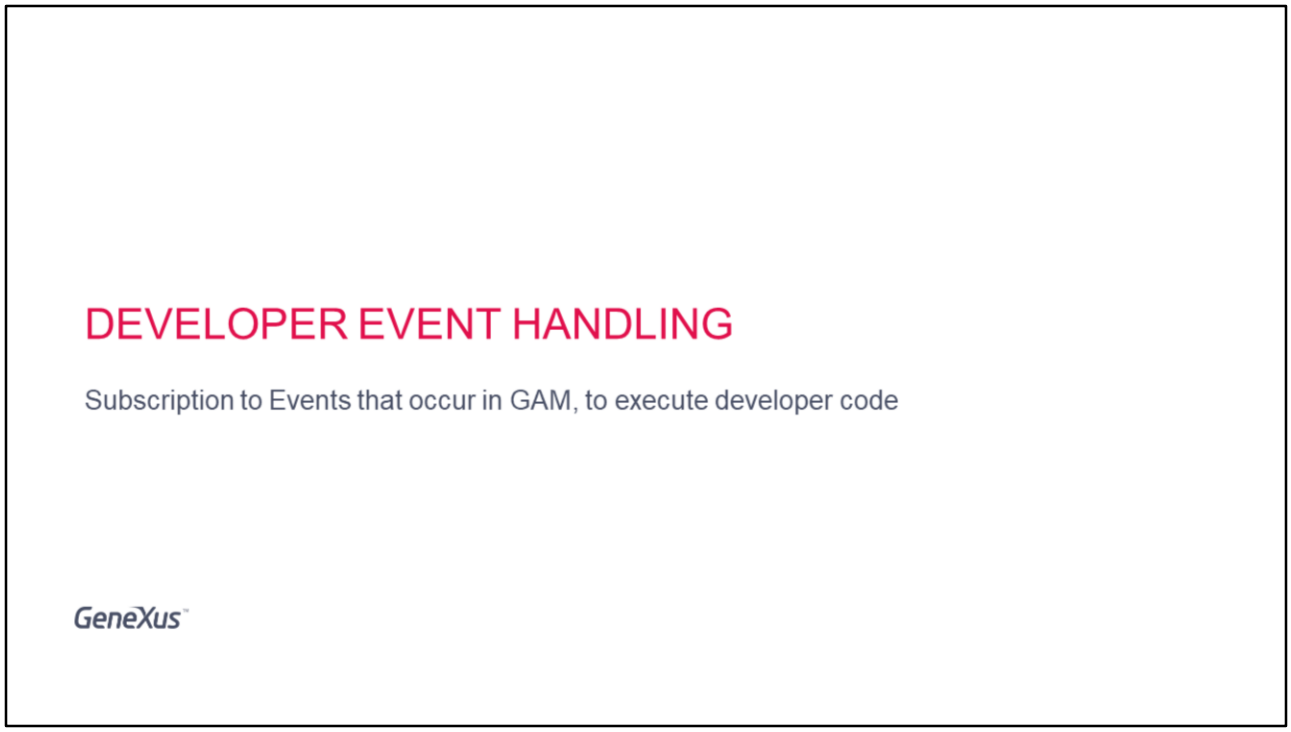

GeneXus™
by Globant

DEVELOPER EVENT HANDLING

Nicolas Adrién

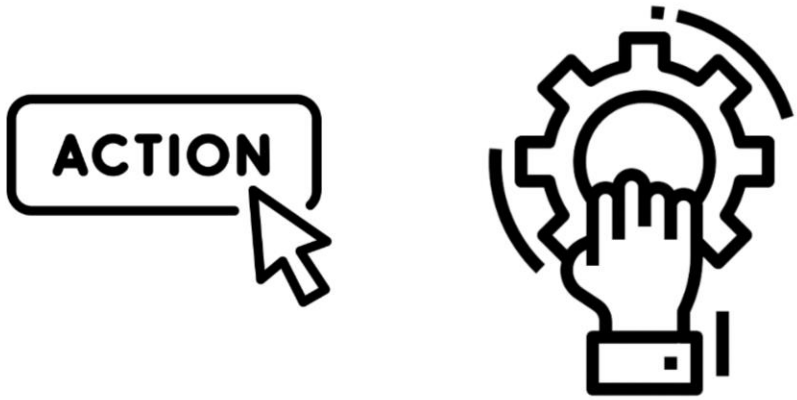
GeneXus



En este video hablaremos sobre la posibilidad de suscripción a eventos que ocurren en GAM, donde se tiene la posibilidad de ejecutar código a cargo de un desarrollador en el momento que estos eventos ocurren.

GeneXus

Purpose



El GAM nos brinda la posibilidad de suscribirnos a distintos eventos que ocurren en las aplicaciones, ya sea por una acción propia de un usuario, o que se desencadene a partir de otra.

El propósito de esto es poder ejecutar código adicional implementado por el desarrollador GeneXus en eventos predefinidos que ofrece GAM

GeneXus

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 Client			
One Time Password Valid User			
One Time Password Generate Code			
One Time Password Send Code			
One Time Password Validate Code			

En los posibles eventos que nos podemos suscribir, tenemos cuatro categorizaciones. Primero tenemos por Usuario, aquí tenemos:

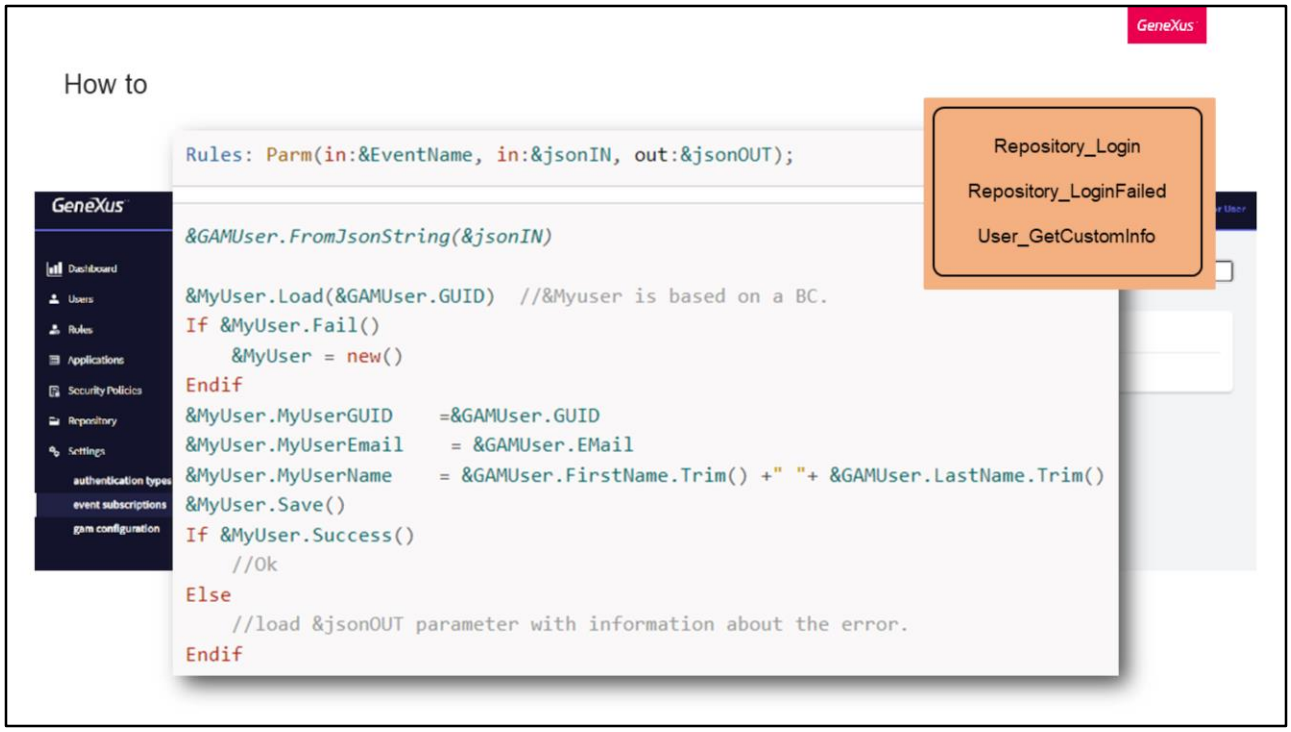
- Insert: Desencadenado por una inserción de un Usuario GAM
- Update: Por la actualización de un Usuario GAM
- Delete: Por la eliminación de un Usuario GAM
- UpdateRoles: Ocurre cuando se cambian los roles de un Usuario GAM
- GetCustomInfo: Ocurre en un IDP Server, y permite ejecutar un código para obtener información personalizada del usuario para enviarla al cliente GAMRemote.
- (Se elimina Autentication: Se ejecutausando elusuario inia sesión en el servidor y tambien en el inicio de Sessión SS0.)
- SaveCustomInfo: Permite leer la información personalizada enviada desde el IDP server y ejecutar código para procesar esa información en el Cliente y por ejemplo, poder guardarla en las tablas que el sistema necesite
- OneTimePasswordValidUser: Permite incluir código para que el desarrollador valide al usuario que solicitó un código OTP
- OneTimePasswordGenerateCode: Es un evento donde el desarrollador puede personalizar como se genera el código OTP que se enviará al usuario
- OneTimePasswordSendCode: Es un evento que permite personalizar el envío del código OTP, ya sea por SMS, notificación, email, etc. De forma predeterminada GAM envía mediante un email
- OneTimePasswordValidateCode: Es un evento del desarrollador utilizado para validar el código OTP

Después tenemos los eventos por Rol:
Aquí simplemente tenemos los clásicos Insert, Update y Delete.

Luego tenemos a los eventos de un Repositorio:

- Login: Ocurre cuando se produce un inicio de sesión de usuario de GAM, sin importar el tipo de autenticación
- LoginFailed: Ocurre cuando el inicio de sesión del usuario falla, solamente por nombre de usuario y/o contraseña incorrectos
- Logout: Se desencadena en el cierre de sesión de GAM
- External Authentication Response: Es un evento para personalizar la forma de procesar la respuesta de un IDP externo. Este evento debe interactuar con el IDP externo y terminar con el login local

Por último tenemos a los eventos de Aplicación, aquí solo tenemos Check Permission Fail, el cual se dispara cuando un permiso da denegado. Esto se podría utilizar por ejemplo para grabar un log de los permisos que se intentan verificar y dieron denegados.



Veamos como suscribirnos a un evento a través del backoffice web de GAM.

Para hacerlo nos dirigimos a Settings/Suscripción de eventos, y presionamos Add.

Allí podemos ingresar una descripción, seleccionar a cual evento nos queremos suscribir, el nombre del archivo (esto es el nombre del archivo .dll o .class que va a escuchar la ejecución del evento), el nombre de la clase (esto es el nombre del programa incluyendo su paquete en el caso de Java), y finalmente el nombre del método (el método del programa en GeneXus siempre es "execute“).

Una posible implementación de un procedimiento que notifica sobre el evento de notificación al usuario podría ser la siguiente, la cual en caso de que el método Success falle, carga el JSON de salida con la información del error.

Prestar especial atención con las reglas que deben tener los procedimientos que realizamos, ya que estos deben recibir el nombre del evento y el JSON de entrada el cual tendrá información sobre el evento.Se elimina: Y como parámetro de salida otro JSON.

El JSON de salida solo es utilizado por determinados eventos:

- Repository_Login y Repository_LoginFailed, donde debe retornar vacío si está OK, y el objeto GAMError si hay un error.
- User_GetCustomInfo donde el parámetro de salida debe tener el JSON a

enviar al cliente.

GeneXus

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
```

Veamos algunos casos de uso de suscripciones a eventos.

Repository_Login.

Como decíamos antes, este evento ocurre cuando se produce un inicio de sesión de usuario de GAM, sin importar el tipo de autenticación, el cual permite cancelar el login y desplegar un mensaje al usuario final.
Por ejemplo, si para loguearse en una aplicación el usuario debe tener un rol determinado, se podría validar y retornar un error.

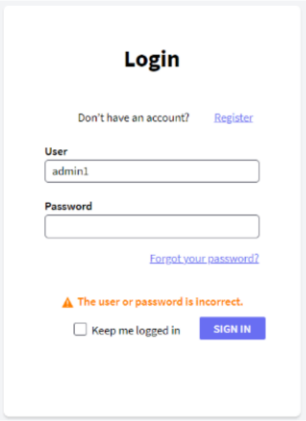
Una posible implementación para esto podría ser la siguiente.
Chequeamos los roles de la sesión comparando por el rol 170, donde ese rol seria el que debe tener el usuario, y si lo tiene, no hacemos nada.

En cambio si no lo tiene, seteamos el GAMError con la información que queramos y debemos detener el login. Para hacer esto, se debe retornar un JSON del objeto GAMError.
Esto ultimo tambien lo hacemos cuando no se tenga roles, ya que seria el mismo caso pero con distinto mensaje de error.

GeneXus

Use cases

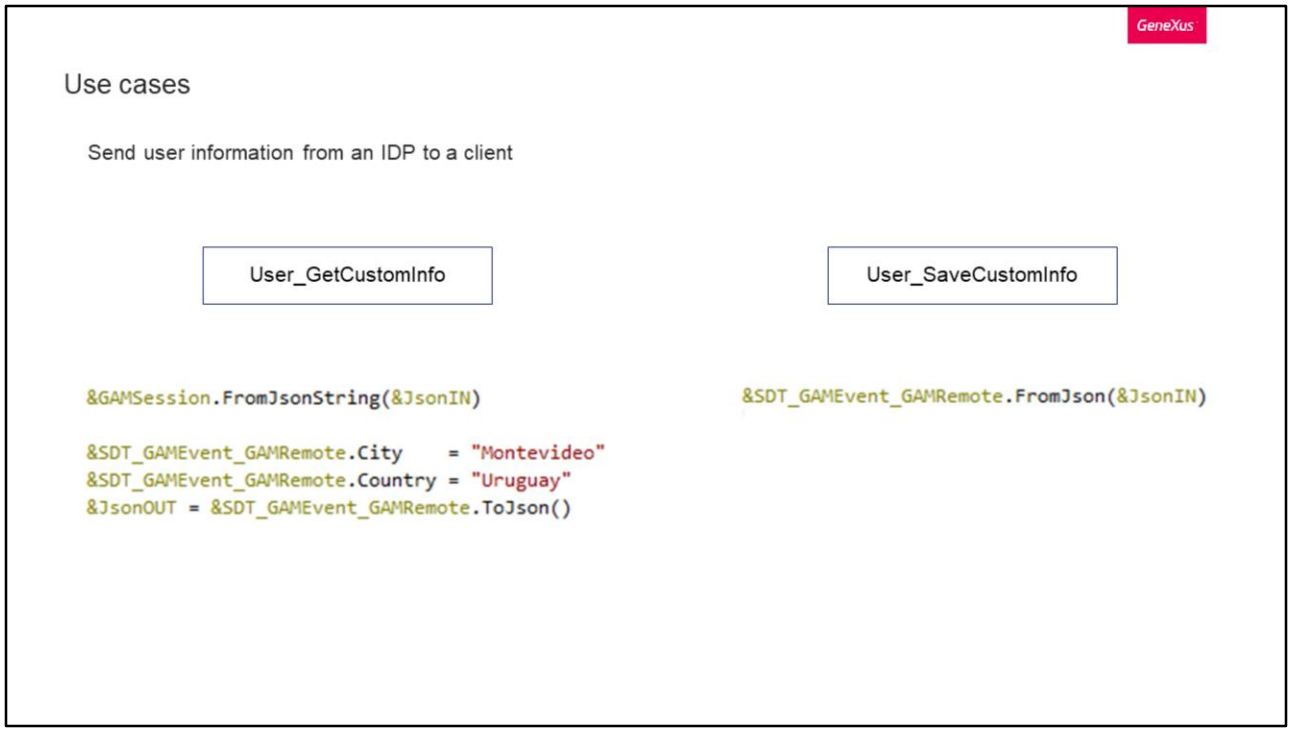
Repository_LoginFailed




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

Sigamos con los eventos de Repositorio, pero esta vez con LoginFailed.
Como decíamos antes, este evento solo se origina cuando el login de un usuario falla con error 11 o 18 de GAM, los cuales representan contraseña o nombre de usuario incorrectos.

Nuestro objetivo es loguear cada intento fallido de los usuarios.
Para eso debemos suscribirnos al evento LoginFailed, y en el procedimiento trabajar a nuestro gusto con el JSON que recibiremos.
En este tipo de evento, el JSON se corresponderá al External Object GAMSession, por lo que podemos incluir todas sus propiedades en el mensaje de error que loguearemos.



Otro caso de uso importante podría ser cómo desde un IDP hecho con GAM mandar cualquier tipo de información a un cliente. Para esto usaremos los eventos de Usuario.

Para esto, en el IDP debemos tener activado el evento GetCustomInfo de la sección de Usuario. Lo recomendable aca es cargar los datos que queramos enviar, y mandarlos como un JSON.

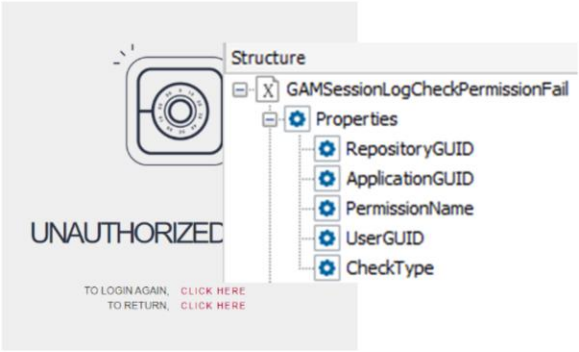
Luego, desde el lado del cliente vamos a tener que suscribirnos al evento SaveCustomInfo, tambien de Usuario, y allí recibir el JSON enviado para poder consultar la información que queríamos enviar y recibir.
Obviamente la información recibida debe ser procesada por el desarrollador, y guardarla en las tablas que la requerían.

Y eso es todo.

GeneXus

Use cases

Application_CheckPermissionFail




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

Ahora veamos un caso asociado a las aplicaciones.
Nuestro objetivo es al igual que en el LoginFailed que vimos, loguear todos los intentos de acceso a objetos donde el usuario no tiene permiso.
Para eso debemos suscribirnos al evento CheckPermissionFail, y en el procedimiento trabajar a nuestro gusto con el JSON que recibiremos.
En este tipo de evento, el JSON se corresponderá al External Object GAMSessionLogCheckPermissionFail, por lo que podemos incluir las siguientes propiedades en el mensaje de error que loguearemos.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com