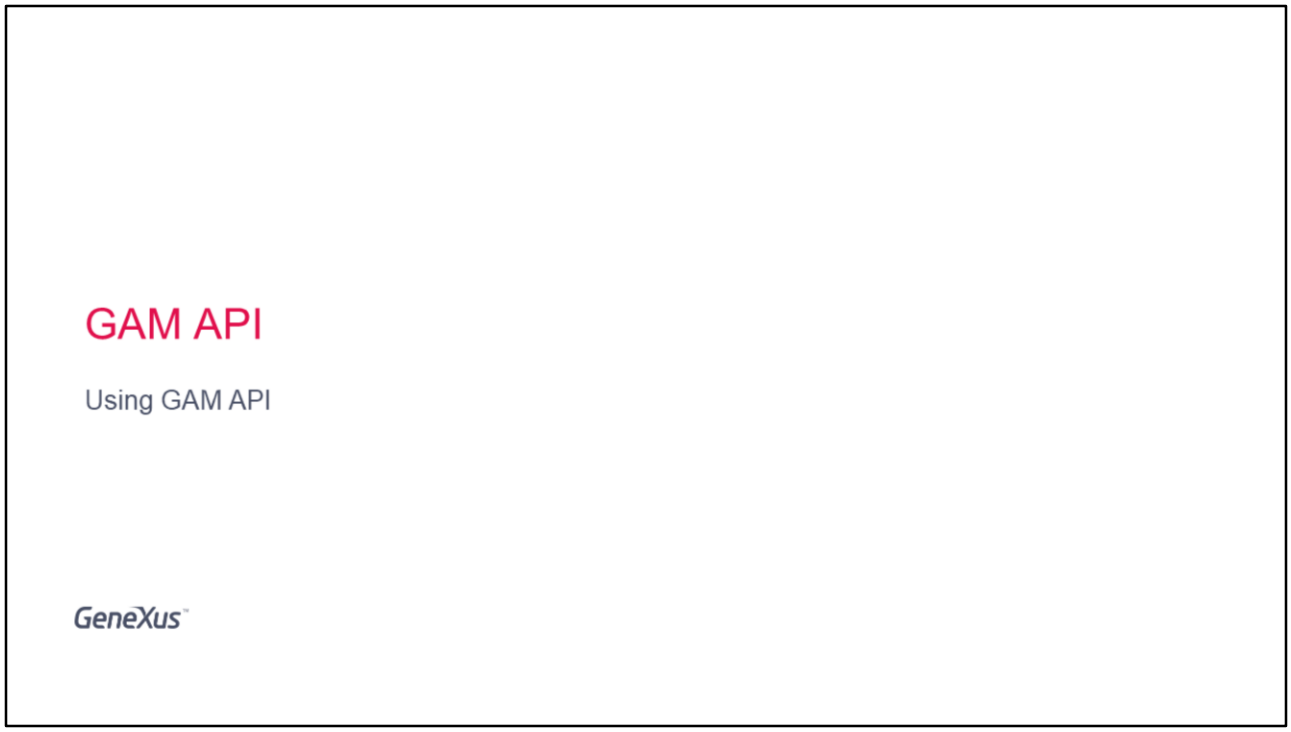

GeneXus™
by Globant

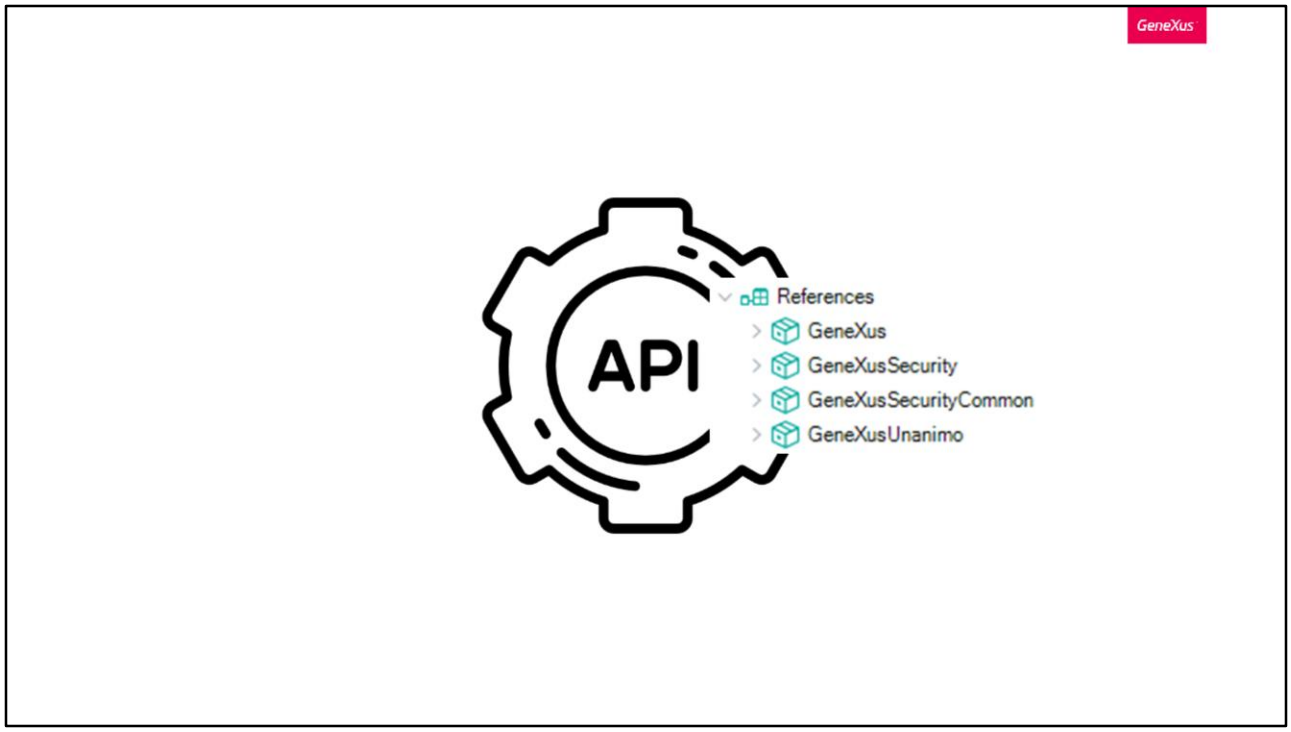
API

Nicolas Adrién

GeneXus



En este video explicaremos en que consiste la API de GAM, con sus métodos e implementaciones de referencia, tipos de autenticación, servicios REST, entre otras.



GAM proporciona una API que permite a los usuarios poder manejar distintos tipos de datos y métodos para agregar seguridad a las aplicaciones GeneXus, tanto aplicaciones Web como aplicaciones Móviles.

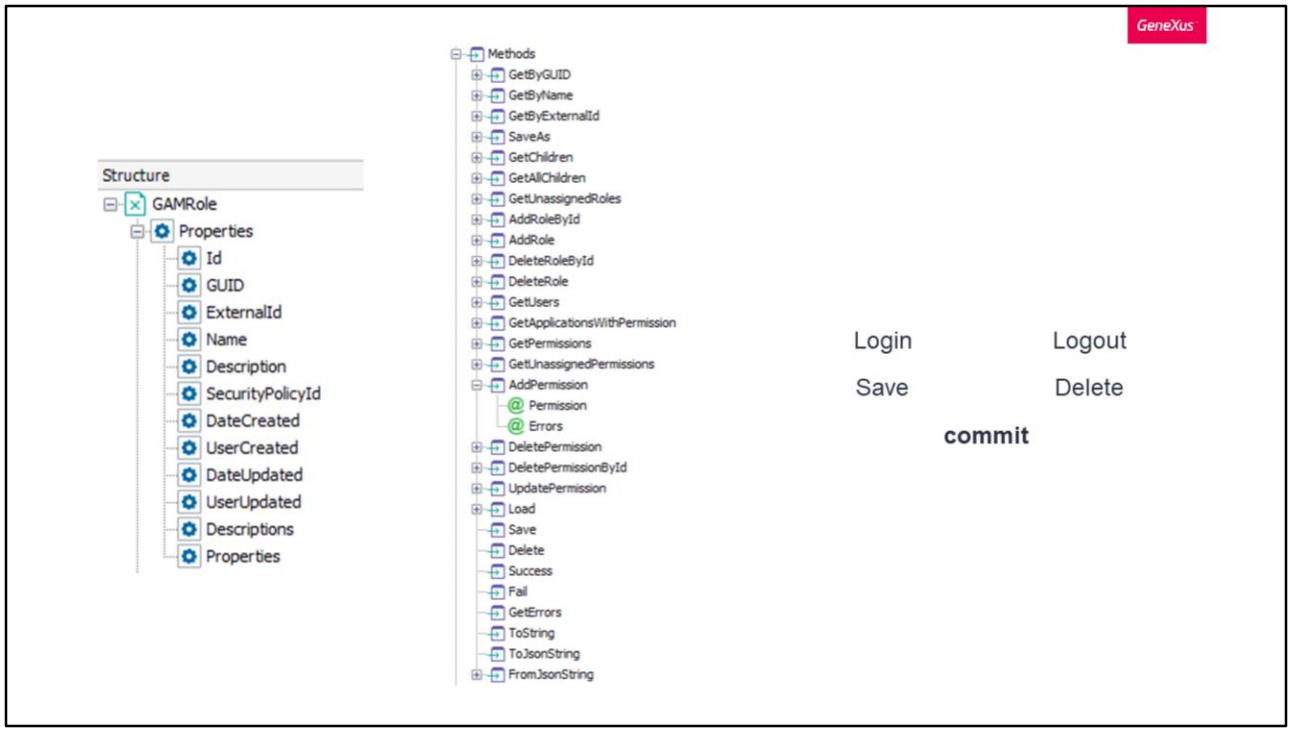
Cuando la seguridad integrada está habilitada en la KB, se incorporan determinados External Objects para permitir que el usuario interactúe con la API de GAM, donde éstos objetos forman parte de esta.
Todos ellos se pueden encontrar bajo el módulo llamado GeneXusSecurity. Sus dominios, están distribuidos en el módulo GeneXusSecurityCommon.

En el curso introductorio, vimos el listado de estos External Objects que brinda GAM, y ejemplificábamos el uso de uno. Ahora veamos en detalle esto.

Estos objetos tienen propiedades y métodos, y a su vez, algunos de ellos implementan los mismos métodos que los Business Component, como son Load, Save, Delete, Fail y Success.
Si se cambia alguna propiedad en esos objetos GAM, se debe llamar al método Save() y ejecutar el comando commit.

Los objetos que se comportan como Business Component son: GAMRepository, GAMApplication, GAMUser, GAMSecurityPolicy, GAMEventSubscription y todos los tipos de autenticación.

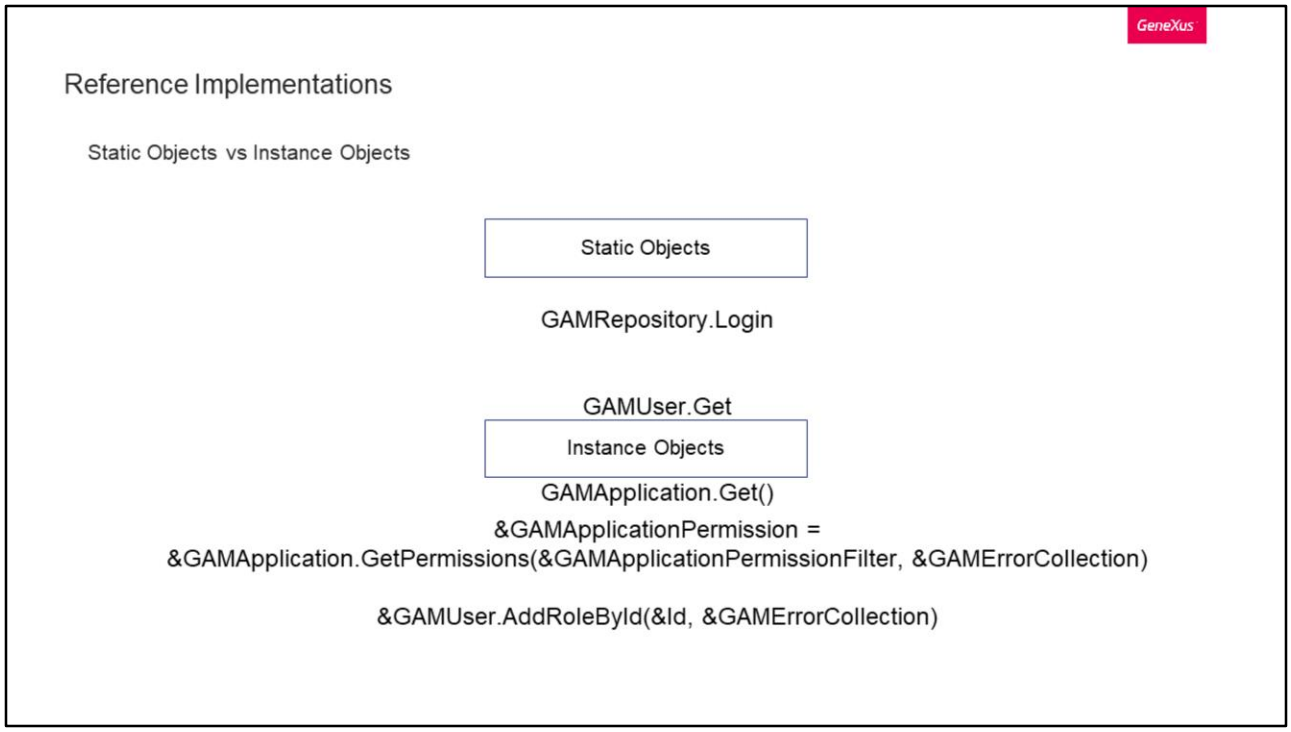
Como dijimos anteriormente, todos estos objetos se pueden encontrar en el módulo GeneXusSecurity.



Acá vemos como ejemplo el objeto GAMRole.
Como hemos mencionado, estos objetos se componen de propiedades y métodos.

A su vez, los objetos GAM también tienen otros métodos implementados para crear, actualizar o eliminar objetos.
Al utilizarlos, el comando Commit debe usarse después de que el método se haya ejecutado correctamente. Este comando, al igual que en los Business Components, además se utiliza después de un Save o Delete.

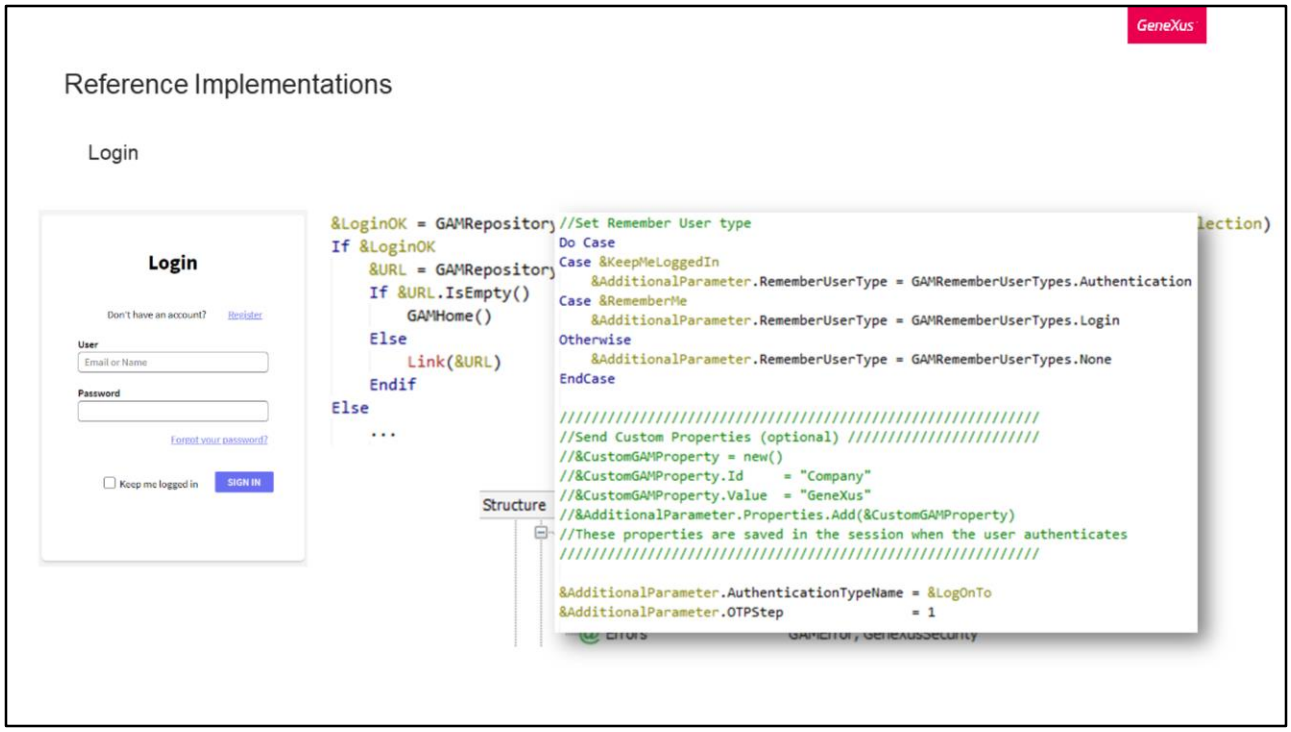
Los únicos métodos GAM que ejecutan un commit implícito son los relacionados con el inicio y cierre de sesión, y se ejecutan en una nueva unidad lógica de trabajo, donde no es necesario realizarlo ya que internamente el GAM produce los distintos inserts y hace el commit.



Dentro de lo que son las implementaciones de referencia, podemos encontrar usos de Objetos Externos estáticos y objetos de instancia.
La diferencia entre estos, es que los estáticos son métodos que aplican cuando en el objeto no se pone el &.

Algunos ejemplos de objetos estáticos, pueden ser los siguientes:
GAMRepository.Login ==> Con esto podemos hacer Login en el repositorio actual
GAMUser.Get ==> Con este podemos obtener la información del usuario actualmente logueado
GAMApplication.Get() ==> Con el cual podemos obtener la información de la aplicación actual

Ejemplos de objetos de instancia pueden ser:
&GAMApplication.GetPermissions ==> Donde al tener el & se le indica que el método se aplicará sobre la instancia del objeto cargado, que en este caso es GAMApplication. Con esto obtenemos los permisos de una aplicación.
&GAMUser.AddRoleById, donde recibe un identificador y una colección de errores, que con esto agregamos un Rol al usuario cargado previamente (que fue instanciado).



Veamos ahora dos implementaciones de referencia propias, como lo es Login y Logout.
Empecemos con el Login. Para esto, contamos con el objeto GAMExampleLogin.

Al momento de presionar el botón de iniciar sesión, la implementación se basa en ejecutar el método Login del External Objects: GAMRepository.
Como vemos, se está utilizando el método Login de este objeto ya provisto por GAM. Los parámetros incluidos en el llamado son:
Nombre de usuario y contraseña, obtenidos desde el WebPanel.
Parámetros adicionales: Estos pueden ser la opción de mantener la sesión iniciada, recordar usuario, u otro caso. A su vez, tenemos la posibilidad de crear propiedades Custom y asociarlas a la sesión del usuario como vemos en pantalla. También podemos enviarle el tipo de autenticación usado, información relacionada con OTP, etc.
Colección de errores: En caso de existir algún error, son cargados en dicha variable para después revisarlos.

El resultado de la operación se guarda en la variable booleana LoginOK, y dependiendo de dicho valor, es la acción que se realiza.
Si este es verdadero, se guarda en la variable URL el valor resultado del método GetLastErrorsURL. Este devuelve la URL donde ocurrió el último error de GAM y una vez que es ejecutado, se elimina su valor. Esto es útil para cuando un usuario intenta

acceder a un objeto con autenticación y no puede debido a que aún no está logueado, por lo tanto es redireccionado primero al login y posteriormente desde aquí se lo vuelve a llevar al objeto en cuestión, ahora sí con una sesión válida.
En caso contrario, simplemente se lo redirecciona al Home definido en la aplicación.

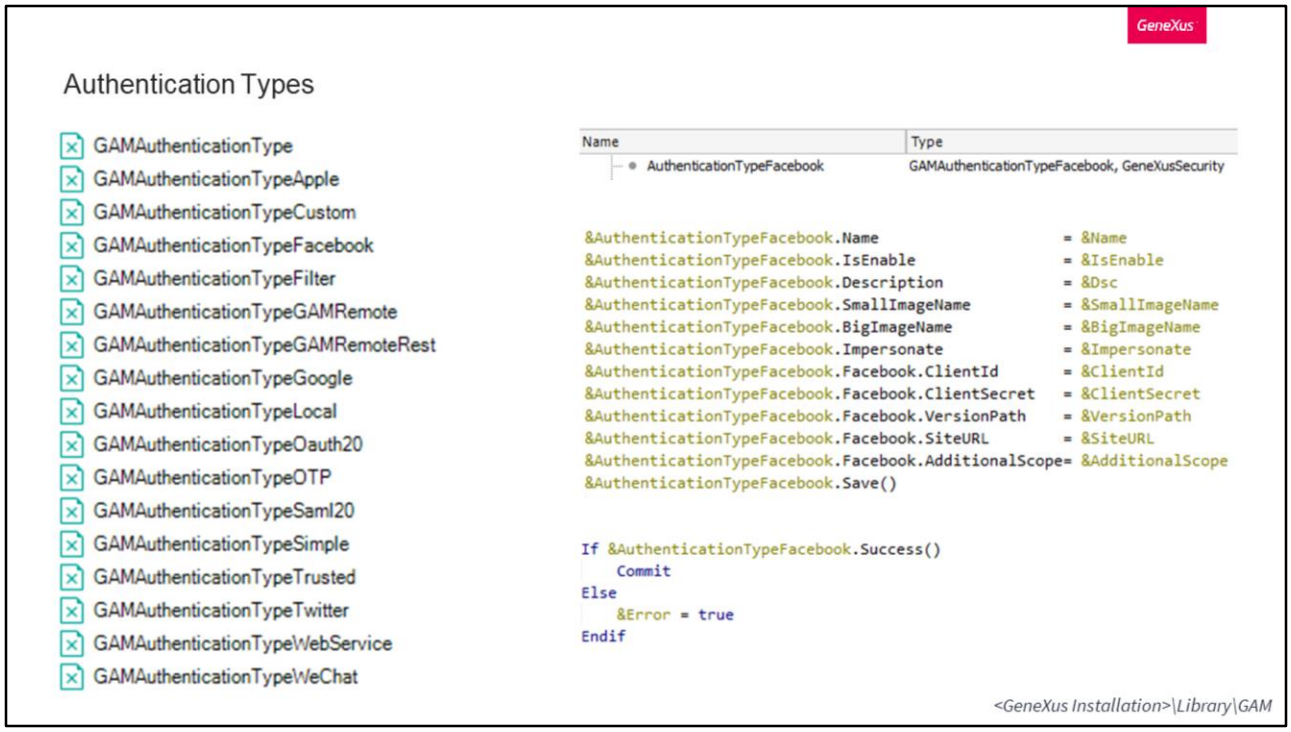
GeneXus

Reference Implementations

Logout

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

Logout.
Una implementación para esto es super sencilla, ya que podría realizarse de la siguiente manera.
Con la primera instrucción se finaliza la sesión actual y en caso de que existan errores se guardan en el SDT &Errors.
La segunda instrucción simplemente finaliza el flujo llevando al usuario hacia el Web Panel GAMExampleLogin, o en su defecto, al Web Panel encargado para el login.
Esta última instrucción sólo se debe realizar en los casos donde ésta aplicación NO sea cliente de un IDP. En el caso de que sí lo sea, no se tiene que llamar a nada luego del Logout para que se realice el cierre de sesión en el IDP.

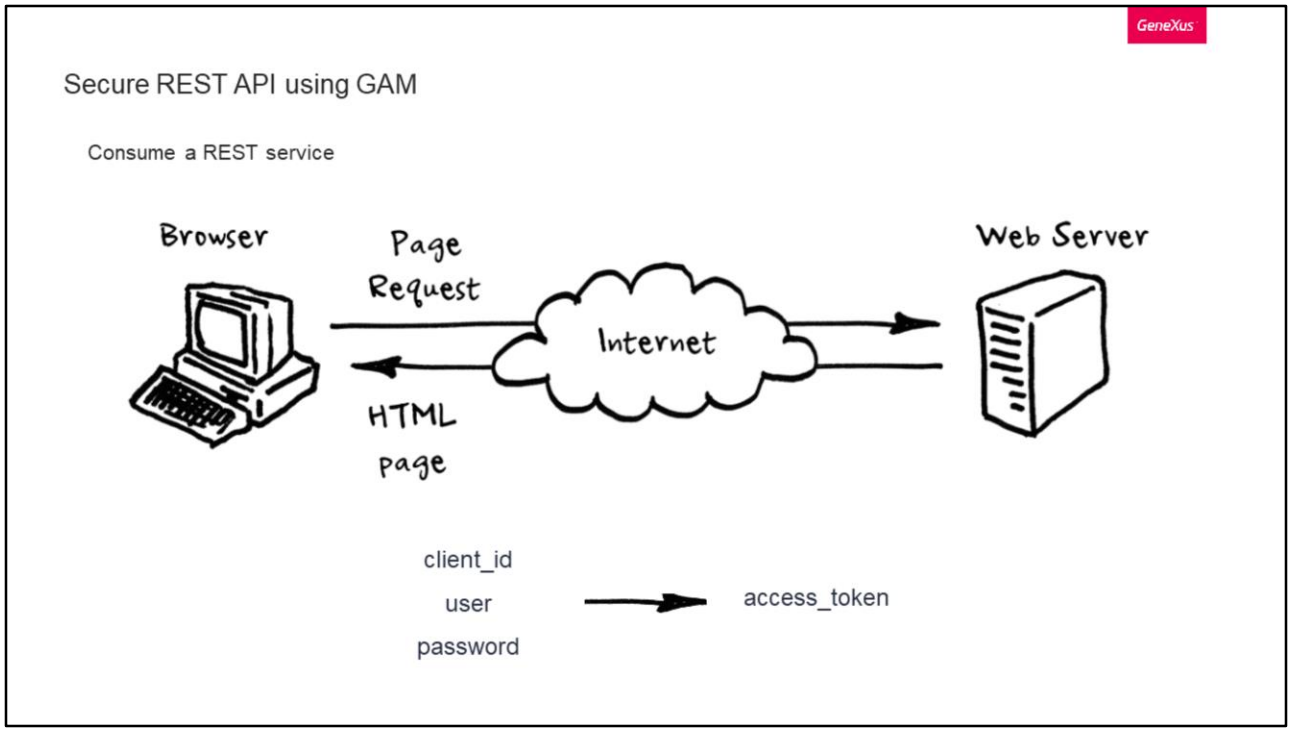


En el curso introductorio también cubrimos los tipos de autenticación desde el backoffice web de GAM, pero mencionamos que esto a su vez es posible a través de la API de GAM.
Para ello, tenemos los External Objects que vemos en pantalla, uno por cada tipo de autenticación soportado por GAM.

Pongamos el ejemplo del tipo de autenticación con Facebook.
Si quisiéramos crear uno de estos, lo primero que debemos hacer es crear una variable de este tipo.
Posteriormente, empezamos a completar los distintos atributos del tipo de autenticación, según nuestros datos.
Si el tipo de autenticación es satisfactorio, realizamos un commit para persistir los datos.

Eso es todo.

En los ejemplos de la API de GAM pueden encontrar este y todos los demás ejemplos para cada tipo de autenticación, tanto para la inserción, como para la modificación y eliminación.
Estos ejemplos se encuentran ubicados en la carpeta de instalación de GeneXus, dentro del directorio Library/GAM.



Servicios REST.

Es muy común que las aplicaciones expongan servicios REST, por lo que la seguridad es muy importante cuando la privacidad de los datos es una preocupación.
En GeneXus, como ya hemos dicho, la solución a esto es utilizar GAM, donde por detrás utiliza la tecnología OAuth facilitando al usuario la complejidad que este contiene.

La API de GAM proporciona una forma de restringir el acceso de los usuarios a los servicios REST definidos en la aplicación.

Para consumir un servicio REST en GeneXus, primero se debe contar con el client_Id de la Aplicación, y el usuario y contraseña con permisos de acceso a esta aplicación.
Antes de consumir el servicio web, primero debe obtener un access_token.

Veamos una posible implementación de un procedimiento que lo consuma.

GeneXus

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)
```

El primer string está conformado por lo siguiente:

- Client_id de la aplicación
- Grant_type del flujo
- Scope de la cuenta de usuario a la que desea acceder
- Nombre de usuario de la cuenta a la que desea acceder, y su contraseña

La URL a consumir será la base de la aplicación, seguido de /oauth/access_token.
Se requiere agregar el header Content-Type.
Y ejecutamos un pedido POST con dicha información, a través del httpclient.

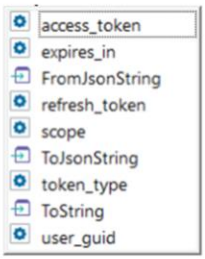
GeneXus

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"
}
```

Para leer la respuesta, podemos hacerlo tanto en un string como en un JSON. En el caso de este ultimo, tenemos los siguientes atributos para poder leer de la respuesta de forma programática.

En caso de que la información esté correcta, la respuesta del pedido debería lucir de la siguiente manera otorgándonos en un JSON los datos que vemos en pantalla.

Ahora supongamos que ya tenemos el access_token que vimos anteriormente, como procedemos para consumir realmente el servicio que queramos?

Para hacer esto, cada llamada debe contener el access_token incluido en el cabezal Authorization de nuestros pedidos, luego de la palabra OAuth como vemos en pantalla.

Un ejemplo de código puede ser el siguiente, donde se muestra como obtener un listado de productos.
Indicamos la url, agregamos los encabezados donde uno es el que dijimos antes con el access_token.
Y finalmente ejecutamos el pedido, que en este caso es un GET a DPProduct, que como dijimos, representa a un listado de productos expuesto como servicio web REST, donde este es un servicio seguro gracias a que GAM estaría habilitado en su KB.

Para cerrar la parte teórica de este tema, veremos los External Objects más comunes y utilizados en las aplicaciones GeneXus.
Comencemos con GAMRepository.

Este objeto es bastante extenso, tanto en sus propiedades como métodos. Las opciones más comunes que brinda es todo lo relacionado directamente a un repositorio, Login, usuarios, sesiones, roles, aplicaciones, etc.

Veamos algunos de los métodos más comunes.

En primer lugar tenemos a GetAuthenticationTypes. Este método sirve para obtener los tipos de autenticación de un repositorio.
Luego tenemos a GetUsersOrderBy el cual obtiene todos los usuarios de acuerdo a las preferencias que le pasemos como filtros, donde además podemos definir un orden para el listado de retorno (que se corresponden al dominio GAMUserListOrder).
GetRoles y GetApplications corresponden a métodos para obtener los roles y aplicaciones respectivamente de un repositorio.
Finalmente tenemos a GetSessionLogs con el cual podemos obtener un log de todas las sesiones de un repositorio, teniendo la posibilidad por ejemplo de filtrar por las sesiones activas.

Como vemos, en todos los métodos siempre se incluyen los mismos parámetros:

Los terminados en Filter son de utilidad para pasarle filtros a la consulta a realizar en la base de datos. Cada uno corresponde a otro External Object con la especificación del mismo, los cuales se pueden revisar antes de utilizarlos con el fin de definir un filtrado adecuado a nuestras necesidades.
El otro parámetro corresponde a los distintos errores que pueda causar el método al ser utilizados, los cuales luego podemos leer para verificar el resultado de la operación.

GAMApplication.

Con este objeto tendremos control sobre las aplicaciones definidas en GAM.

Veamos cuatro de sus métodos más comunes.
En primer lugar tenemos GetPermissions, el cual nos permite obtener el listado de permisos definidos en una aplicación. En su primer parámetro le indicaremos el filtrado que queremos realizar en el listado de resultado.
Los tres restantes, nos sirven para agregar, modificar o eliminar un permiso de una aplicación. En este método debemos indicarle a través del primer parámetro los datos del permiso al que aplicaremos una de esas funciones.

GAMUser.

Con este objeto tenemos el control total de un usuario de GAM. Entre sus métodos, se destaca el poder obtener toda la información de cada uno, y el poder controlar sus roles, permisos y sesiones,

Entre los más comunes tenemos el clásico GetRoles para obtener los roles de un usuario.
Luego tenemos GetUnassignedRoles que vendría a ser el método inverso al anterior. Este nos devuelve los roles que el usuario no tiene asignados.
GetPermissions, que nos devuelve los permisos asociados al usuario, y al igual que el método anterior, tenemos la variable de filtrado.
Y finalmente PhysicalDelete, que elimina físicamente al usuario y todos sus datos relacionados.

GAMRole.

Con este objeto podemos manejar los roles de GAM. Entre sus métodos, se destaca el poder obtenerlos, agregar y eliminar, controlar sus permisos, entre otros.

Los métodos más comunes son los siguientes:
Con GetPermissions obtendremos todos los permisos asociados al rol que tenemos en la variable GAMRole.
Con GetChildren obtendremos todos los roles hijos que contiene el rol identificado.
En ambos métodos podemos utilizar el filtrado que venimos comentado en todos los demás métodos.

Luego tenemos agregar y eliminar rol por identificador, que su función es básicamente lo que dice el nombre del método con la salvedad de que trabajan con los roles hijos del rol al cuál se instancia. En el parámetro Id le pasamos el identificador del rol a agregar o eliminar.

Sigamos con GAMSession.

Con este objeto tenemos el control de todas las sesiones. Podemos obtener todos los atributos de una sesión, sus roles, entre otras cosas.

Veamos dos de sus métodos:
Si queremos cargar los datos de la sesión actual, lo hacemos de la siguiente manera utilizando a GAMSession.Get() y almacenando el resultado en la variable &GAMSession. Dicha variable es exclusivamente para la sesión actual.
Si por ejemplo quisiéramos ver el histórico de todas las sesiones, debemos utilizar el objeto &GAMSessionLog.

De la misma forma que cargamos los datos de la sesión actual, podemos cargar los roles que tiene una sesión utilizando el método GetRoles.

GAMExamples

GAMVersion

Por último tenemos a GAMVersion, que si bien es un dominio y no un objeto, puede ser de utilidad conocerlo.

Para obtener la versión de GAM, tenemos dos formas de hacerlo:
La primera es a través del Backoffice web, como vemos en pantalla, en la opción GAM Configuration dentro de Settings.
La segunda forma es a través de código GeneXus, y para esto tenemos distintas opciones:

- Si queremos conocer la versión del API, podemos hacer lo siguiente:
  1. Primero podemos consultarla con el dominio, de la siguiente manera.
  2. Otra opción es realizando el siguiente For, donde en la variable GAMAPIVersion de tipo GAMVersion tendremos la versión buscada.

- Si queremos conocer la versión en la que está la estructura de la base de datos, podemos realizar una consulta a la base de datos como la que vemos en pantalla, esto nos devuelve un booleano indicando el resultado y en la variable GAMDatabaseVersion nos queda registrado el número de versión.

Un detalle con esto que tienen que tener en cuenta, es que la versión de la base de datos puede ser más nueva que la versión del API de GAM.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com