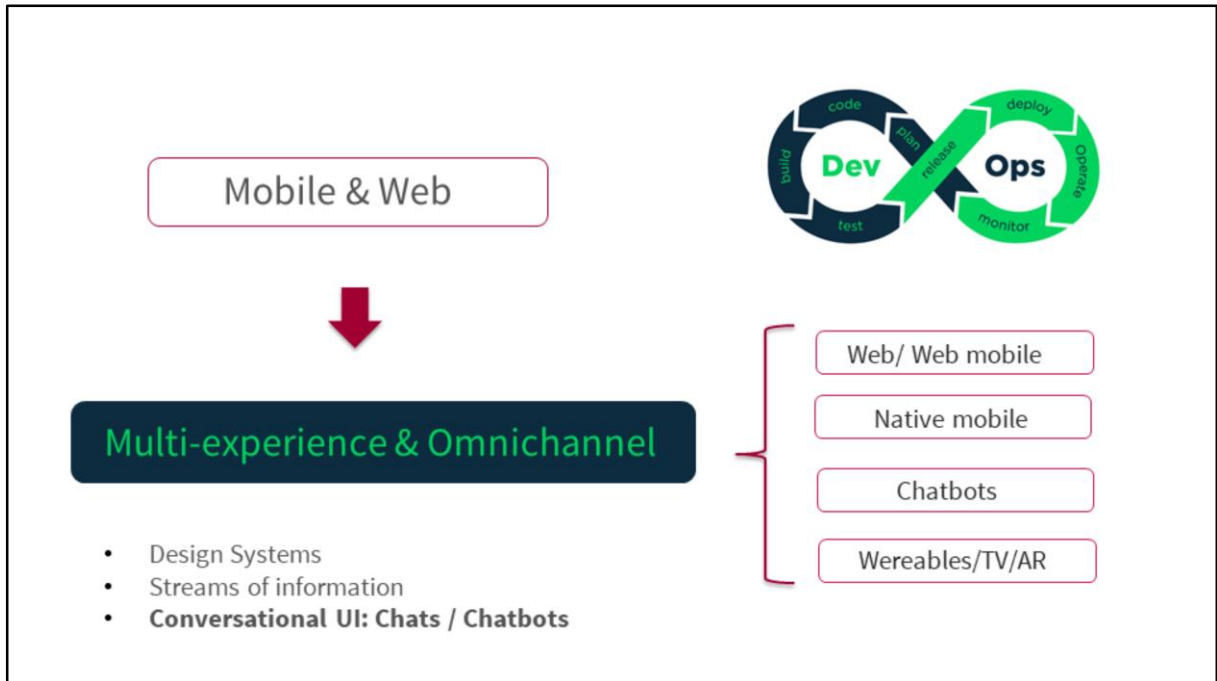


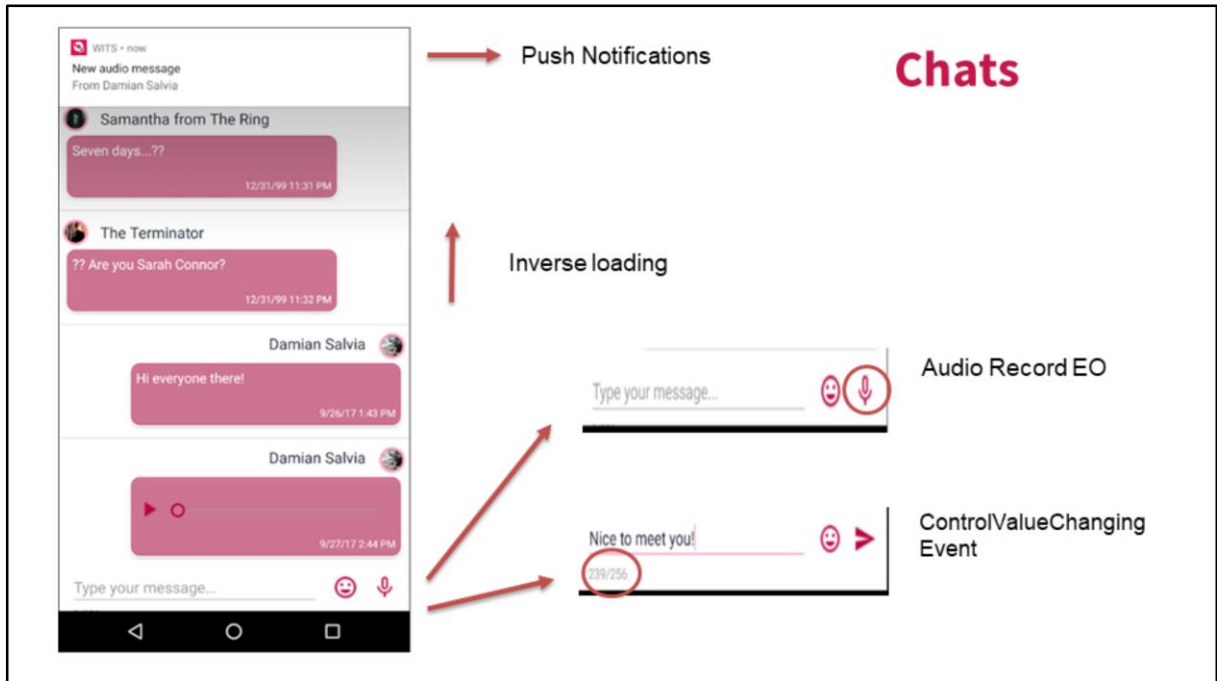
GeneXus™
The power of doing

Conversational UI: Chats / Chatbots

GeneXus™ 16



Ahora veremos el siguiente punto : las características para implementar Chats Simples y las facilidades para implementar Chatbots.



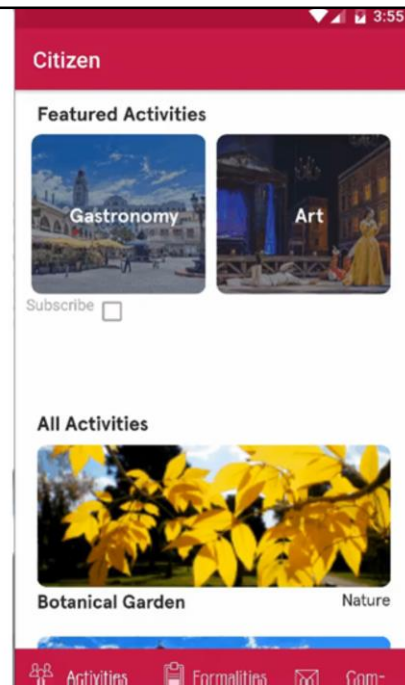
Para implementar un chat simple contamos con:

- Inverse Loading, la cual es una propiedad de un grid que permite visualizar la última información de abajo hacia arriba
- Audio Record, un external object que permite grabar un audio
 - Control Value Changing, es un evento que permite saber si el usuario ha cambiado el valor de un campo editable (numérico o character), en este ejemplo se puede utilizar para llevar el conteo de los caracteres del mensaje a enviar.
- Push Notifications, que permiten informar al usuario sobre algún evento mediante una alerta, en este contexto, informamos que hay una nueva actividad de interés

Push Notification Demo

Nos enfocaremos ahora en las Push Notification y la implementación de la Socket API, sobre las novedades de la versión 16 hablaremos más tarde.
Veamos un ejemplo en ejecución

Subscribing



Desde nuestra aplicación citizen el usuario tiene la opción de suscribirse para recibir notificaciones de las nuevas actividades que existan.

Inserting from Backend Web

Citizen Service















by GeneXus

RecentCultural Activity — Cultural Activities

Cultural Activities

Activity Name

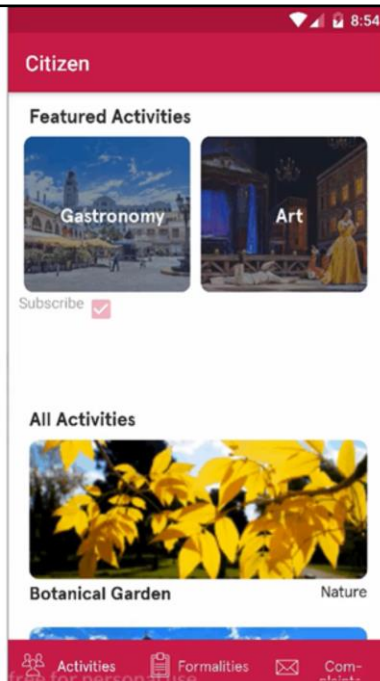
+ INSERT

Name	Category	Photo	Featured	Static Map		
Botanical Garden	Nature		<input type="checkbox"/>		UPDATE	DELETE
Carnaval Museum	Culture		<input type="checkbox"/>		UPDATE	DELETE
Centenario Stadium	Sports		<input checked="" type="checkbox"/>		UPDATE	DELETE
Dámaso Antonio Larrañaga Zoological Museum	Culture		<input type="checkbox"/>		UPDATE	DELETE
Japanese Garden	Nature		<input type="checkbox"/>		UPDATE	DELETE
Juan Manuel Bienes Museum	Art		<input type="checkbox"/>		UPDATE	DELETE
Mercado del Puerto	Gastronomy		<input checked="" type="checkbox"/>		UPDATE	DELETE

Posteriormente desde el backend web alguien realiza la inserción de una nueva actividad y al guardar se envía la notificación a los usuarios suscritos.

Receiving the Push Notification

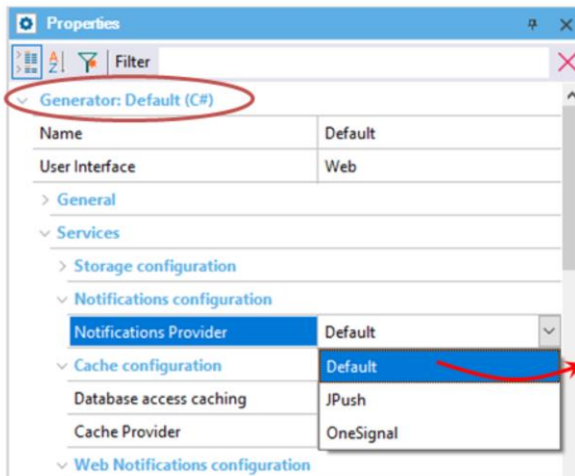
How..?



El usuario en su dispositivo recibe la notificación y al dar tap sobre ésta, podrá ver el detalle de la nueva actividad.

¿Y cómo se realiza esto?

Notifications Provider property



OneSignal (since GX15 u4)

JPush (since GX15 u8)

Default	GeneXus Notifications by using RemoteNotification external object.
JPush	JPush Notifications
OneSignal	OneSignal Notifications

Para ello contamos con la propiedad Notification Provider (Gx15 U4) a nivel de Generador web, permitiéndonos elegir el proveedor externo con el que manejaremos las notificaciones . Al elegir un proveedor u otro se nos desplegarán diversas propiedades para su configuración

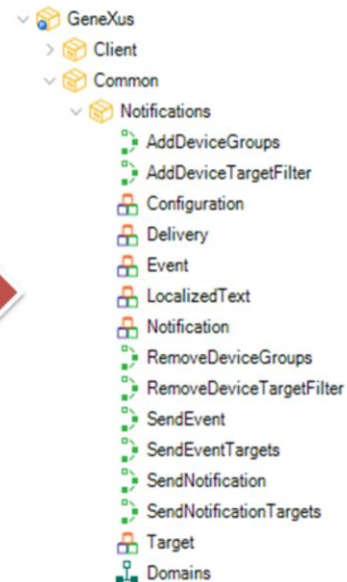
Nota: La principal diferencia entre OneSignal y Jpush es que este último se utiliza para sitios donde no se soporta el servicio de Google, principalmente se utiliza en app para China.

Notification Provider API

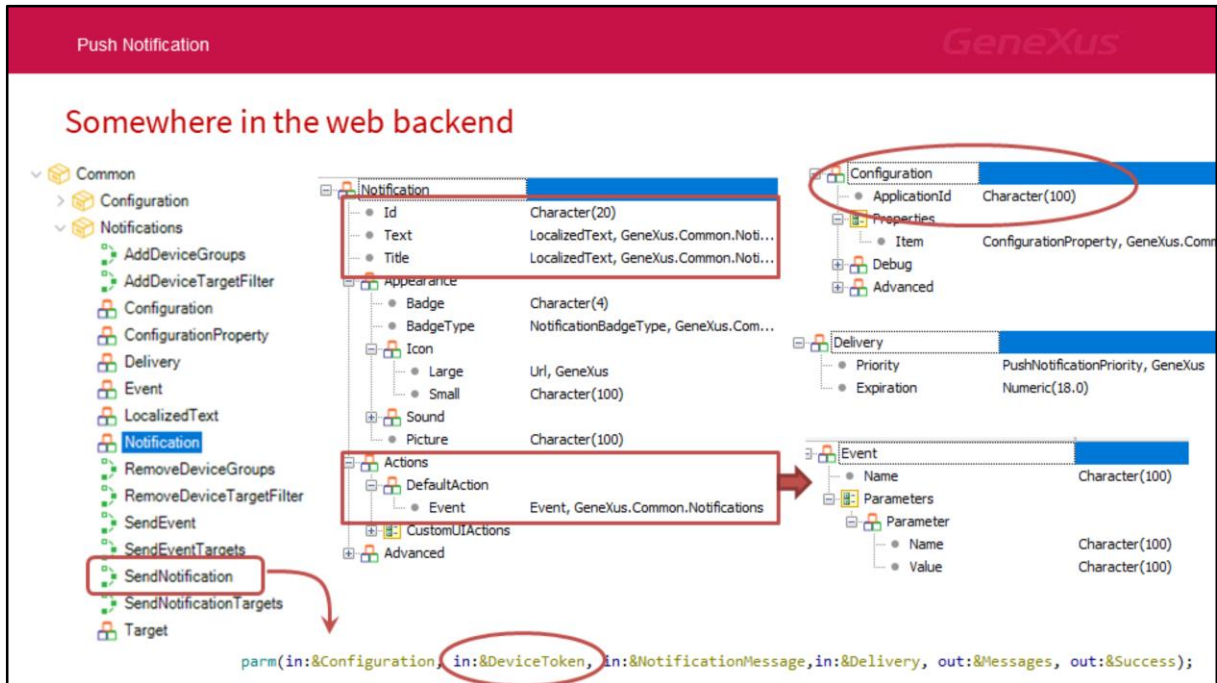
Notification parameter
Remote notification
Notification configuration
Remote notification result



(Since GeneXus 15 Upgrade 3)



Anteriormente utilizábamos varios External Object para el manejo de las Notificaciones; ahora los unificamos en una sola API de nombre Notification Provider, que encontramos en el módulo Genexus-Common-Notifications.



Con esta Api podemos configurar la notificación utilizando variables del tipo de los SDTs y enviándola desde el backend web.

Detengámonos a analizar los sdt que utilizaremos:

- El SDT Notification, nos permite configurar el contenido de la notificación así como la acción que desencadenará algún evento, por ejemplo abrir un sdpanel pasando parámetros.
- Configuration permite indicar el nombre del objeto main SD donde se ejecuta la notificación en su ítem Application ID.
- Delivery permite configurar prioridad de la notificación y expiración de la misma.

Por último tenemos algunos procedimientos para poder enviar la notificación, por ejemplo Send Notification, que se encarga de mandar la notificación al dispositivo con las configuraciones realizadas

Configuring the Notification

```

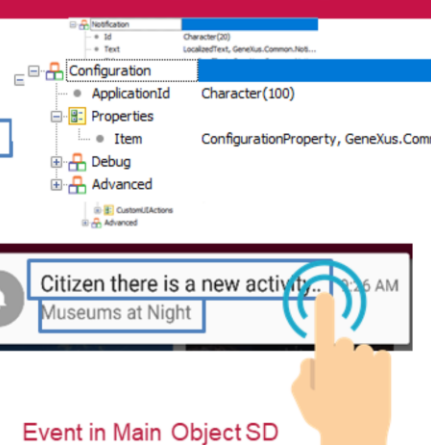
&TheNotification.Title.DefaultText = "Citizen there is a new activity for you"
&TheNotification.Text.DefaultText = &ActivityName
&TheNotification.Actions.DefaultAction.Event.Name = "Subscribe"
// Declared in the SD Main object
&TheNotification.Actions.DefaultAction.Event.Parameters.FromJson
('{"Name":"ActivityID","Value":'+&ActivityID.ToString()+'}')

&TheNotificationDelivery.Expiration = 3000
&TheNotificationDelivery.Priority = PushNotificationPriority.High

&TheNotificationConfiguration.ApplicationId = !"CitizenMenu"
// Name Main Object SD

```

&TheNotification is based in Notification SDT
 &TheNotificationDelivery is based in Delivery SDT
 &TheNotificationConfiguration is based Configuration SDT



The screenshot shows the GeneXus IDE interface. On the right, the 'Configuration' object is visible with properties like 'ApplicationId' (Character(100)) and 'Item' (ConfigurationProperty, GeneXus.Common...). Below this, a mobile notification UI is shown with a bell icon, the text 'Citizen there is a new activity...', and 'Museums at Night' with a timestamp '10:46 AM'. A hand icon is shown tapping the notification. Below the UI, the event configuration is displayed: 'Event \'Subscribe\'', 'ActivityDetail(&ActivityID)', and 'endevent'.

Dentro de la notificación podemos configurar:

- Título
- Texto
- El evento que fue definido en el objeto main sd y el que se encarga de abrir el Detalle de la actividad una vez que el usuario dio tap sobre la notificación

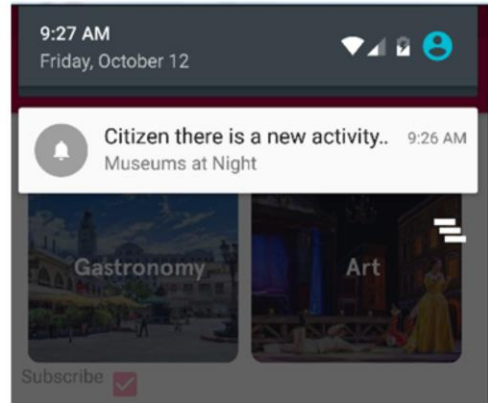
Después configuramos la prioridad y expiración de la notificación

Y por último configuramos el application ID el cual es el nombre de nuestro objeto main

Sending the Notification

```
For each Device
  Where DeviceSubscribe= true

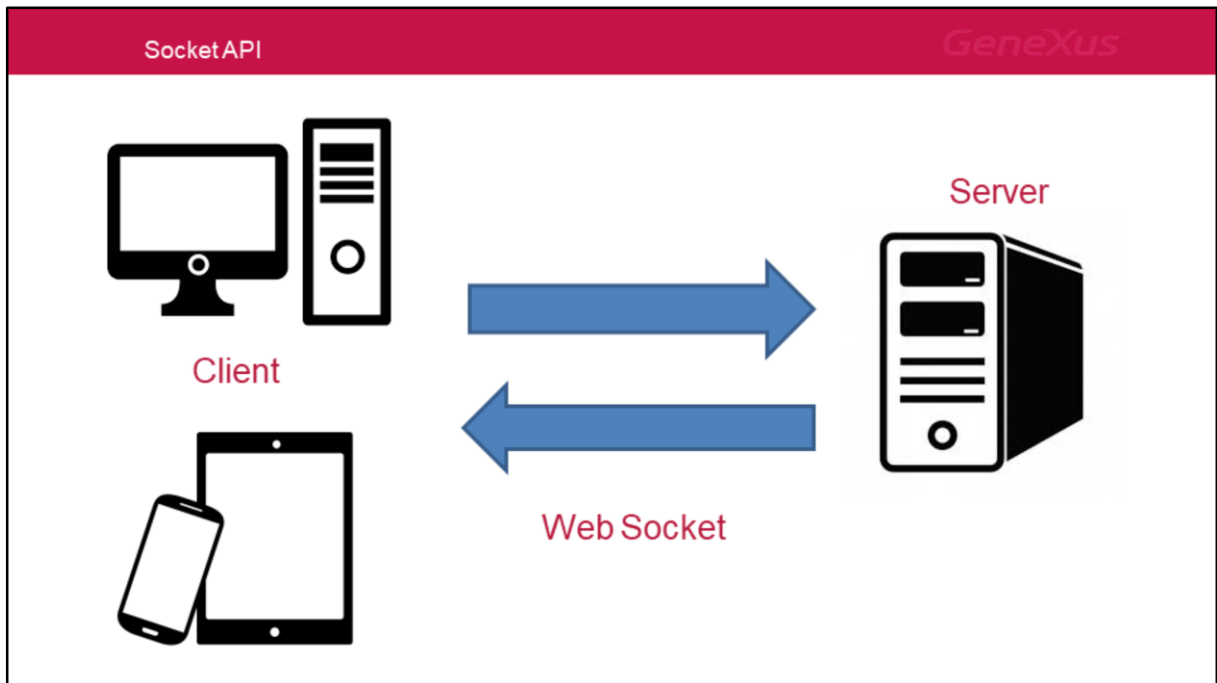
  GeneXus.Common.Notifications.SendNotification
  (
    &TheNotificationConfiguration,
    DeviceToken, // Target device token
    &TheNotification,
    &TheNotificationDelivery,
    &OutMessages,
    &IsSuccessful
  )
endfor
```



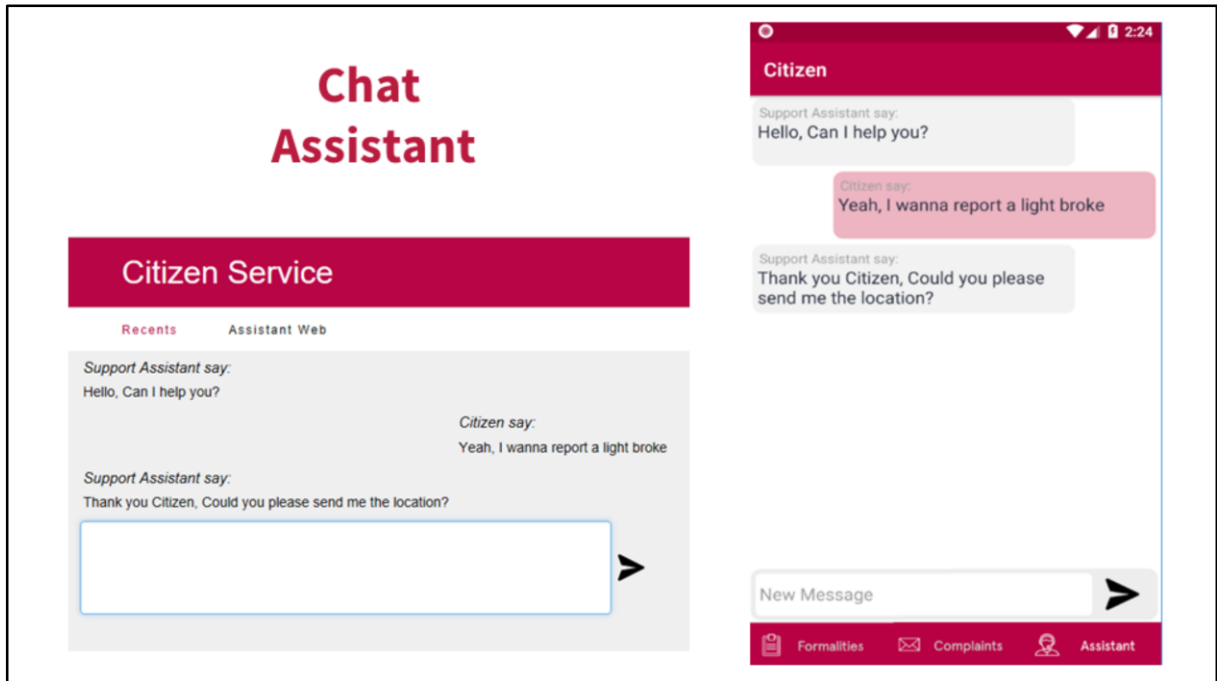
Por último debemos llamar al procedimiento SendNotification pasando las variables que configuramos e indicando el dispositivo que recibirá esta notificación, en este ejemplo, los usuarios que se subscribieron.

Socket API

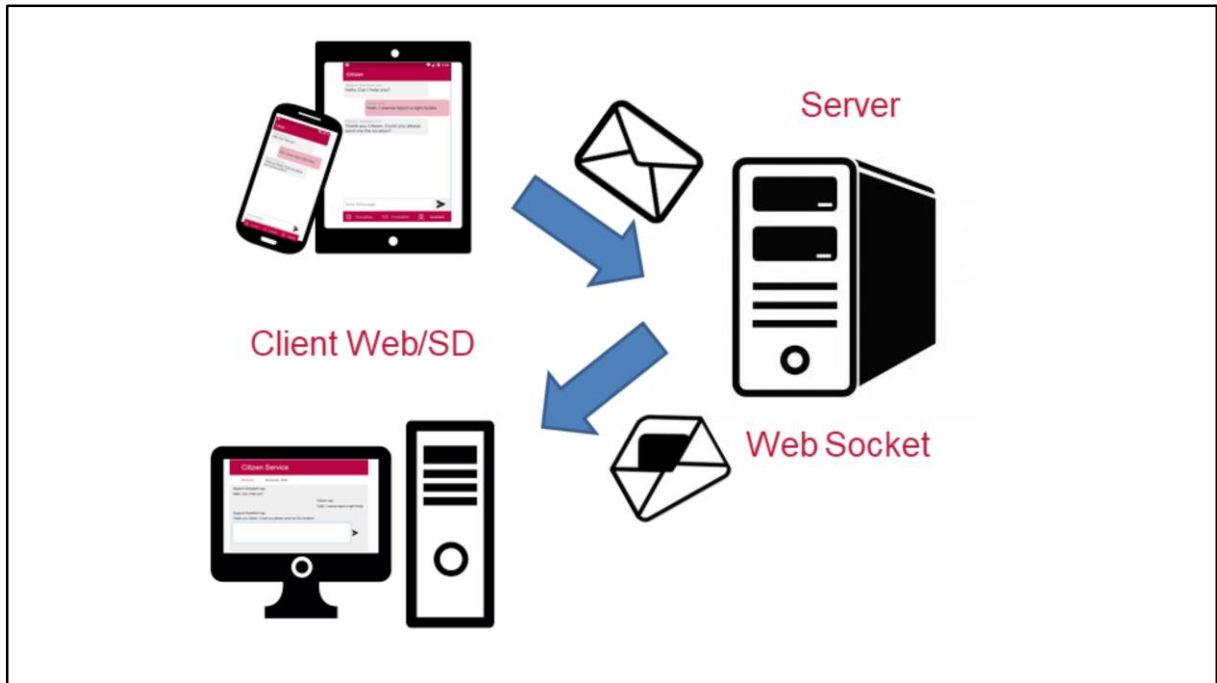
Pasemos al siguiente tema



Sobre la Socket API cabe destacar que nos permite establecer una comunicación bidireccional entre el cliente y servidor mediante los web sockets, que son un protocolo de comunicaciones que proporciona un canal de comunicación sobre una conexión tcp.

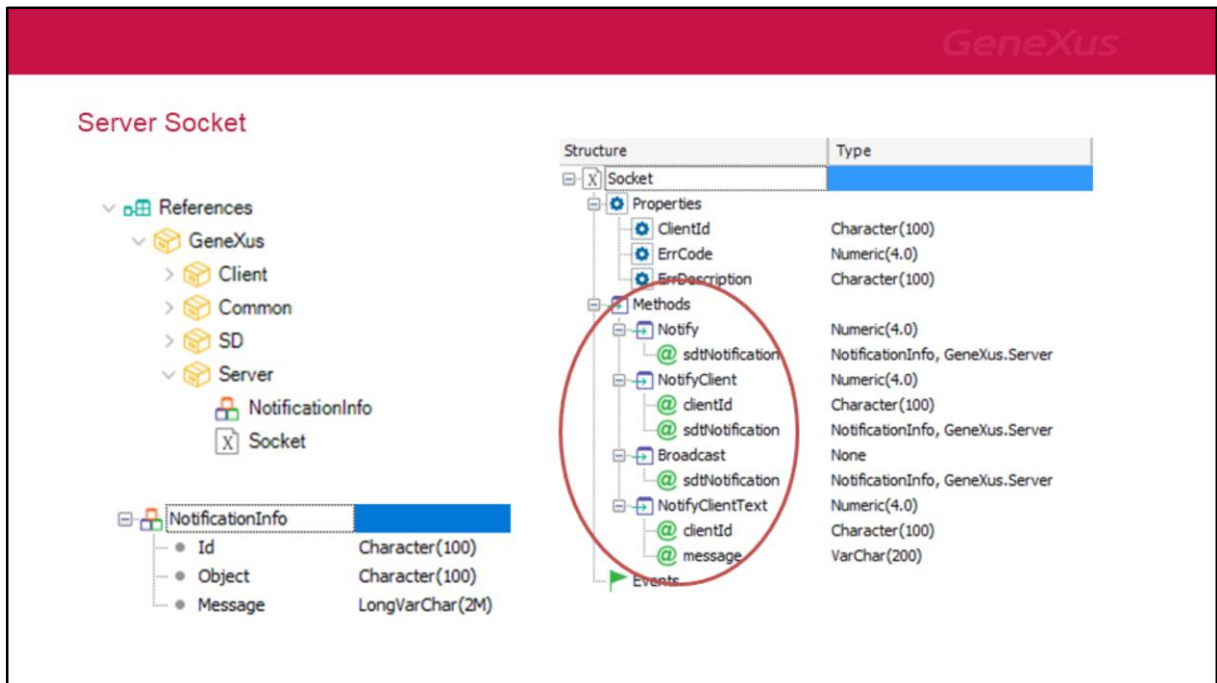


Un ejemplo de uso de la Socket API es un chat de Asistencia, que en nuestro ejemplo, le permite al usuario hacer un reporte en tiempo real desde su móvil y un asistente le apoyará desde la parte web.



El funcionamiento es el siguiente:

- El cliente SD envía el mensaje al Server
- El server lo envía al cliente WEB
- El cliente web refrescará su pantalla para mostrar el nuevo mensaje
- Y viceversa



Para ello contamos con Server Socket, que nos permite hacer la comunicación del server hacia los clientes, enviando una notificación utilizando algunos de los métodos, por ejemplo:

- Notify nos permite enviar una notificación al usuario que originó una acción.
- Notify client, este método nos permite enviar una notificación a un usuario específico.
- Broadcast el cual permite enviar notificaciones a todos nuestros usuarios.
- NotifyClientText permite enviar texto sin formato a un cliente en específico.

NotificationInfo, nos permite especificar la información de la notificación que se enviará Como ID de la notificación, el Objeto que recibirá esa notificación y el mensaje.

Opcional

También tenemos algunas propiedades, clienID la que nos permite obtener el id del cliente de la actual sesión, error code y error description los que nos devuelve información en caso de que sucediera algún error.

Sending Message from Web

Citizen Service

Recents Assistant Web

Support Assistant say:
Hello, Can I help you?

Citizen say:
Yeah, I wanna report a light broke

Support Assistant say:
Thank you Citizen, Could you please send me the location?

Event 'Send'

```
&commentNotificationInfo.PostId= random()  
&commentNotificationInfo.PostCommentContent= &NewMessage  
&NotificationInfo.Message= &commentNotificationInfo.ToJson()  
&ServerSocket.NotifyClient(&clientID,&NotificationInfo)  
Endevent
```

CommentNotificationInfo X	
Structure Documentation	
Name	Type
CommentNotificationInfo	Id
PostID	Id
PostCommentContent	Character(256)

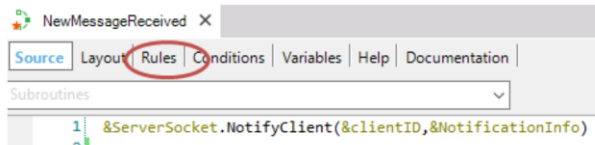
En nuestro ejemplo, enviamos un mensaje nuevo desde el chat Web (un web panel) al usuario con el que se está conversando (un Panel for Smart Devices).

Cabe destacar que el SDT CommentNotificationInfo lo creamos para enviar nuestro mensaje en formato json.

Receiving Message in Web

Web Notifications configuration

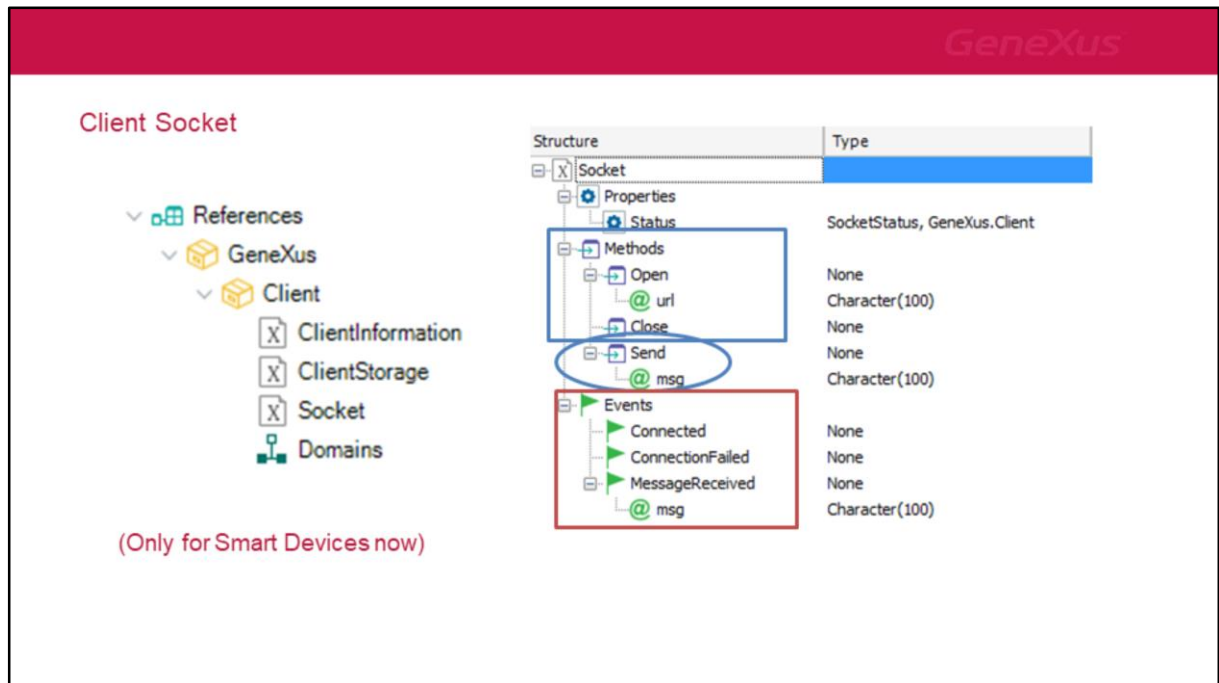
Web Notifications Provider	InProcess
Received Handler	NewMessageReceived
Open Handler	NewConnection
Close Handler	LostConnection
Error Handler	WSocketError



Event OnMessage(&NotificationInfo)
//code
refresh
Endevent

```
parm(in:&clientid, in:&NotificationInfo);
```

Posteriormente para recibir la respuesta necesitamos configurar a nivel de la Kb los procedimientos que manejarán la conexión y el mensaje recibido. Veamos que en nuestro caso tenemos el Procedimiento NewMessageReceived.
Por último para mostrarlo utilizamos el evento OnMessage en nuestro WebPanel y hacemos un refresh para cargar el nuevo mensaje que se mostrará.

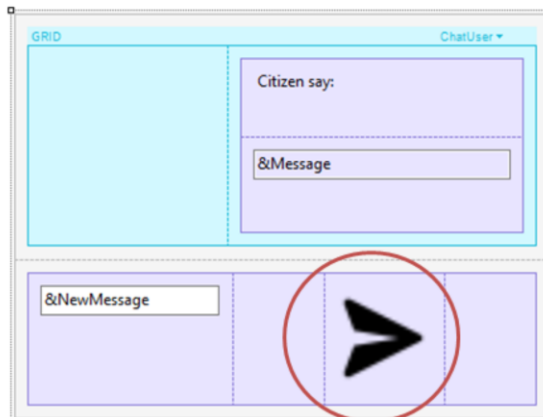


Por otro lado para la comunicacion entre el cliente y el server contamos con Client Socket que tiene métodos para poder establecer la conexión con el Web Socket y un método para enviar un mensaje.

Así mismo contamos con eventos que se ejecutan al establecer o no la conexión y al recibir un mensaje.

Cabe destacar que ahora Client Socket está disponible solo para Generadores Smart Devices.

Sending Message from SD

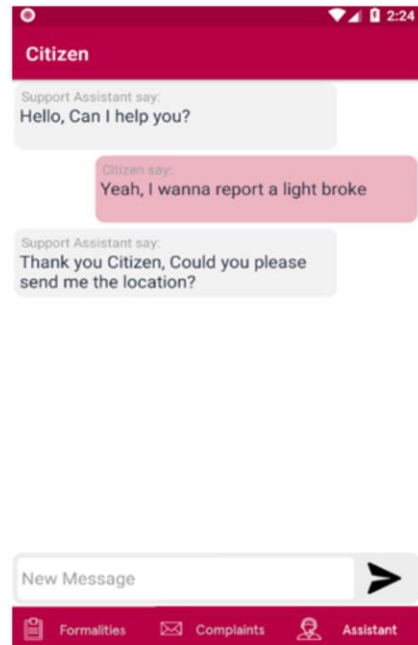


```
| Event ImageSend.Tap  
    GeneXus.Client.Socket.Send(&NewMessage)  
- Endevent
```

Siguiendo con nuestro ejemplo, desde la parte móvil configuramos el envío utilizando el método Send pasando la variable del nuevo mensaje.

Receiving Message in SD

```
Event Client.Socket.MessageReceived(&MessageReceived)
  Composite
    &NotificationInfo.FromJson(&MessageReceived)
    //code
    refresh
  EndComposite
EndEvent
```



Y para recibir los mensajes utilizamos el Evento MessageReceived para posteriormente desplegarlos al usuario.

Chatbot

CONVERSATIONAL USER INTERFACES



GeneXus

CHATBOT GENERATOR

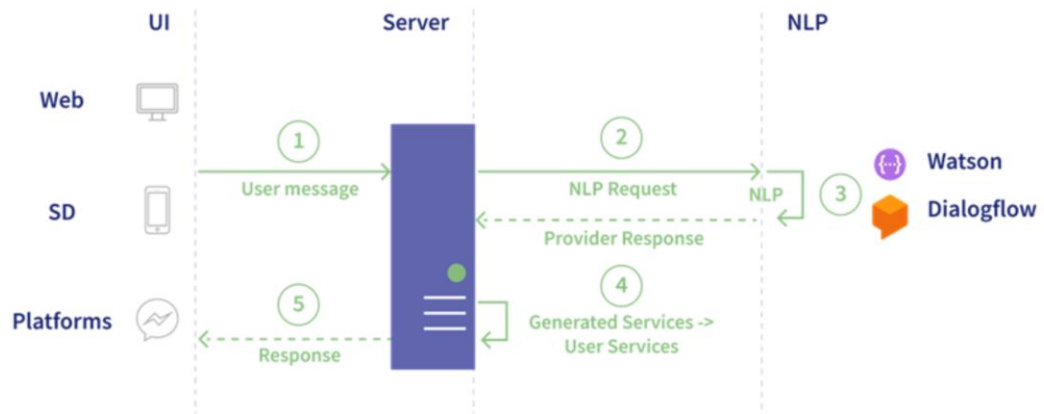
Cross-platform

Integrated to the solution

Hybrid UI

GeneXus

CHATBOT GENERATOR ARCHITECTURE



Chatbot Generator

New Object

Select a Category: Select a Type:

Common
Workflow
Reporting
Documentation
Web
Extensibility
Deploy
Chatbots
Smart Devices

Conversational Flows

Create a pattern instance of Conversational

Name: ConversationalFlows1

Description: Conversational Flows1

Module/Folder: Root Module

Create Cancel

Conversational X

Pattern Settings

Conversational Configuration

- Default Welcome Messages
- Default Welcome Triggers
- Default Insert Trigger Messages
- Default Search Trigger Messages
- Default Update Trigger Messages
- Default Delete Trigger Messages
- Transaction

Prop

Welc

Mess

CHATBOT GENERATOR OVERVIEW

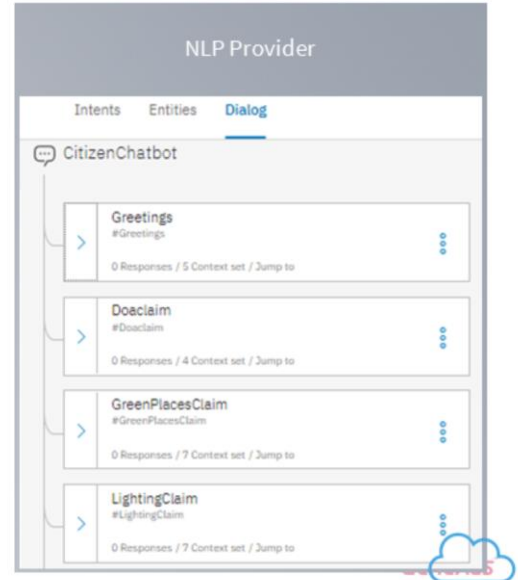


Conversational Flows

Chatbot
Generator

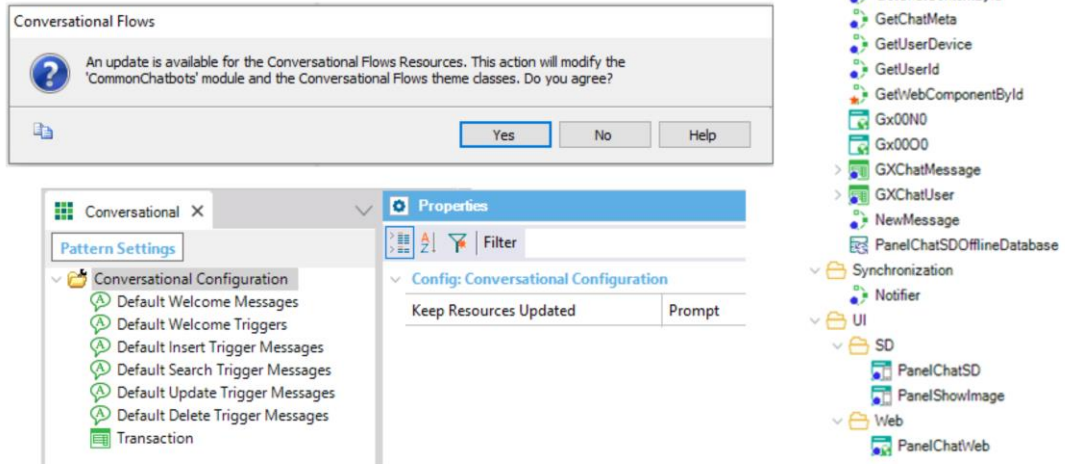


GeneXus objects generated



Resources update

CommonChatbots module update



Commonchatbots es un módulo que se actualiza cada vez que se actualiza el build de GX, que contiene recursos para implementar el chatbot.

Entre ellos, pueden ver los objetos siguientes

Chatbot Generator Resources

GXChatMessage, GXChatUser
Carmine, CarmineSD
PanelChatWeb, PanelChatSD

GXChatMessage es una tabla en la BD de la aplicación, donde se guardan los mensajes, la metadata, el usuario.

Themes – clases prefijadas con CF

PanelChatWeb y PanelChatSD. Notar que PanelChatSD es offline. Eso es para tener la conversación del usuario disponible, cuando está desconectado.

Estos objetos se pueden cambiar, dado que no se pasan por arriba en la generación del chatbot. Se reemplazan en cada update, dependiendo de la prop Keep Resources Update de los pattern settings.

GXChatMessageId -> Message Identifier (GUID)

GXChatUserId -> FK of the GXChatUser table (GUID)

GXChatMessageMessage -> The message (The text you see in the chat)

GXChatMessageType -> Message type (if it is a response or is from the user including its format)

GXChatMessageImage -> If you send an Image, it is saved there.

GXChatMessageDate -> DateTime of the message

GXChatMessageMeta -> Stores the metadata of the message (The JSON sent by the provider)

GXChatMessageRepeat -> Auxiliary for SD (see again)

GXChatUserDevice -> FK of the GXChatUser table

GXChatMessageInstance -> Instance of the conversational flow to which the message belongs

GXChatUser: Table in which users and their devices are stored.

GXChatUserId -> User identification (GUID)

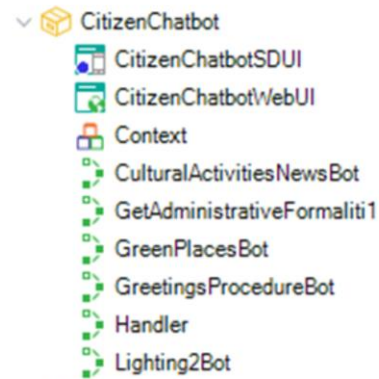
GXChatUserDevice -> User device identification

Generated Objects

Module containing generated objects

CitizenChatbotWebUI:

```
Event Start  
└─ CommonChatbots.PanelChatWeb(Chatbot.Conversational.Watson, !"Citizen")  
Endevent
```



El chatbot generator genera objetos que procesan el mensaje, lo mandan al proveedor, y luego la respuesta del proveedor se procesa nuevamente y mediante un handler se le envía la respuesta al usuario.

Entonces, tendremos un módulo con el nombre de la instancia, que contiene los objetos generados.

Por ejemplo, el objeto CitizenServiceChatWebUI, que es un panel main el que llama al PanelChatWeb

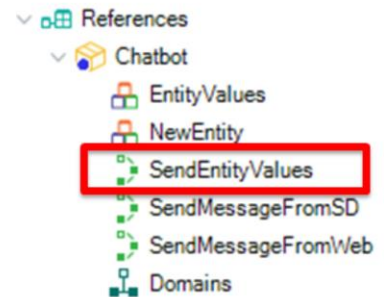
```
CommonChatbots.PanelChatWeb(Chatbot.Conversational.Watson, !"CitizenService")
```

<https://wiki.genexus.com/commwiki/servlet/wiki?37102,Chatbot+generator,#+2+Generated+objects+of+the+Pattern>

Chatbot External module

Communication with the NLP Providers

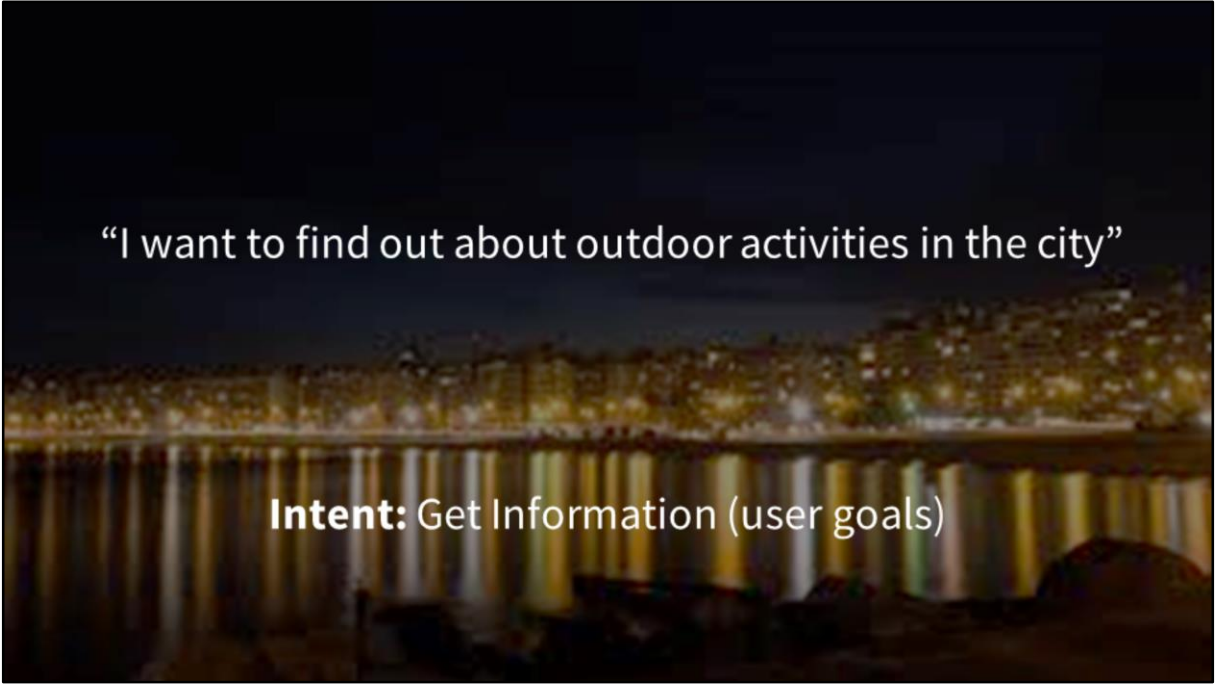
```
Chatbot.SendEntityValues(&Provider,&SDTEntityValues,  
!"UserIdentification",&InstanceName,&messages)
```



El módulo externo se instala cuando creamos un objeto Conversational Flows en la KB.

Vean que se exponen algunos métodos que podemos usar; por ejemplo, SendEntityValues, sirve para subir valores de entidades y sinónimos en el Provider.

How to create a Chatbot using GeneXus



“I want to find out about outdoor activities in the city”

Intent: Get Information (user goals)

Pensemos en los verbos que usarán los usuarios para determinar las intenciones

<https://www.youtube.com/watch?v=wyWgsF9eYc8>



“I want to find out about outdoor activities in the city”

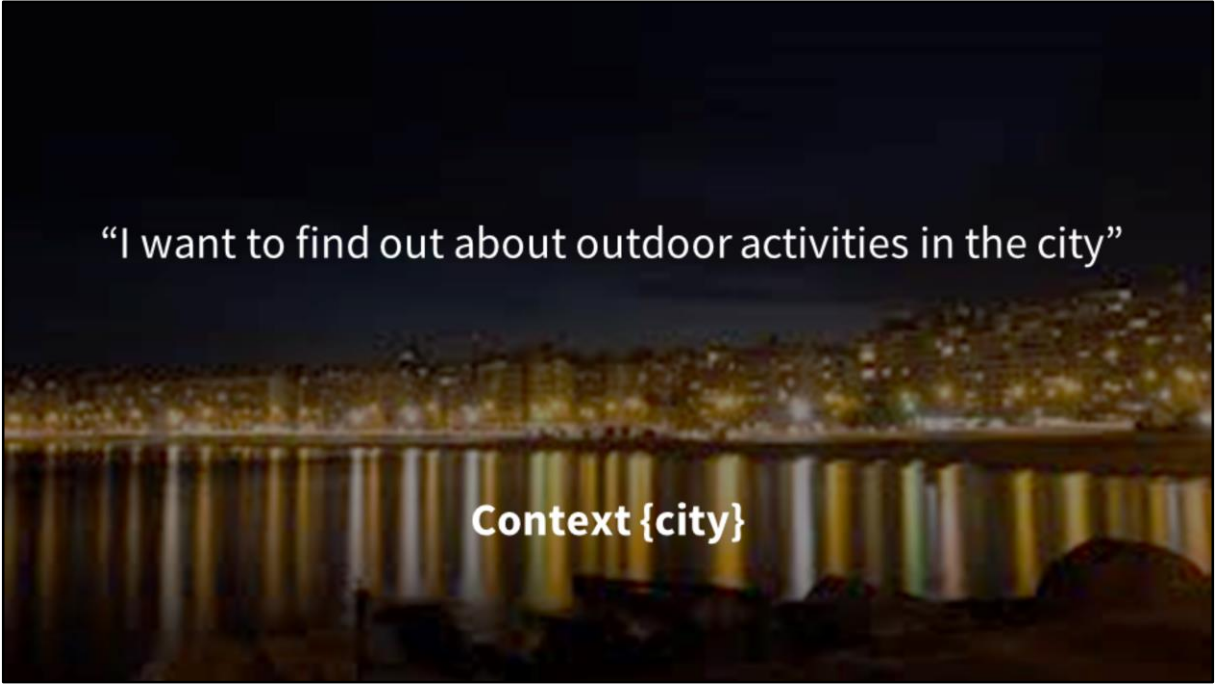
Entity : Activities {artistic, outdoors, cultural}
Values & synonyms

Dada una intención del usuario, para responder mejor a esa intención, se puede inferir información adicional, llamada entidades. Entidades son grupos de valores (cada valor tiene sus sinónimos), que se mencionan dentro de la consulta y sirven para dar mejor comprensión a la misma.

Por ejemplo, dada la intención de obtener información de actividades, la entidad será el tipo de actividades, que puede ser artística, al aire libre, o cultural por ejemplo.

Activities_Category es una entidad, en donde un valor puede ser “outdoor”, con sinónimos nature, fresh air.

Las entidades se almacenan en el NLP provider, con sus posibles valores y sinónimos.



“I want to find out about outdoor activities in the city”

Context {city}

Toda conversación tiene un contexto

Sería difícil entendernos, si uds no recordaran de qué estoy hablando, o lo que dije hace un minuto atrás por lo menos, y más difícil sería que yo tuviera que ir repitiendo ese contexto de forma constante.

Lo mismo ocurre con un chatbot, es imprescindible que mantenga en el contexto la información que se intercambia.

Citizen service chatbot

Do a traffic claim

Find out about activities

Get debt refinancing info

Setup an appointment



NLP response

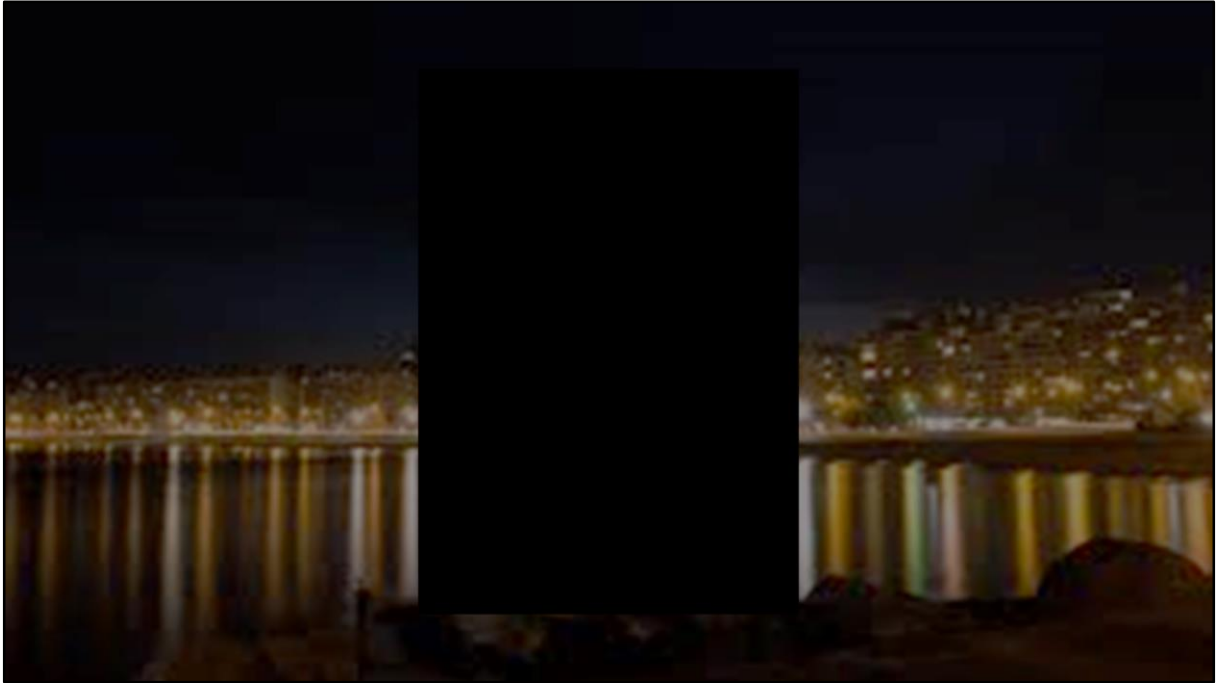
```
{  
  "intents": [  
    {  
      "intent": "Doaclaim",  
      "confidence": 1.0  
    }  
  ],  
  "entities": [],  
  "input": {  
    "text": "do a claim"  
  }  
},
```



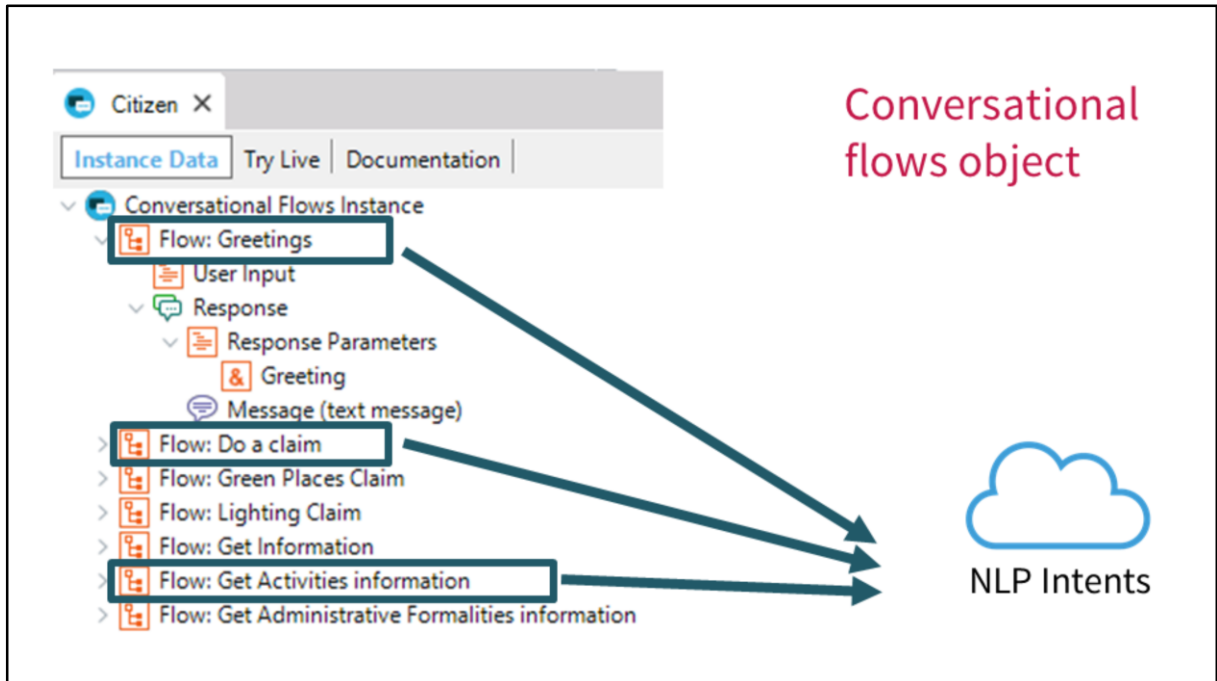
Pensemos en los verbos que usarán los usuarios para determinar las intenciones

Language Understanding

Aquí participa el NLU, que es quien interpreta el lenguaje en el texto ingresado por el usuario, y según esa “interpretación” determina con cierto nivel de confianza los intents asociados a ese contenido.



Mostrar video (solicitud de info de act culturales, y luego agendar una actividad). Que salude con el nombre de la persona. Aclarar que eso es porque fue solicitado en otro flujo, y lo mantiene en el contexto.



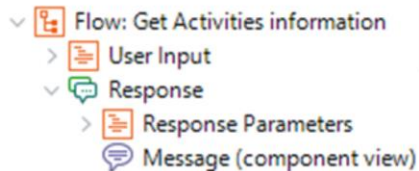
Veremos en GeneXus.

El siguiente es el objeto Conversational Flows, que representa un chatbot. Tiene una vista de árbol, en donde cada nodo Flow, representa una intención: vemos que tenemos varias, como “Hacer un reclamo”, etc.

Tener en cuenta que este objeto está dentro de mi base de conocimiento por lo cual tengo acceso a la lógica de negocios de mi aplicación, y cualquier respuesta puede recuperar información de mis tablas, o incluso persistir datos.

Las que vimos en el video, son “Get activities information” y “Schedule guided visit”.

Trigger messages



NLP Messages

Properties	
Filter	
flow: Flow: Get Activities information	
Conversational Object	CulturalActivitiesNews
Name	Get Activities information
Trigger Messages	Social events;info social events;activities

Know about activities

Find out about leisure

Social events information

--

En el cada flow, modelamos un flujo de la conversación. Allí tenemos el input que se le va a solicitar al usuario, el objeto GeneXus que se va a ejecutar para resolver ese flujo, y la forma en que se va mostrar la respuesta.

Cómo identifica el NLP provider el intent? A través del Trigger Message se puede dar pautas al NLP para que pueda identificarse el intent. Obviamente, no seremos exhaustivos en ellas porque no es necesario, ese es el trabajo del NLP.

Ya logramos entrenar para que se identifique la intención, de obtener información acerca de actividades.

Queda pedirle información al usuario, acerca de qué tipo de actividad le interesa.

User input

"I want to find out about activities in the city"



I'm very glad to help you! What type of activities are you interested in? It can be Culture, Art, or Nature.

NLP Entity



Activities

{artistic (art, music, theatre),
outdoors (nature, fresh air),
culture (cultural, museum)}

SendEntityValues(in:&Provider,
in:&EntityValues, in:&Entity,
in:&ChatbotInstance,
out:&Messages);

Cuando el usuario nos pregunta acerca de actividades en la ciudad, el bot contesta qué tipo de actividades le interesan.

Las categorías de actividades son una entidad en el proveedor, con sus valores y sinónimos.

El chatbot generator crea estas entidades, y tenemos una API para cargar sus valores.

User input

“Outdoors activities! Thanks!”

Flow: Get Activities information

User Input

CulturalActivitiesCategory

Response

“outdoors”

Properties

variable: CulturalActivitiesCategory

Name	CulturalActivitiesCategory
Description	CulturalActivitiesCategory
Data Type	CulturalActivitiesCategory
Match With Entity	True
Entity	Activities
Ask again	True
Try Limit	2
Collection	False
Ask Messages	I'm very glad to help you! What type of activities are you interested in? It can be Culture, Art, or Natu...
On Error Messages	&UserName I don't know about &GXUserInput yet, sorry. Please, enter Culture, Art, or Nature,;
Clean Context Value	True

Para solicitarle al usuario que indique qué tipo de actividad le interesa, agregamos un user input al Flow.

Para eso, le presentaremos las opciones que el chatbot sabe hablar (obviamente no es un mundo infinito de opciones). Esta info se la solicitamos a través del nodo user input (vean la propiedad Ask Messages).

Este User Input tiene la propiedad Match With Entity = TRUE, lo cual significa que cuando el usuario ingrese su consulta necesariamente se va a inferir de ella en forma automática un valor de la entidad.

En la variable &ActivitiesCategory nos vamos a quedar con el valor de la entidad.

El dominio de lo que se sabe hablar, se define en el provider, es decir, el conocimiento esta allí.

Notemos aquí que si bien se podría haber usado botones para la selección, - lo cual es válido, usar un modelo en donde predomina el texto, está basado en toda la potencia de NLP, y es trasladable a Voz (voz a texto / texto a voz). Eso va a gusto del consumidor.

Veamos esta potencia, en donde el usuario puede referirse a cualquier sinónimo de la entidad, y ésta será reconocida.

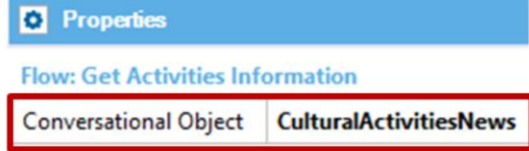
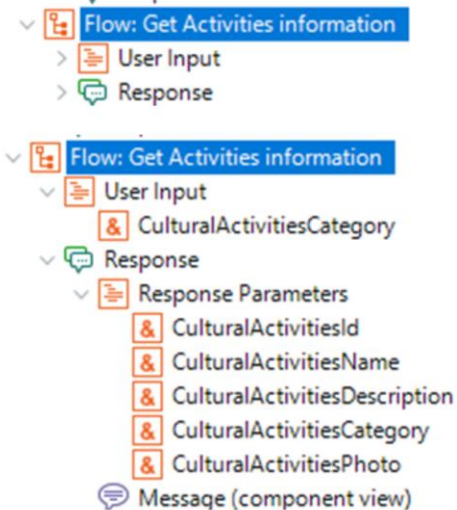
Integrado

Notemos un valor importantísimo a la hora de reconocer las entidades, que es el Provider quien dará el OK o no a lo ingresado por el usuario. No tiene sentido permitir solicitar información de

un tópico que el sistema no cubre, por ejemplo, de la cotización de la moneda. Para que se valide en el rango de los posibles valores de la entidad (o sinónimos) se usa la propiedad Match with entity

Tenemos una API para usar en GeneXus que nos permite dar de alta los valores de estas entidades, y sus sinónimos.

Response



```
1 parm(in:&CulturalActivitiesCategory);
2
3 {
4     CulturalActivitiesNew
5     where CulturalActivitiesCategory = &CulturalActivitiesCategory
6     {
7         CulturalActivitiesId = CulturalActivitiesId
8         CulturalActivitiesName = CulturalActivitiesName
9         CulturalActivitiesDescription = CulturalActivitiesDescription
10        CulturalActivitiesCategory = CulturalActivitiesCategory
11        CulturalActivitiesPhoto = CulturalActivitiesPhoto
12    }
```

¿Cómo se resuelve el intent?

En este caso se usan dos objetos para resolver el intent.

Uno de ellos resuelve la consulta en la BD, y el otro muestra los resultados.

El objeto que calcula los resultados está dado en la propiedad Conversational object en este caso.

Se trata de un data provider que recibe como entrada el input ingresado por el usuario (el tipo de actividad) y devuelve una lista de actividades.

Bien, y ¿cómo mostramos este resultado?

Response

- Flow: Get Activities information
 - User Input
 - CulturalActivitiesCategory
 - Response
 - Response Parameters
 - CulturalActivitiesId
 - CulturalActivitiesName
 - CulturalActivitiesDescription
 - CulturalActivitiesCategory
 - CulturalActivitiesPhoto
 - Message (component view)

Properties

messages: Message (component view)	
Condition	
Action	component view
Messages	Activities to do in our City
Component view properties for Smart Devices	
Show Response As	Component
Component references	
SD Component	(none)
Generated SD Component	CitizenServiceChatbot.CulturalActivitiesNewsComponentSD
Web Component	(none)
Generated Web Component	

```

parm(in:&CulturalActivitiesCategory);

Event Start
    &CulturalActivitiesNew = CulturalActivitiesNews(&CulturalActivitiesCategory)
Endevent
    
```

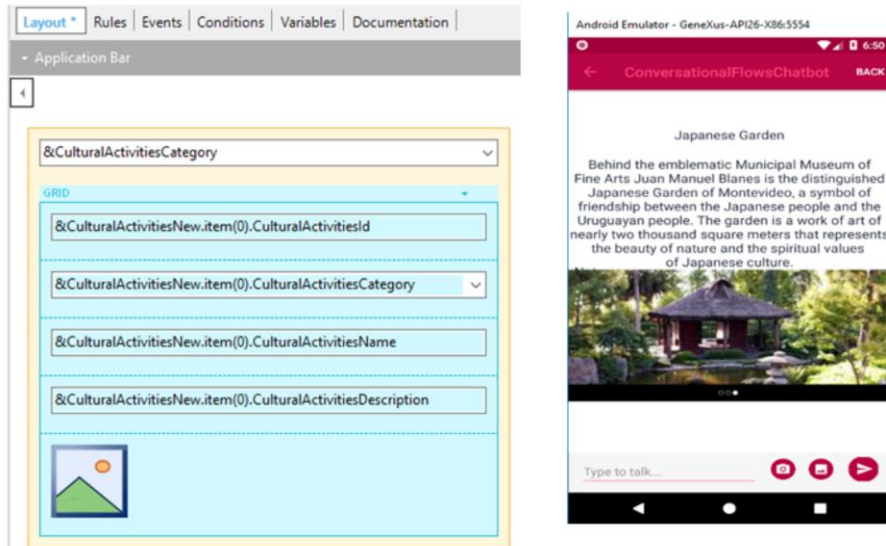
Dentro del nodo response, el nodo message nos permite indicar cómo será nuestra respuesta, La respuesta puede ser texto, o rica en contenido (puede ser un webpanel de nuestra KB, o un SD panel) incluyendo videos, audio, imágenes etc.

Vean que tenemos la propiedad Show Response as con el valor component.

El component que será ejecutado está indicado en la propiedad Generated SD Component, Este objeto es generado automáticamente por el pattern. Este objeto llama a mi Data provider (conversational object) y muestra los resultados en un horizontal grid.

Este SD panel es llamado automáticamente por un handler generado por el chatbot generator, para resolver este intent.

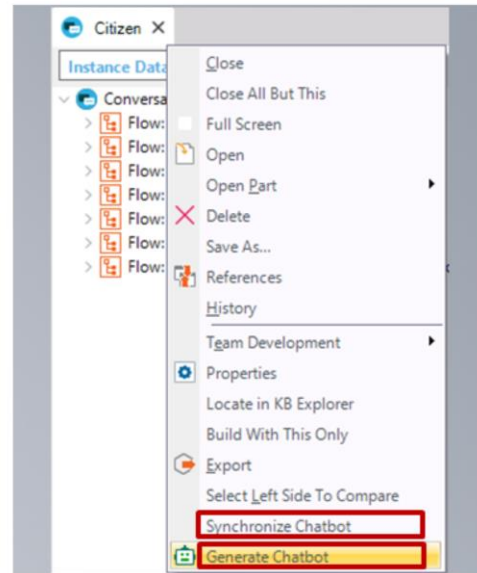
Response



A partir de los response parameters, que se infieren automáticamente del conversational object (recuerden que es un dataprovider que devuelve la lista de actividades culturales), se arma el objeto SD predeterminado para mostrar la respuesta.

Impacting model changes

Synchronize Chatbot
Generate Chatbot



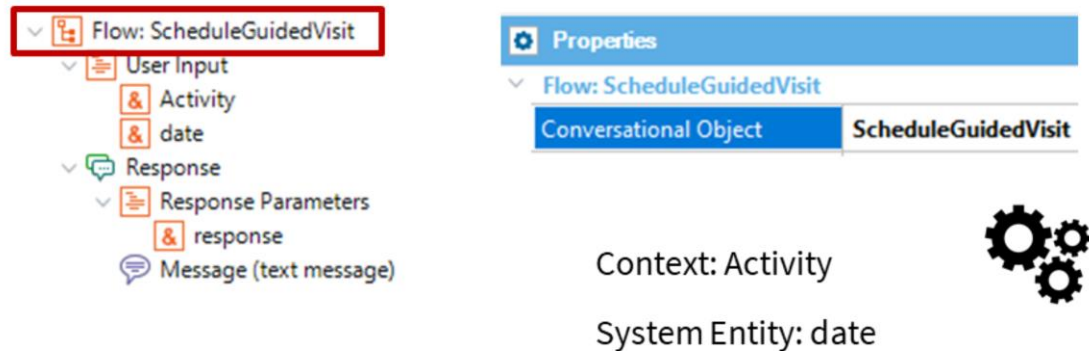
¿Para generar estos cambios qué debemos hacer?

El generador de chatbots se comporta como un pattern (Conversational Flows).

El objeto conversational flows se sincroniza contra el NLP provider, y también a partir de él se generan objetos en la KB para darle forma a la solución.

Haciendo un build se generan los objetos en la KB que son llamados desde el server para que el chatbot responda.

Another example: scheduling



Lo que vimos hasta ahora era una consulta

También se puede persistir datos usando nuestro chatbot, que es el ejemplo de agendar una visita guiada.

En este ejemplo, tenemos el flujo que resuelve el intent, en donde se solicita el tipo de actividad (si no viene dada en el contexto se solicita, de lo contrario no),

Si en otro flujo se le solicitó la actividad, esta fue grabada en el contexto, porque explícitamente le indicamos al modelo que así lo hiciera.

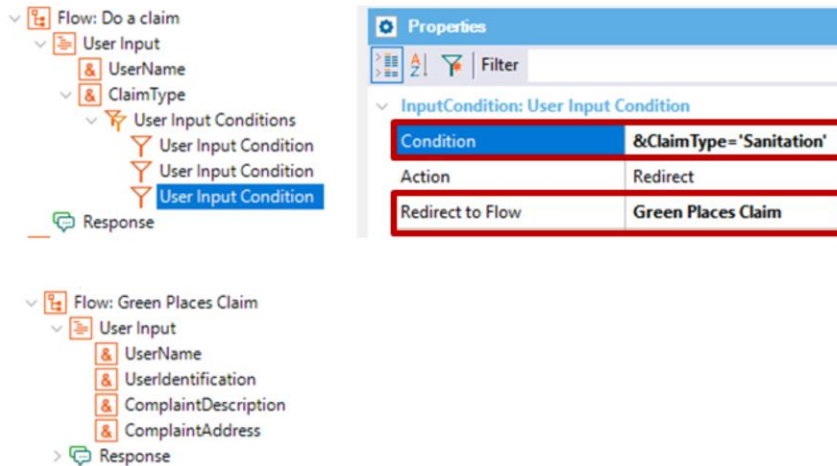
Además se pide al user la fecha en la cual quiere agendar.

Lo interesante acá es que la fecha se convierte en una entidad del sistema en Watson, y se interpreta como tal, por eso el usuario puede ingresar expresiones como tomorrow 5 pm, o today.

Este flujo se resuelve con la ejecución de un conversational object que es un procedure que usa el calendar API para grabar la información en la agenda del dispositivo.

El mensaje de salida en este caso es texto.

Redirect to another flow



Es importante tener en cuenta que los flujos a veces no tienen una solución en sí mismos, sino que, en función de la información que se le pide al usuario, es necesario redirigir a otro flujo.

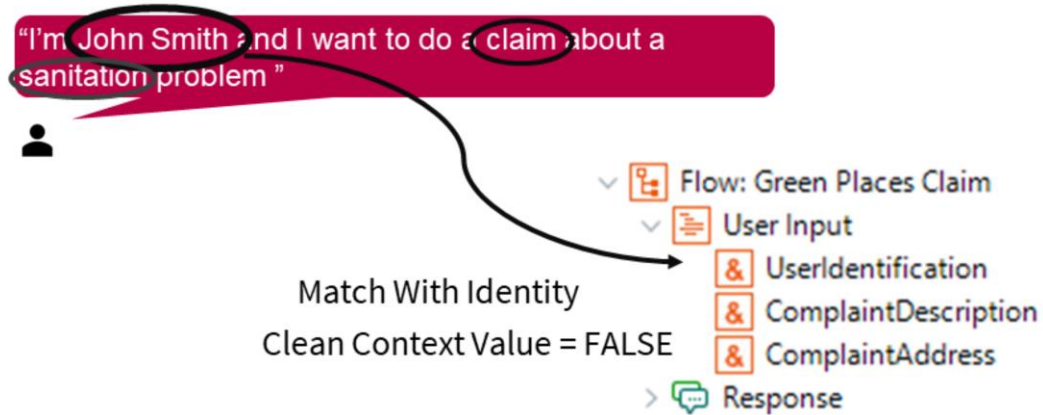
Por ejemplo en este caso, en donde se hace un reclamo, dependiendo del tipo de reclamo, se redirige al flujo que corresponde. Lo que se hace es agregar una User Input condition bajo el user input, que en este ejemplo vean que pregunta si el tipo de reclamo es de saneamiento.

Cuando lo es, la acción es dirigir al flujo Green Places Claim, donde se pedirá la información pertinente a ese reclamo.

Por supuesto que se puede ingresar al flujo de Do a claim, tanto como se puede ingresar directamente al flujo de Green places claim, cuando el NLP detecta que el reclamo es específico de esa área.

Si el usuario hace referencia a que se trata de un reclamo de saneamiento, entra directamente al 2º flujo.

Entity inference from the query



Tengamos en cuenta que cuando el usuario ingresa el valor de una entidad, sea en la consulta misma o en un user input, no se le vuelve a preguntar, y eso convierte al chatbot en algo más parecido a lo que es una conversación humana.

Si el usuario dice esto, se entiende que quiere hacer un reclamo de saneamiento (eso es porque ingresó palabras que en el NLP ayudan a indentificar ese intent)

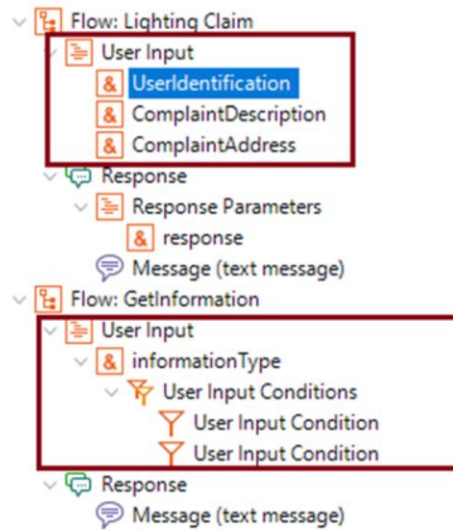
Pero además, incluyó en su consulta su identifcación, que mapea con una entidad, por lo cual, ésta no se le va a solicitar nuevamente.

Incluso, se puede guardar en el contexto.

Context

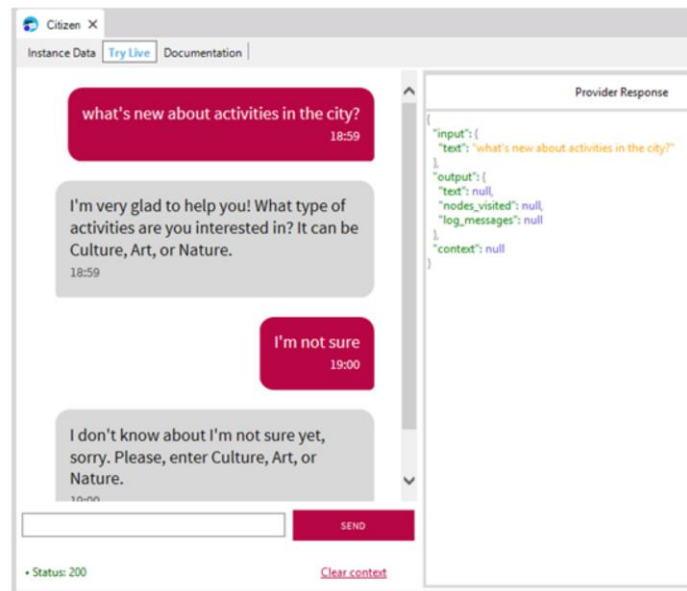
- &GXUserInput
- User Inputs

Clean Context Value = TRUE/FALSE



El contexto se puede usar en las propiedades AskMessages, on error messages, input conditions.

Try live



En la KB, tenemos un objeto Conversational Flows, que representa el chatbot.

Cuando se salva el objeto o se sincroniza, se arma un archivo JSON que se usa para actualizar en el NLP provider el diálogo, y el flujo de conversación que hemos definido.

A su vez, dentro de la KB se generan objetos que contienen la lógica necesaria para comunicarse con el chatbot.

La complejidad de interacción con los componentes, está resuelta, y solo tenemos que modelar nuestro chatbot.

More about SD

GeneXus™ 16

Veamos ahora algunas novedades dentro de Smart Devices

Control Value Changed

Web and SD

Service

Citizen Web — Formality Reservat...

Make a Reservation

User Identification

Formality Description

Apply for debt refinancing

Formality Requirements

Present the current account number or identification, according to the tax in question.

Formality Price

\$ 200

Formality Address

18 de Julio Av 1360, Montevideo Department

From

01:00 PM

To

05:00 PM

Formality Date Time

/ / 12:00 AM

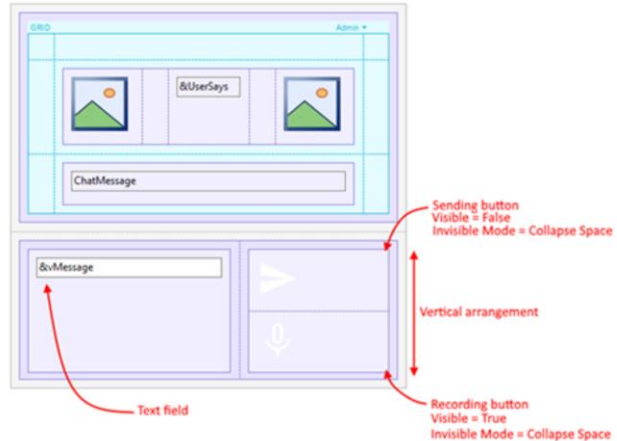
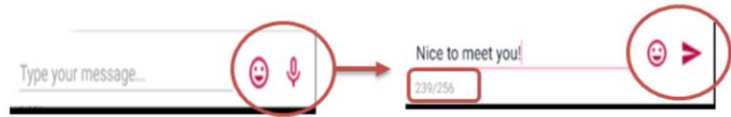
25

Confirm

Control Value Changed. Es un Evento nos permite ejecutar algún código justo en el momento en que el usuario terminó de ingresar un valor.
Aplica tanto para Web y SD.

ControlValue Changing

```
Event &vMessage.ControlValueChanging(&vMessageNew)  
Composite  
    &MessageLen = &vMessageNew.Length()  
    SendButton.Visible = &MessageLen > 0  
    RecButton.Visible = &MessageLen = 0  
EndComposite  
Endevent
```



El evento control value changing nos permite validar si el usuario está cambiando el valor de un campo editable y ejecutar un código.

Por ejemplo cuando el usuario no ha escrito aún tiene habilitado el botón de grabar un audio y cuando comienza a redactar el texto aparece el botón de enviar.

También lo podemos utilizar para llevar el conteo de las letras a enviar.

Audio Recorder

Structure	Type
AudioRecorder	
Properties	
IsRecording	Boolean
Methods	
Start	Boolean
Stop	Url, GeneXus
Events	

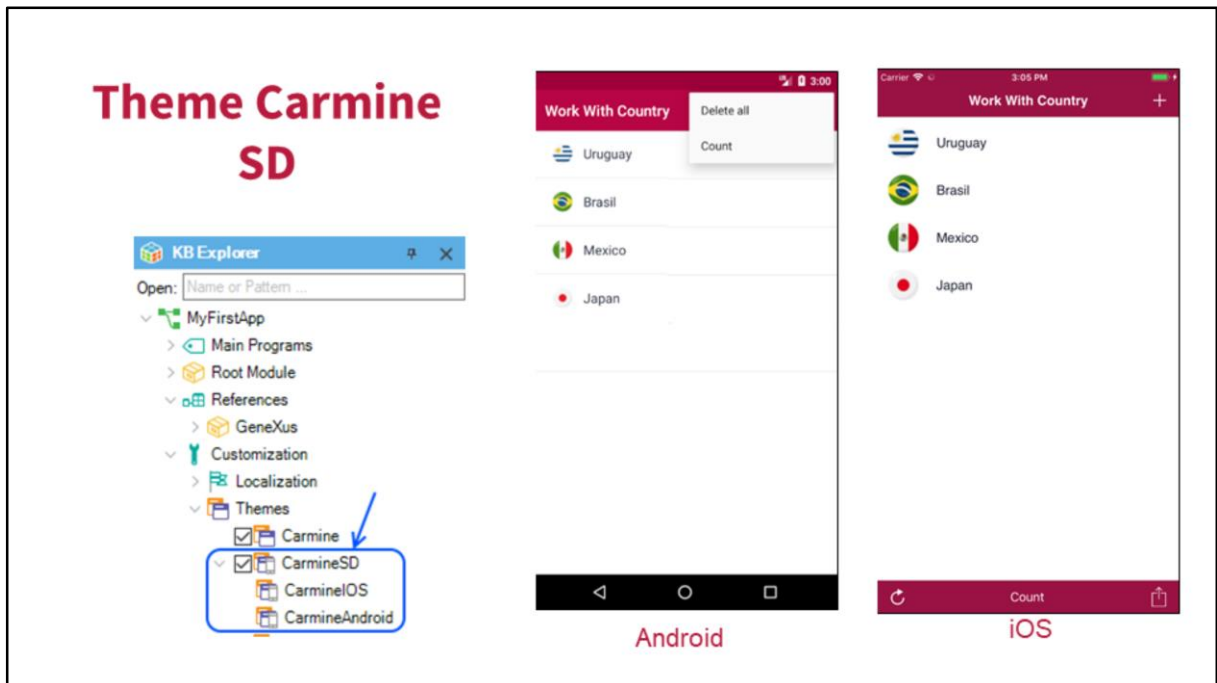
```

Event 'StartRecording'
  &HasSuccess = AudioRecorder.Start()
EndEvent

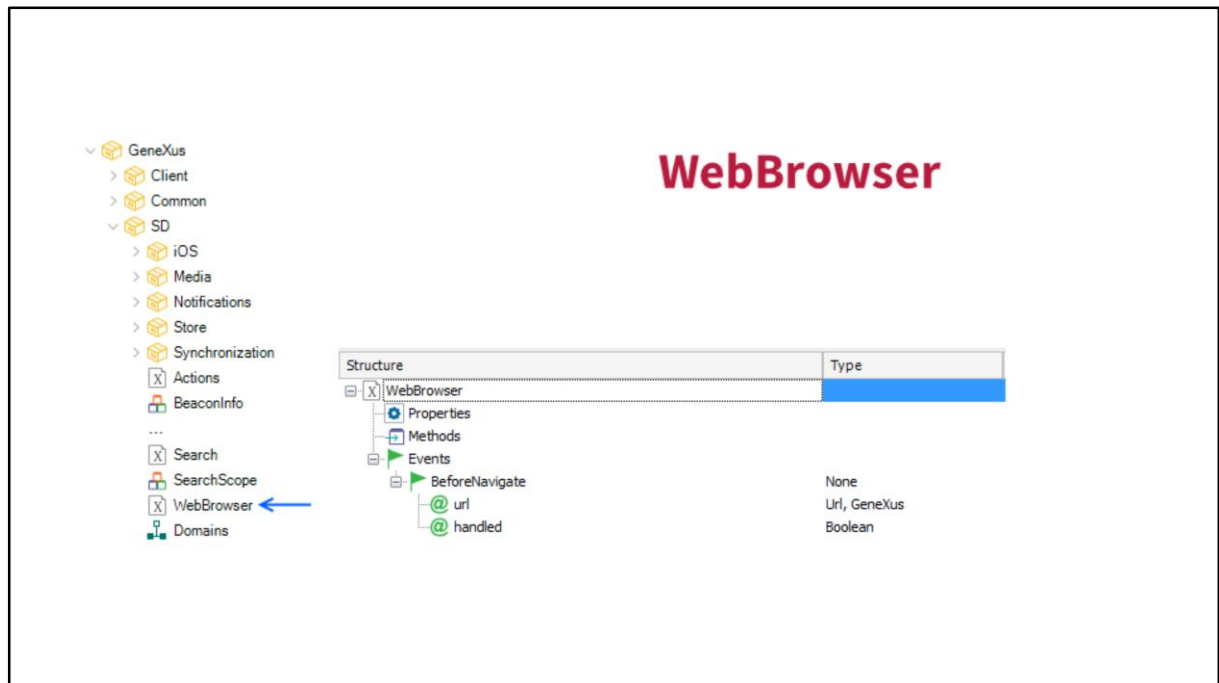
Event 'StopRecording'
  Composite
    &IsRecording = AudioRecorder.IsRecording
    If &IsRecording
      &FilePath = AudioRecorder.Stop()
      &Audio.AudioURI = &FilePath
      SendAudioMessage(&Audio,&Username)
      Refresh
    EndIf
  EndComposite
EndEvent

```

Audio recorder es un external Object que nos permite grabar una pista de audio utilizando sus métodos Start o Stop.



Desde el upgrade 6 de Genexus 15 contamos con el tema Carmine el cual nos permite tener un look and feel moderno y sofisticado.



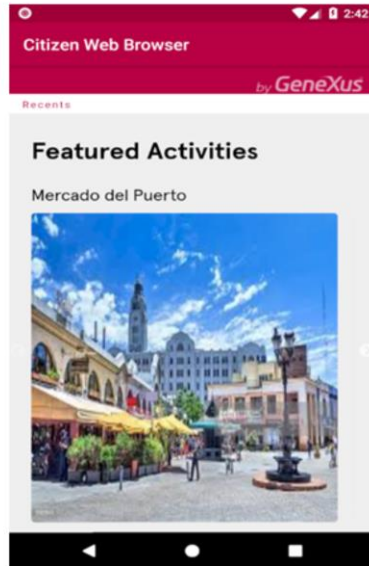
Web Browser es un External Object que nos permite manejar los eventos disparados de una url embebida en una aplicación Smart Devices.

Web app Embedded in SD app

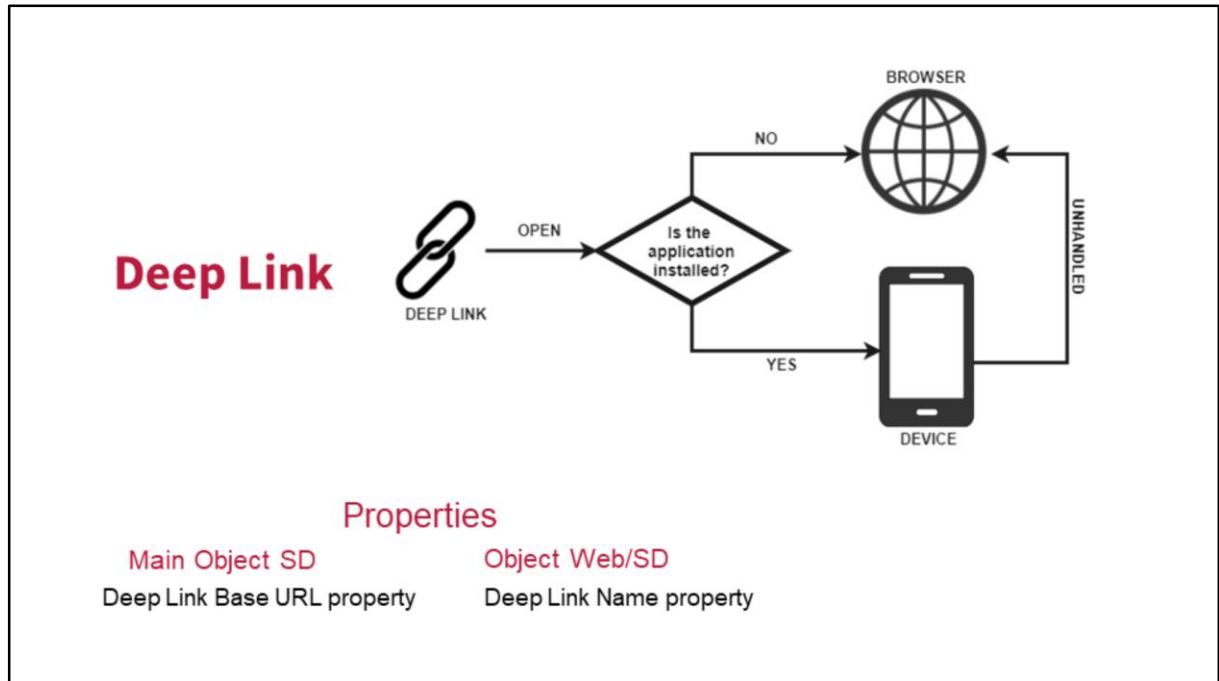
Sample: WebBrowser

Before Navigate to URL or Web app Embedded

```
Event GeneXus.SD.WebBrowser.BeforeNavigate(&Url, &Handled)
  composite
    if &Url = //something
      //<your_code>
      &Handled = true
      return
    endif
  endcomposite
EndEvent
```



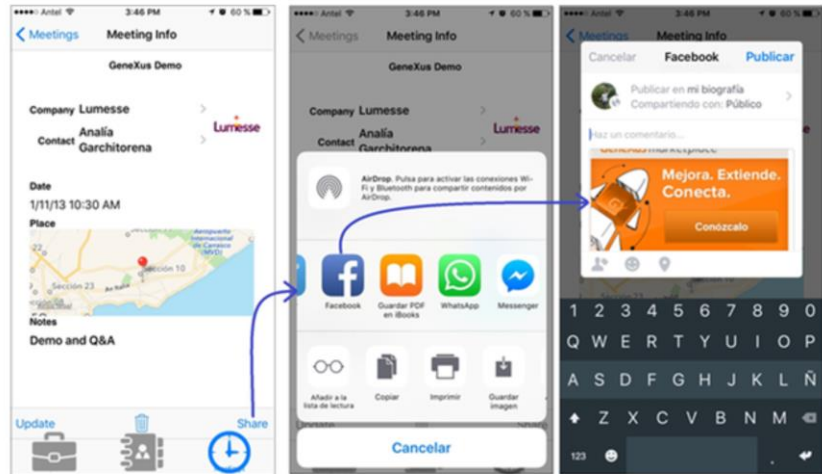
Para manejar la navegación utilizamos el Evento Before Navigate , el cual se dispara antes de navegar una url o aplicación web embebida.
Con esto controlamos si el usuario está entrando a una url externa a nuestra aplicación.



Deep Link Nos permite elegir cómo abrir una URL, si tenemos la app SD instalada lo hará ahí, de lo contrario abrirá su objeto correspondiente en Navegador Web.

Share

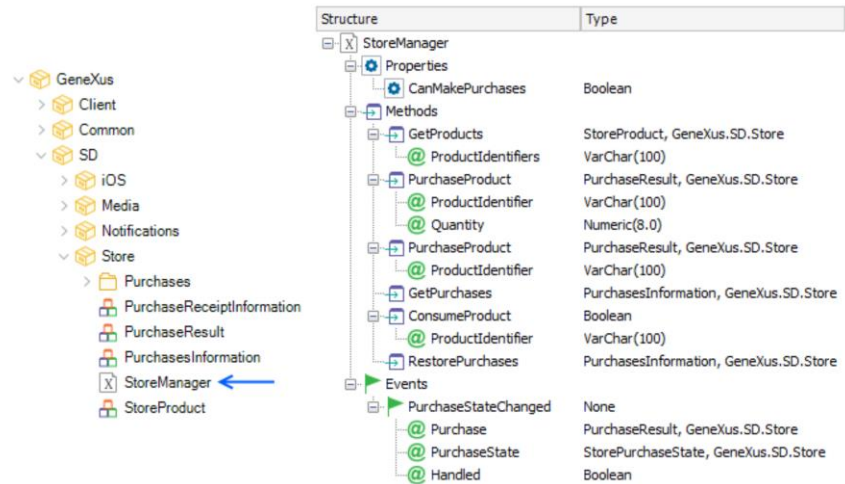
Structure	Type
Share	
Properties	
Methods	
ShareText	None
@ text	VarChar(200)
@ url	Url, GeneXus
@ title	VarChar(200)
ShareImage	None
@ image	Image
@ text	VarChar(200)
@ url	Url, GeneXus
@ title	VarChar(200)
Events	



```
Event 'Share'  
  Share.ShareImage(CompanyLogo, MeetingTitle,!"http://www.genexus.com",MeetingNote)  
EndEvent
```

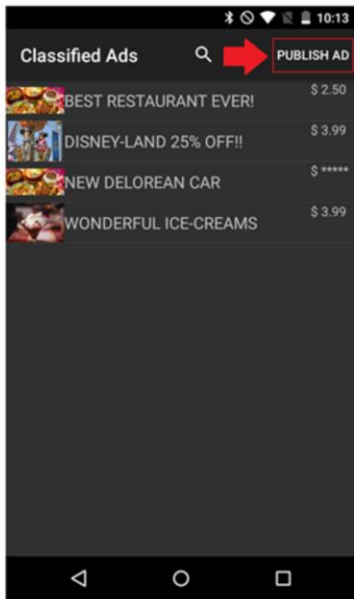
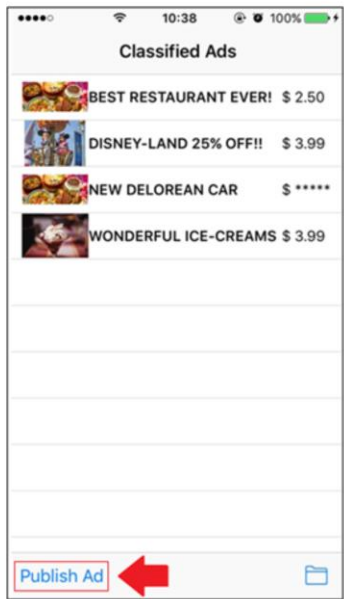
Share es un external Object que nos permite compartir Texto o Imagen con alguna aplicación externa que lo permita.

Store Manager



Store Manager es un External Object que permite administrar las aplicaciones de compras.

Sample: Store Manager



```
Event 'Publish Ad'  
Composite  
    &ProductId = 'publish_my_ad_id'  
    &PurchaseResult =  
        StoreManager.PurchaseProduct(&ProductId)  
    //Code  
EndComposite  
Endevent
```


En este ejemplo apreciamos que al momento de dar tap al botón publish ad se despliega la opción de completar la compra.

Client Information

Main Object SD property
Include Network Id in Client Information = TRUE

Structure	Type
ClientInformation	
Properties	
Id	VarChar(128)
OSName	VarChar(40)
OSVersion	VarChar(40)
NetworkID	VarChar(128)
Language	Character(20)
DeviceType	SmartDeviceType, GeneXus
PlatformName	VarChar(128)
AppVersionCode	VarChar(40)
AppVersionName	VarChar(40)
ApplicationId	VarChar(128)
Methods	
Events	

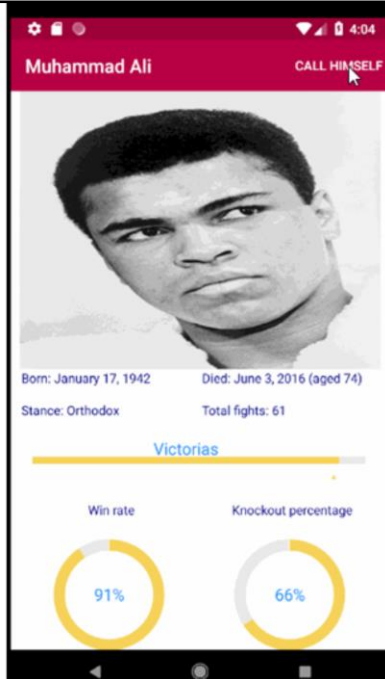
IMEI
MEID
ESN



Only Android

Client Information es un External Object que sirve para recuperar la información del device, una de las características importantes es su propiedad Network ID, que en conjunción con la propiedad Include Network ID de nuestro objeto main, nos permite recuperar el IMEI del dispositivo.

SDGauge



SDGauge es un control que nos permite desplegar información en forma de rangos, lineal o circular.

Ahora en esta versión el control cuenta con animación para mejorar la experiencia del usuario.

HttpClient

```
Event 'GetInfo'  
composite  
  &httpClient.Host = //your host  
  &httpClient.Secure = //Secure  
  &httpClient.Port = //Port  
  &httpClient.BaseUrl = //Base Url  
  &httpClient.Execute("GET", "<something>")  
endcomposite  
Endevent
```



A partir de la versión 15 El tipo de dato HttpClient se puede utilizar desde aplicaciones smart devices y a partir del upgraded 12 se permite el manejo desde eventos del lado del cliente.

Share session to Webview property

SD object property

✓ Security

Share session to Webview

True

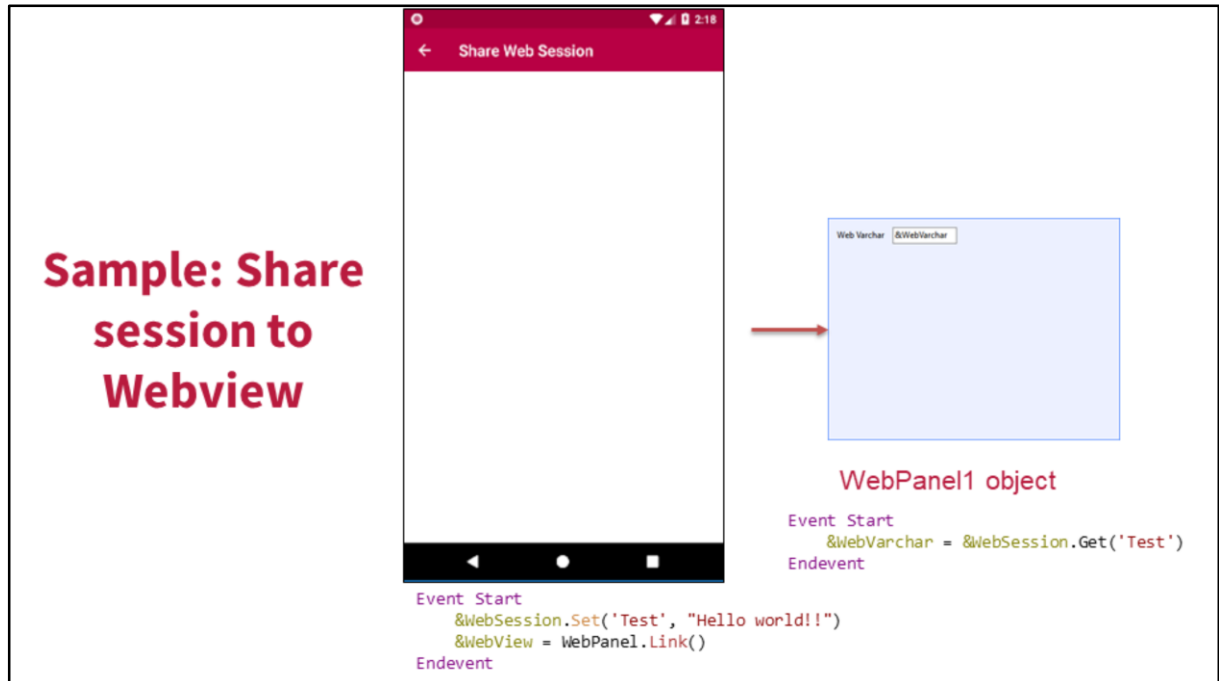
Mobile app



Web Part in Webview



Share sesión to webview property permite compartir session entre la aplicacion móvil y su parte web

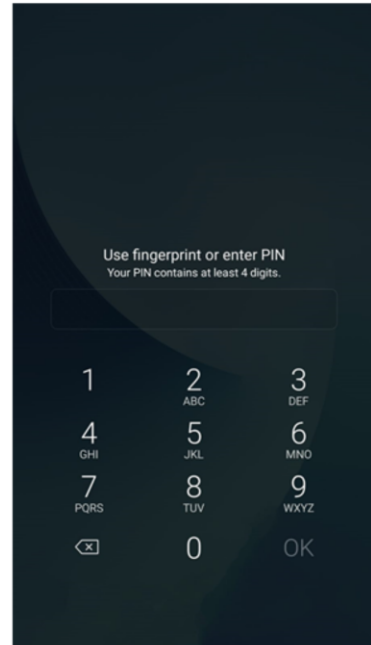
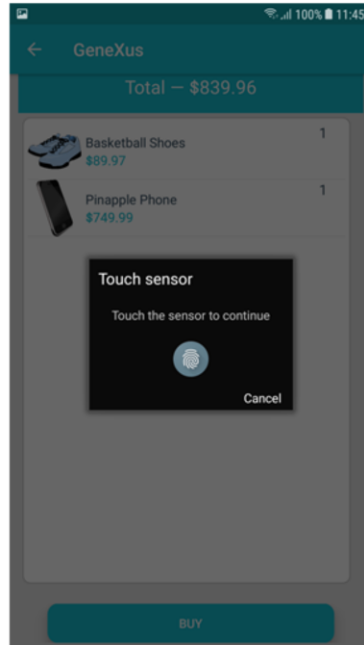


Por ejemplo, desde el objeto SD seteamos la web session

En un web panel obtenemos el valor de esa web session y abrimos el web panel en un web view.

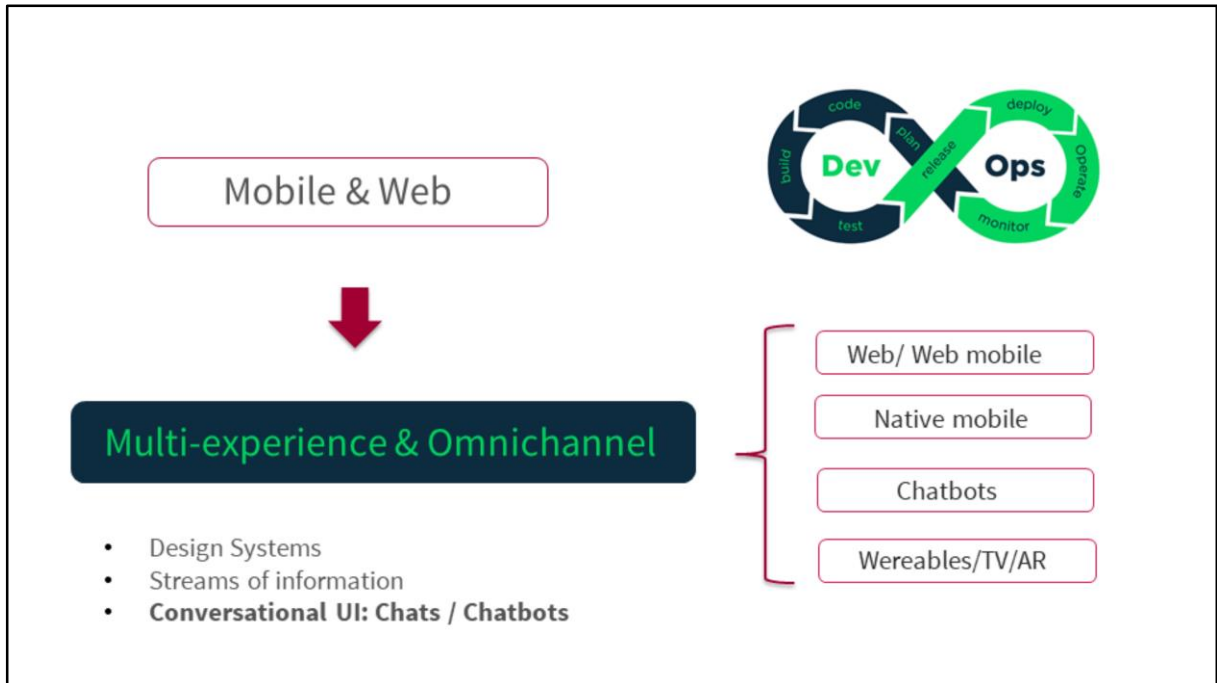
Device Authentication

Structure	Documentation	
Structure	Type	Is Collection
DeviceAuthentication		
Properties		
BiometricDescription	VarChar(40)	<input type="checkbox"/>
AllowableReuseDuration	Numeric(8,0)	<input type="checkbox"/>
Methods		
IsAvailable	Boolean	<input type="checkbox"/>
Authenticate	Boolean	<input type="checkbox"/>
method	DeviceAuthenticationPolicy, GeneXus...	<input type="checkbox"/>
method	DeviceAuthenticationPolicy, GeneXus...	<input type="checkbox"/>
title	VarChar(40)	<input type="checkbox"/>
usageDescription	VarChar(40)	<input type="checkbox"/>
Events		

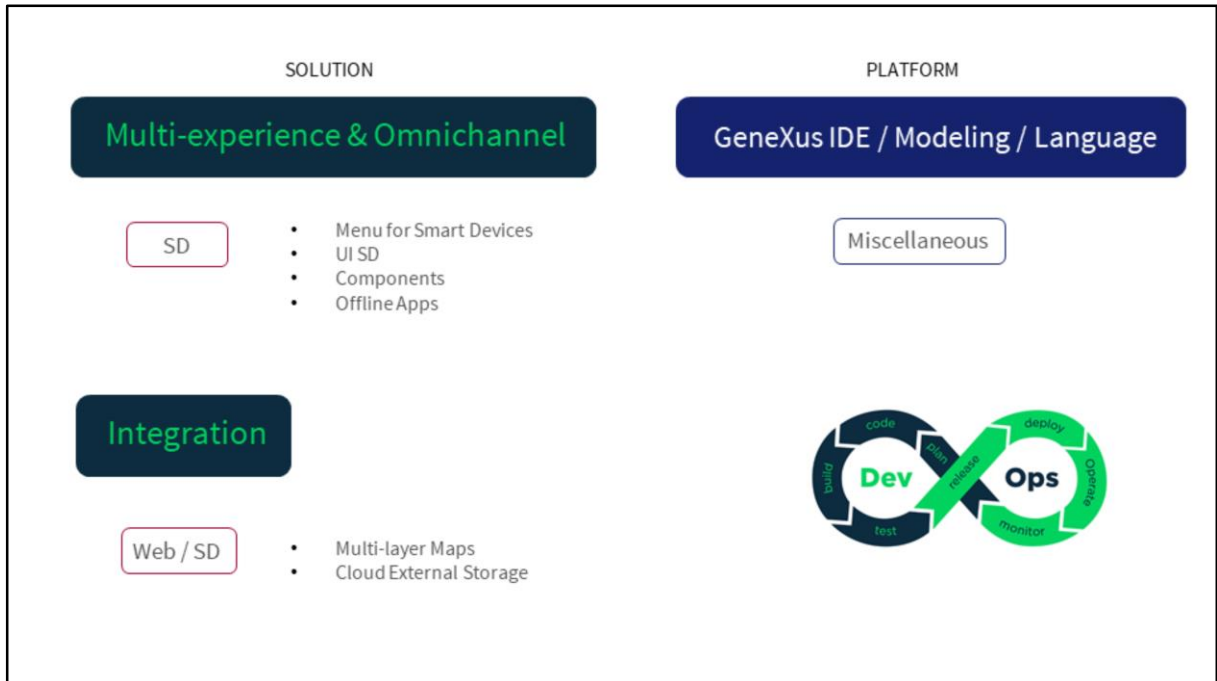


Device Authentication Permite al desarrollador tener acceso a los métodos de autenticación local del dispositivo.

Cada plataforma maneja los propios.



Vimos las facilidades para desarrollar un chat simple y un chatbot, y algunos aspectos sobre SD.



En lo que sigue veremos más funcionalidades que se han ido incorporando a partir de los diferentes upgrades de GeneXus 15 hasta llegar a la versión 16.

Estudiaremos lo nuevo en lo que podemos hacer con los mapas, y también algunas mejoras en lo que hace al almacenamiento externo de archivos multimedia en las nubes soportadas.

Veremos algunos aspectos relacionados con la plataforma GeneXus.

GeneXus™
The power of doing