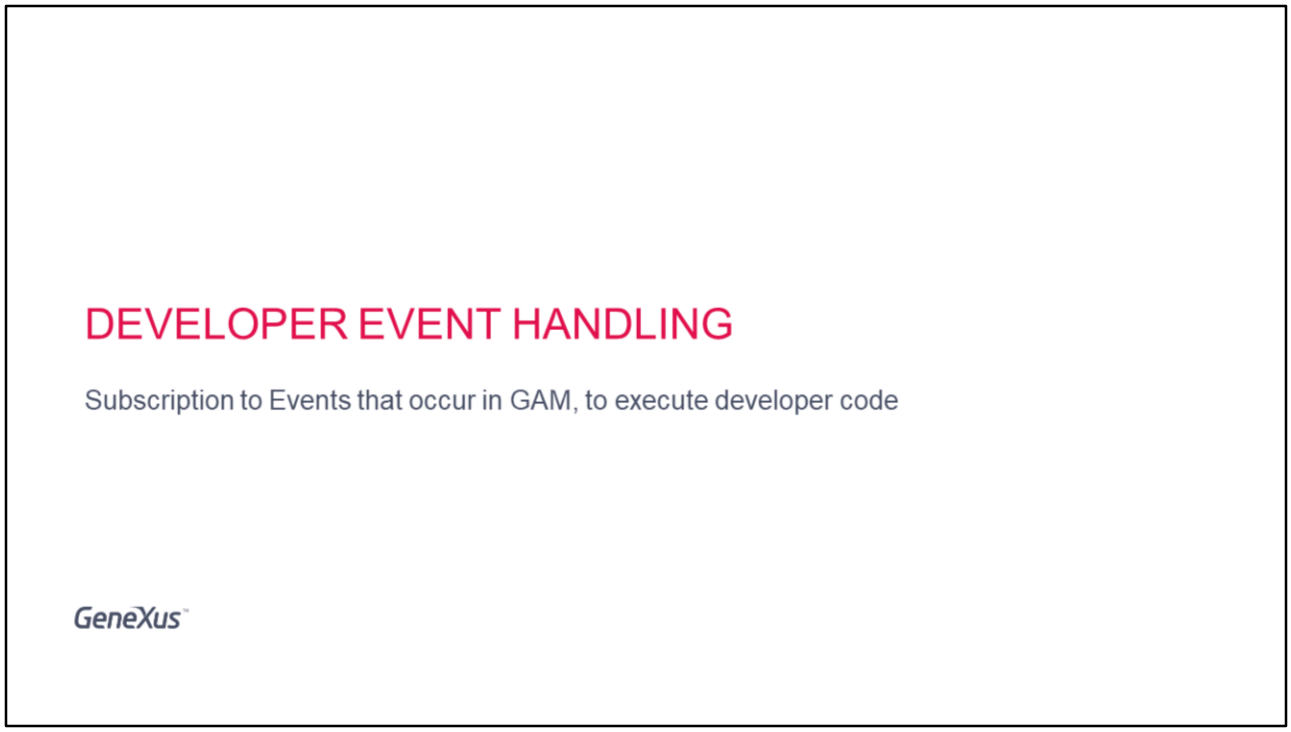

GeneXus™
by Globant

DEVELOPER EVENT HANDLING

Nicolas Adrién

GeneXus



Neste vídeo falaremos sobre a possibilidade de subscrição em eventos que ocorrem no GAM, onde existe a possibilidade de executar código sob responsabilidade de um desenvolvedor no momento em que estes eventos ocorrem.

GeneXus

Purpose

O GAM nos oferece a possibilidade de subscrever diferentes eventos que ocorrem nas aplicações, seja por uma ação própria de um usuário, ou que seja desencadeada a partir de outra.

O objetivo disto é poder executar código adicional implementado pelo desenvolvedor GeneXus em eventos predefinidos oferecidos pelo 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			

Nos possíveis eventos que podemos subscrever, temos quatro categorizações.
Primeiro temos por Usuário, aqui temos:

- Insert: Acionado por uma inserção de um Usuário GAM
- Update: Pela atualização de um Usuário
- Delete: Pela exclusão de um Usuário
- UpdateRoles: Ocorre quando são alteradas as roles de um Usuário GAM
- GetCustomInfo: Ocorre em um IDP, e permite executar um código para obter informação personalizada do usuário para enviar ao cliente GAMRemote.
- SaveCustomInfo: Permite ler a informação personalizada enviada do IDP server e executar código para processar essa informação no Cliente e, por exemplo, poder salvá-la nas tabelas que o sistema necessita
- OneTimePasswordValidUser: Permite incluir código para que o desenvolvedor valide o usuário que solicitou um código OTP
- OneTimePasswordGenerateCode: É um evento onde o desenvolvedor pode personalizar como é gerado o código OTP que será enviado ao usuário
- OneTimePasswordSendCode: É um evento que permite personalizar o envio do código OTP, seja por SMS, notificação, e-mail, etc. Por padrão, o GAM envia por e-mail
- OneTimePasswordValidateCode: É um evento do desenvolvedor utilizado para validar o código OTP

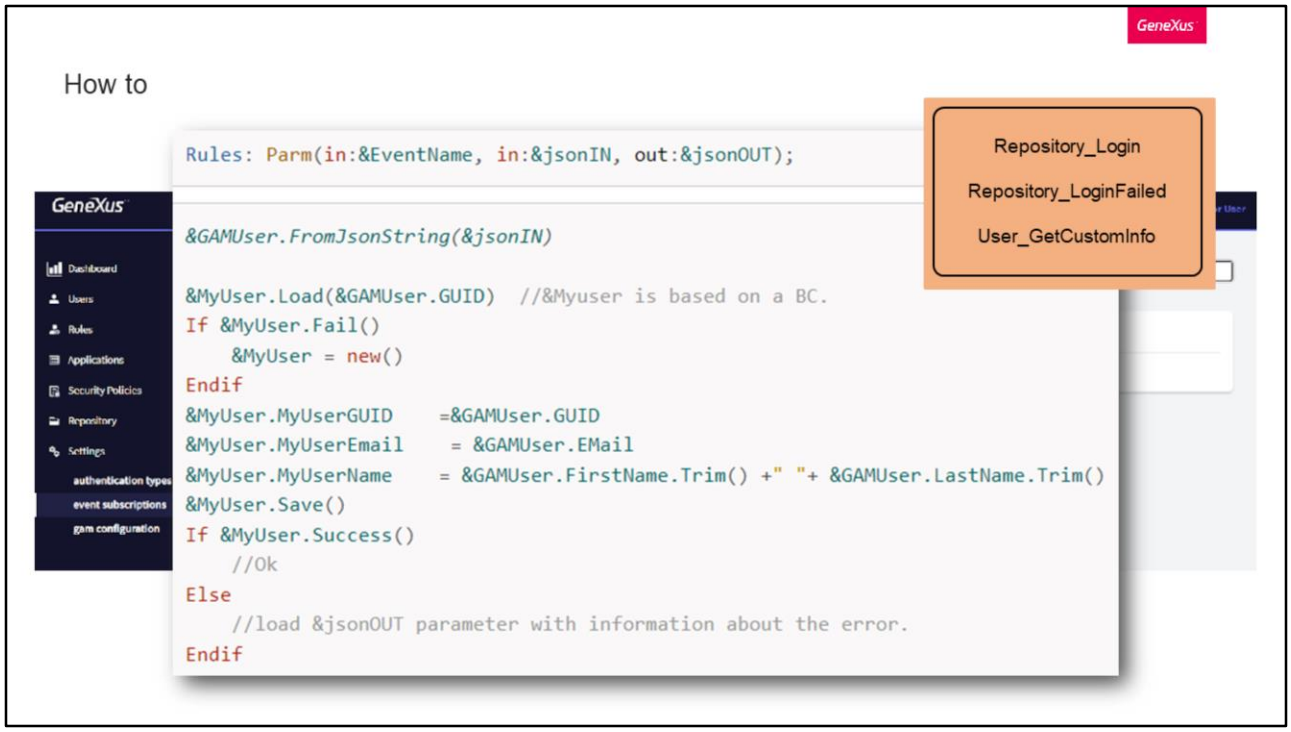
Depois temos os eventos por Role:

Aqui simplesmente temos os clássicos Insert, Update e Delete.

Então temos os eventos de um Repositório:

- Login: Ocorre quando é produzido um login de usuário do GAM, sem importar o tipo de autenticação
- LoginFailed: Ocorre quando o login do usuário falha, apenas devido a nome de usuário e/ou senha incorretos
- Logout: É acionado no logout do GAM
- External Authentication Response: É um evento para personalizar a forma de processar a resposta de um IDP externo. Este evento deve interagir com o IDP externo e terminar com o login local

Por último temos os eventos de Aplicação, aqui temos apenas Check Permission Fail, que é disparado quando uma permissão é negada. Isto poderia ser utilizado, por exemplo, para gravar um log das permissões que tentam ser verificadas e foram negadas.



Vejamos como subscrever um evento através do backoffice web do GAM.

Para fazer isso, vamos para Settings/Event subscriptions e pressionamos Add.

Lá podemos inserir uma descrição, selecionar qual evento queremos subscrever, o nome do arquivo (este é o nome do arquivo .dll ou .class que irá escutar a execução do evento), o nome da classe (este é o nome do programa incluindo seu pacote no caso de Java) e finalmente o nome do método (o método do programa em GeneXus é sempre "execute“).

Uma possível implementação de um procedimento que notifica sobre o evento de notificação o usuário, poderia ser a seguinte, que no caso em que o método Success falhe, carrega o JSON de saída com a informação do erro.

Prestar especial atenção às regras que devem ter os procedimentos que realizamos, pois devem receber o nome do evento e JSON de entrada que terá informação sobre o evento.
O JSON de saída é utilizado apenas por determinados eventos:

- Repository_Login e Repository_LoginFailed, onde deve retornar vazio se está OK, e o objeto GAMError se houver um erro.
- User_GetCustomInfo onde o parâmetro de saída deve ter o JSON para enviar ao 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
```

Vejamos alguns casos de uso de subscrições de eventos.

Repository_Login.

Como dissemos anteriormente, este evento ocorre quando é produzido um login de usuário do GAM, sem importar o tipo de autenticação, que permite cancelar o login e exibir uma mensagem ao usuário final.
Por exemplo, se para efetuar login em uma aplicação o usuário deve ter uma role determinada, poderia ser validada e retornar um erro.

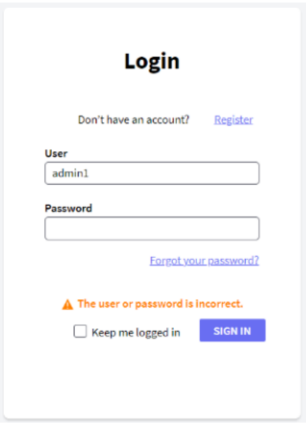
Uma possível implementação para isto poderia ser a seguinte.
Verificamos as roles da sessão comparando pela role 170, onde essa role seria aquela que deve ter o usuário, e se tiver, não fazemos nada.

Em vez disso, se não tiver, configuramos o GAMError com a informação que queremos e devemos interromper o login. Para fazer isso, deve ser retornado um JSON do objeto GAMError.
Este último também fazemos quando não se tenha roles, pois seria o mesmo caso, mas com diferente mensagem de erro.

GeneXus

Use cases

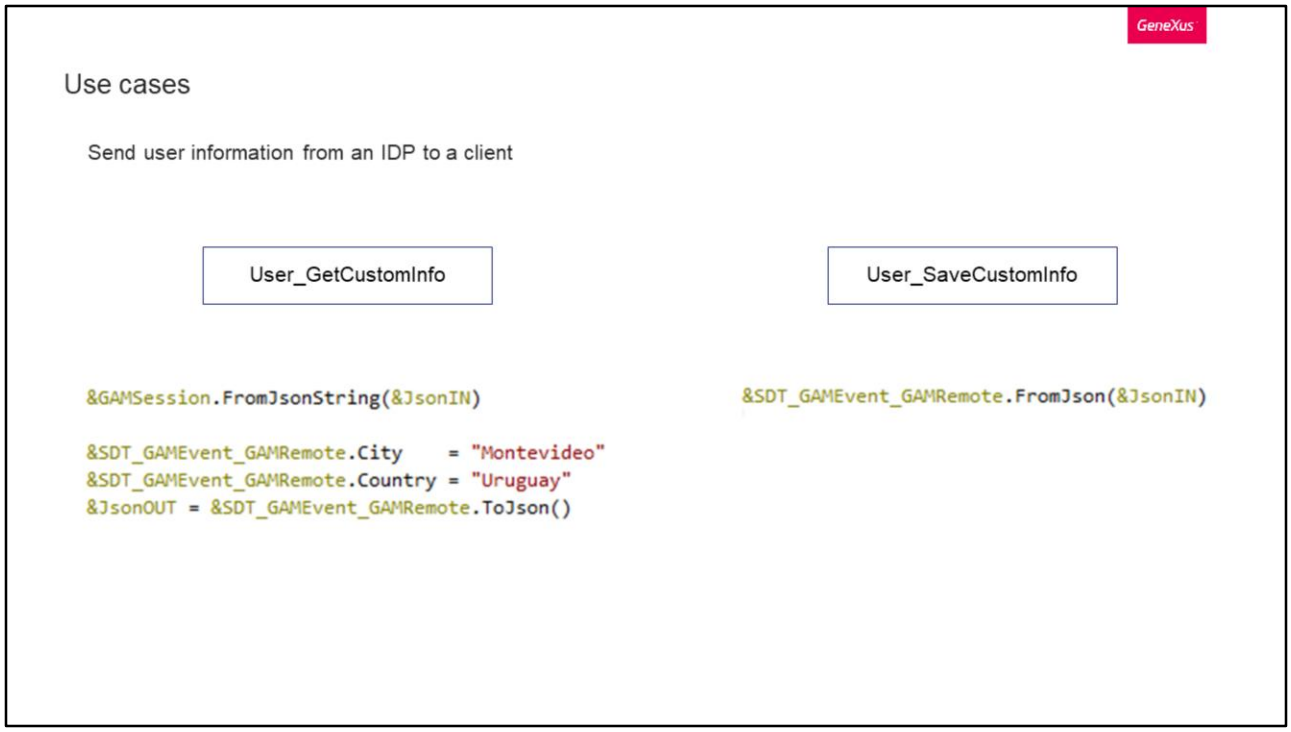
Repository_LoginFailed




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

Vamos continuar com os eventos de Repositório, mas desta vez com LoginFailed.
Como dissemos antes, este evento só é originado quando o login de um usuário falha com erro 11 ou 18 do GAM, que representam senha ou nome de usuário incorretos.

Nosso objetivo é registrar cada tentativa malsucedida dos usuários.
Para isso devemos nos inscrever no evento LoginFailed, e no procedimento trabalhar a nosso gosto com o JSON que receberemos.
Neste tipo de evento, o JSON corresponderá ao External Object GAMSession, pelo que podemos incluir todas as suas propriedades na mensagem de erro que iremos registrar.



Outro caso de uso importante poderia ser, como a partir de um IDP feito com GAM enviar qualquer tipo de informação para um cliente. Para isto usaremos os eventos de Usuário.

No IDP devemos ter ativado o evento GetCustomInfo da seção de Usuário. O recomendado aqui é carregar os dados que queremos enviar, e enviá-los como um JSON.

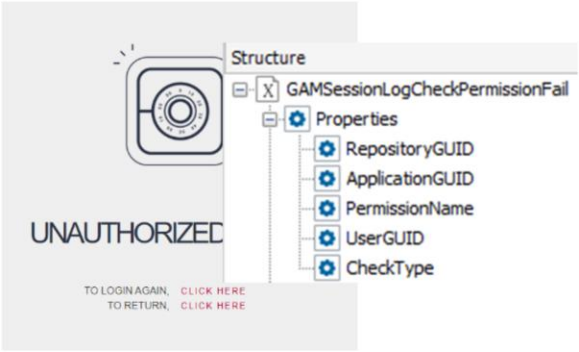
Depois, do lado do cliente vamos ter que subscrever o evento SaveCustomInfo, também de Usuário, e ali receber o JSON enviado para poder consultar a informação que queríamos enviar e receber.
Obviamente, a informação recebida deve ser processada pelo desenvolvedor e armazenada nas tabelas que a necessitavam.

E isso é tudo.

GeneXus

Use cases

Application_CheckPermissionFail




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

Agora vamos ver um caso associado às aplicações.
Nosso objetivo é, assim como no LoginFailed que vimos, registrar todas as tentativas de acesso a objetos onde o usuário não tem permissão.
Para isso devemos subscrever o evento CheckPermissionFail, e no procedimento trabalhar a nosso gosto com o JSON que receberemos.
Neste tipo de evento, o JSON corresponderá ao External Object GAMSessionLogCheckPermissionFail, pelo que podemos incluir as seguintes propriedades na mensagem de erro que iremos registrar.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com