel with the information of the logged in user.

In the back end of the client application, we can see the user we had created in the IDP with its information.

## DEMO: Custom Authentication

Third demo: Custom Authentication.



```

Parm(in:&StrInput, out:&StrOutput); //&StrInput and &StrOutput are varchar(256)

&Key = '03E1E1AAA5BCA19FBA8C42058B4ABF28'
&GAMWSLoginIn.FromJson(&StrInput) // &GAMWSLoginIn is &GAMWSLoginInSDT data type

//Decrypt parameters
&UserLogin = Decrypt64( &GAMWSLoginIn.GAMUsrLogin, &Key )
&UserPassword = Decrypt64( &GAMWSLoginIn.GAMUsrPwd, &Key )
&GAMWSLoginOut = New GAMWSLoginOutSDT() //&GAMWSLoginOut is &GAMWSLoginOutSDT data type
&GAMWSLoginOut.WSVersion = GAMAutExtWebServiceVersions.GAM10
&GAMWSLoginOut.User = New GAMWSLoginOutUserSDT()

Do 'ValidUser'

&StrOutput = &GAMWSLoginOut.ToJson()

Sub 'ValidUser'
  If &UserLogin = !"user"
    If &UserPassword = !"password"
      &GAMWSLoginOut.WSStatus = 1
      &GAMWSLoginOut.User.Code = !"code"
      &GAMWSLoginOut.User.FirstName = !"FirstName"
      &GAMWSLoginOut.User.LastName = !"LastName"
      &GAMWSLoginOut.User.Email = !"name2@domain.com"
      Do 'GetRoles' //optional
    Else
      &GAMWSLoginOut.WSStatus = 3
    EndIf
  Else
    &GAMWSLoginOut.WSStatus = 2
  EndIf
EndSub

Sub 'GetRoles'
  &GAMWSLoginOutUserRol = New()
  &GAMWSLoginOutUserRol.RoleCode = "role_1"
  &GAMWSLoginOut.User.Roles.Add(&GAMWSLoginOutUserRol)
  &GAMWSLoginOutUserRol = New()
  &GAMWSLoginOutUserRol.RoleCode = "role_2"
  &GAMWSLoginOut.User.Roles.Add(&GAMWSLoginOutUserRol)
EndSub

```

To perform a Custom authentication, we must create a procedure.

In the GeneXus Wiki, you can find the example shown on the screen, with a very simple logic already defined. It is up to the developer to modify it as needed.

First, we see that two Varchar are defined as rules: an input and an output one, which will bring the data entered by the user and return the result of the login with certain user information (if successful, of course).

Then a key is defined that we will explain in detail later on, and the parameters of that input parameter from the rules are decrypted, in addition to creating a data type that will be loaded in the output parameter at the end of the process.

Next, in the **ValidUser** method the user name and password are validated; in the example, by verifying that the user name is "user" and the password is "password."

Otherwise, different errors are returned depending on the failure.

This method should be changed for a more secure login logic that does not distinguish between errors based on username or password.

Optionally, the **GetRoles** method can be used to define certain roles for the logged in user.

This method is useful when we want to program how we validate a user's password, either to validate it against a local database, against an LDAP, or against another place where the user's credentials are stored.



GENEXUS  
ACCESS MANAGER

USERS ROLES

SECURITY POLICIES  
APPLICATIONS  
REPOSITORY CONFIGURATION  
REPOSITORY CONNECTIONS  
AUTHENTICATION TYPES  
CHANGE PASSWORD  
CHANGE WORKING REPOSITORY  
EVENT SUBSCRIPTIONS  
GAM CONFIGURATIONS

ADMINISTRATOR

HIDE FILTERS

Users

Try a Search

+ ADD

GENDER  
(All)

AUTHENTICATION TYPE  
(All)

ROLE  
(All)

User Name	First Name	Authentication	
custom	FirstName	custom	EDIT
admin	Administrator	local	EDIT
test	Test	local	EDIT


FIRST / PREV / NEXT

Now that we have a custom authentication procedure, we need to configure it in GAM.

The first step is to go to Authentication Types, and create a new one of Custom type.

GENEXUS ACCESS MANAGER      USERS   ROLES   SETTINGS -      A Administrator

### Authentication Type

Type	Custom	 Delete
Name*	custom	
Function	Authentication and Roles ▾	
Enabled?	<input checked="" type="checkbox"/>	
Description	Custom Authentication Type	
Small image name		
Big image name		
Impersonate	(none) ▾	
Enable Two Factor Authentication?	<input type="checkbox"/>	
JSON version	Version 1.0 ▾	
Private encryption key	03E1E1AA5BCA19FBAB8C42058B4ABF	<input type="button" value="Generate Key Custom"/>
File name*	agamlogincustom.class	
Package	com.gamcourse	
Class name*	agamlogincustom	

Here, the settings to highlight are as follows:

**Function:** It allows specifying if we want the authentication type to be Authentication and roles, or only Authentication. In our case we leave the first option.

**Private encryption key:** here you must configure the encryption key used in the procedure to decrypt the user and password received. As you may remember, in the slide of the GeneXus procedure that I showed before, a key was defined that we enter in this property. It is useful because the GeneXus encryption function uses it to encrypt the username and password when they are sent to the program.

**File name:** here we specify the name of the file corresponding to the external procedure. In the case of Java, it is optional.

**Package:** in the case of Java models, the same Java package name property value is specified here, and in the case of NET models the value of the application namespace property is specified here. This property is optional and depends on whether the procedure or program used has a package or not.

Lastly, **Class name**, a mandatory property that specifies the name of the procedure's class.

# LOGIN

Log on to

Custom Authentication Type ▼

User name

Password

Keep me logged in

**SING IN**

[FORGOT YOUR PASSWORD?](#)  
or [create an account](#)

Once everything is configured, we simply set the custom authentication type in our login, and the login is made.

It should be mentioned that in this case the authentication type is selected through the highlighted combo box because we indicated that it should not be redirected to the IDP. Otherwise, the authentication type is shown as an icon at the bottom of the login as we saw in the IDP Demo.

DEMO: OTP

OTP.

### Repository Configuration

General Users Session **E-Mail**

Server Host	<input type="text" value="smtp.mail.outlook.com"/>	Server Port	<input type="text" value="587"/>
Timeout (seconds)	<input type="text" value="20"/>	Secure	<input checked="" type="checkbox"/>
Sender email address	<input type="text" value="admin@outlook.com"/>	Sender name	<input type="text" value="Mail"/>
Server require authentication?	<input checked="" type="checkbox"/>		
User name	<input type="text" value="admin@outlook.com"/>	Password	<input type="password" value=""/>
Send email when user activate account?	<input type="checkbox"/>		
Send email when user change password?	<input type="checkbox"/>		
Send email when user change email/username?	<input type="checkbox"/>		
Send email for recovery password?	<input type="checkbox"/>		

A prerequisite to make OTP work is that the repository must have the email service configured to send the codes. This is configured in the “Repository Configuration” option of the GAM back end.

GENEXUS ACCESS MANAGER

USERS ROLES

SECURITY POLICIES  
APPLICATIONS  
REPOSITORY CONFIGURATION  
REPOSITORY CONNECTIONS  
AUTHENTICATION TYPES  
CHANGE PASSWORD  
CHANGE WORKING REPOSITORY  
EVENT SUBSCRIPTIONS  
GAM CONFIGURATIONS

ADMINISTRATOR

HIDE FILTERS

Users

Try a Search

+ ADD

User Name	First Name	Authentication	
custom	FirstName	custom	EDIT
admin	Administrator	local	EDIT
test	Test	local	EDIT

FIRST / PREV / NEXT

GENDER: (All)

AUTHENTICATION TYPE: (All)

ROLE: (All)

Now, to define this type of authentication, everything is done and configured again through the GAM back end.

As in the previous Demo, we go to Authentication Types and add a new type.



GENEXUS  
ACCESS MANAGER

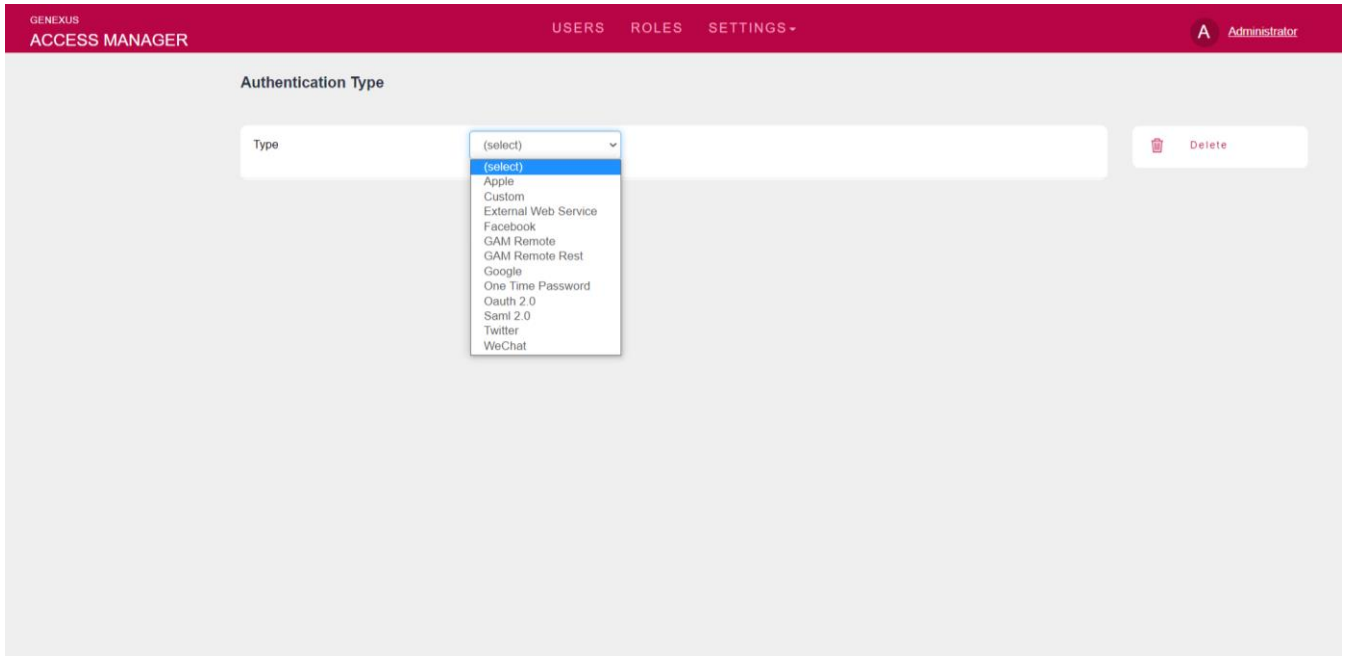
USERS ROLES SETTINGS -

A Administrator

Authentication Type

Type (select) Delete

- (select)
- Apple
- Custom
- External Web Service
- Facebook
- GAM Remote
- GAM Remote Rest
- Google
- One Time Password
- OAuth 2.0
- Saml 2.0
- Twitter
- WeChat



In this case, we select the One Time Password type.

## Authentication Type

Type	One Time Password
Name*	otp
Function	Only Authentication
Enabled?	<input checked="" type="checkbox"/>
Description	<input type="text" value="One Time Password"/>
Small image name	<input type="text"/>
Big image name	<input type="text"/>
Impersonate	<input type="text" value="local"/>
Use For First Factor Authentication?	<input checked="" type="checkbox"/>
User validation event	<input type="text" value="(none)"/>
Code generation type	<input type="text" value="OTP"/>

 Delete

Let's describe the most important properties:

**Impersonate:** Here we specify the type of authentication where users are going to be validated when using OTP. As I mentioned earlier in the theory section, the users must already exist. This is the only authentication type that requires configuring this property since the users must already exist in the GAM database.

**Use as first factor authentication:** If you do not configure this property, OTP could only be used as a second factor. In our case, we enable it.


Autogenerated OTP code length	<input type="text" value="6"/>
Generate code only with numbers?	<input checked="" type="checkbox"/>
Code expiration timeout (seconds)	<input type="text" value="1800"/>
Maximum daily number of codes	<input type="text" value="12"/>
Number of unsuccessful retries to lock the OTP	<input type="text" value="3"/>
Automatic OTP unlock time (minutes)	<input type="text" value="60"/>
Number of unsuccessful retries to block user based on number of OTP locks	<input type="text" value="3"/>
Send code using	<input type="text" value="Email by GAM"/>
Mail message subject	<input type="text" value="We have sent the code to access %1"/>
Mail message HTML text	<input type="text" value="To access the application %1 enter the following code: %2"/>
Validate code using	<input type="text" value="GAM"/>

The rest of the properties are used to define properties of the code to send and the format of the email.

In this case, we will use GAM's default format, which is email, but remember that it is possible to send the code via SMS. If the developer chooses this second way, he/she must implement and configure the GAM event that must be selected in the "Send code using" property.

# LOGIN

Log on to

One Time Password 

User name

Keep me logged in

**SEND ME A CODE**

We have sent the code to access GAMCourse



**Mail**   
para mí 

To access the application GAMCourse enter the following code: 784693

# LOGIN

user

Code

**VALIDATE CODE**

[BACK TO LOGIN](#)

Finally in the Login, we select the OTP type, where we can see that we will only be asked for the user name to send the code.  
After receiving the code via email, simply log in to authenticate in the system.

## DEMO: TOTP

In this demo, the steps to configure a new TOTP authentication type are almost the same as OTP, except for one difference.

GENEXUS  
ACCESS MANAGER

USERS ROLES

SECURITY POLICIES  
APPLICATIONS  
REPOSITORY CONFIGURATION  
REPOSITORY CONNECTIONS  
AUTHENTICATION TYPES  
CHANGE PASSWORD  
CHANGE WORKING REPOSITORY  
EVENT SUBSCRIPTIONS  
GAM CONFIGURATIONS

A Administrator

X HIDE FILTERS

Users

+ ADD

GENDER  
(All)

AUTHENTICATION TYPE  
(All)

ROLE  
(All)

User Name	First Name		Authentication	
custom	FirstName		custom	EDIT
admin	Administrator	User	local	EDIT
test	Test	GAM	local	EDIT

FIRST / PREV / NEXT

To define this type of authentication, we go again to Authentication Types and add a new type.

GENEXUS  
ACCESS MANAGER

USERS ROLES SETTINGS -

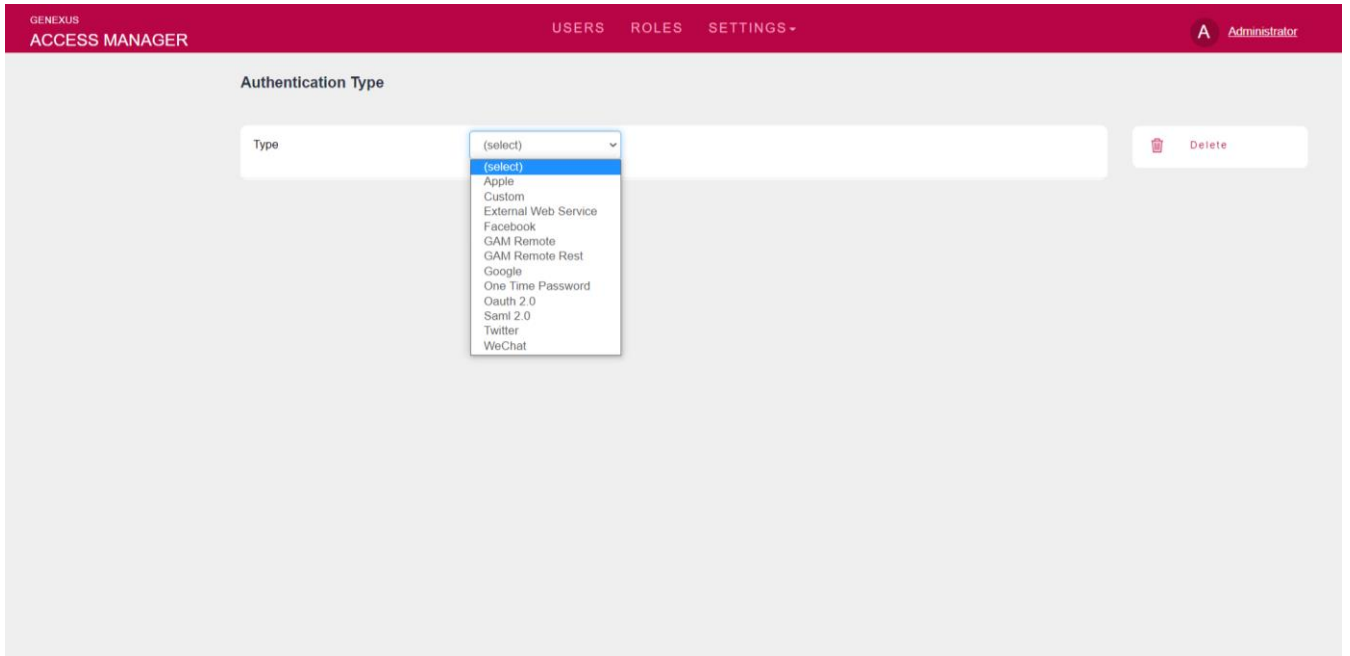
A Administrator

### Authentication Type

Type

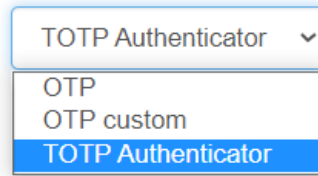
- (select)
- Apple
- Custom
- External Web Service
- Facebook
- GAM Remote
- GAM Remote Rest
- Google
- One Time Password
- OAuth 2.0
- Saml 2.0
- Twitter
- WeChat

Delete



In this case, we also select One Time Password type.

Code generation type



A dropdown menu with a light blue border and a white background. The menu is open, showing three options: 'TOTP Authenticator' (selected and highlighted in blue), 'OTP', and 'OTP custom'. A small downward arrow is visible on the right side of the top option.

TOTP Authenticator
OTP
OTP custom
TOTP Authenticator

The difference with OTP is the property shown on the screen, where in this case we choose TOTP Authenticator.  
The rest of the properties are for code configurations that are not relevant here.



## User

GUID	e7483297-a3e3-440e-a543-8b1801d09226
Name Space	GAMCourse
Authentication Type	GAM Local
User Name*	test
E-Mail*	user@mail.com


[Edit](#)[Permissions](#)[Roles](#)[Change Password](#)[Unblock OTP Codes](#)[Enable authenticator](#)[Delete User](#)

Let's see the most important caveat about OTP.

Each user must enable authentication through their settings. The most important difference is that this code algorithm is time-based and the codes are generated by the different authenticator applications.

For the purposes of the demo, it was created using the administrator user of the GAM back end for a "test" user of a sample application. The steps to be followed for this way consist of going to the user in question and clicking on Enable authenticator.

## Enable TOTP authenticator

User Name	test
Email	user@mail.com
	
Secret Key	EQ75LTFDWE62CVQK
Type a code	<input type="text"/>

[BACK](#)[ENABLE](#)

Once there, the QR code will be provided to be configured in a software system or mobile application based on one-time password authentication. After reading it, it will return the code to be entered in the "Type a code" field.

# LOGIN

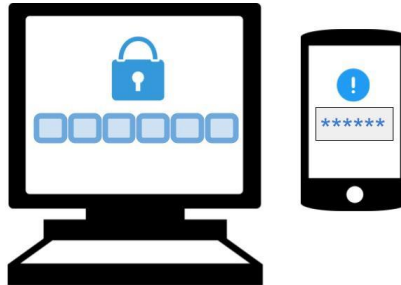
Log on to

Time-Based One Time Passw 

User name

Keep me logged in

NEXT



# LOGIN

user

Code

VALIDATE CODE

BACK TO LOGIN

Lastly, the login is the same as in all the previous types, and in this case there is an intermediary application that provides the access code.

# GeneXus™

[training.genexus.com](http://training.genexus.com)

[wiki.genexus.com](http://wiki.genexus.com)

[training.genexus.com/certifications](http://training.genexus.com/certifications)