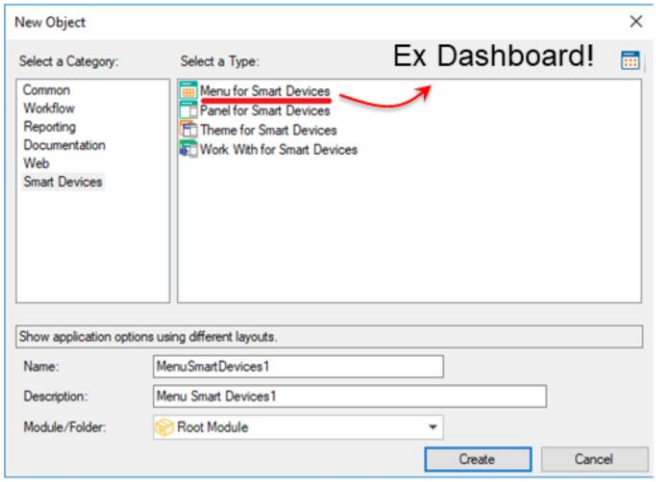
Este material fue elaborado cuando GeneXus 15 se encontraba en el upgrade 8.

Se incluyen aquí la gran mayoría de nuevas features del generador Smart Devices. Por cuestiones de tiempo no serán abordadas todas en la profundidad que en este material se presentan. Las dejamos para que le queden como documentación. No obstante encontrará más y mejor documentación en nuestro community wiki.

Object Name Changed

Menu for Smart Devices



Also, many properties at Smart Devices generator level have been moved to the main object properties level!

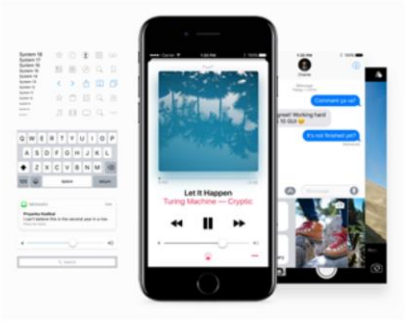
GeneXus

Design guides & GeneXus

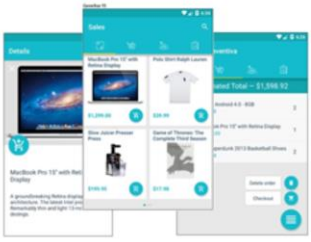
GeneXus

Design Guides

iOS
(Human Interface guidelines)



Material Design for Android (5 +)

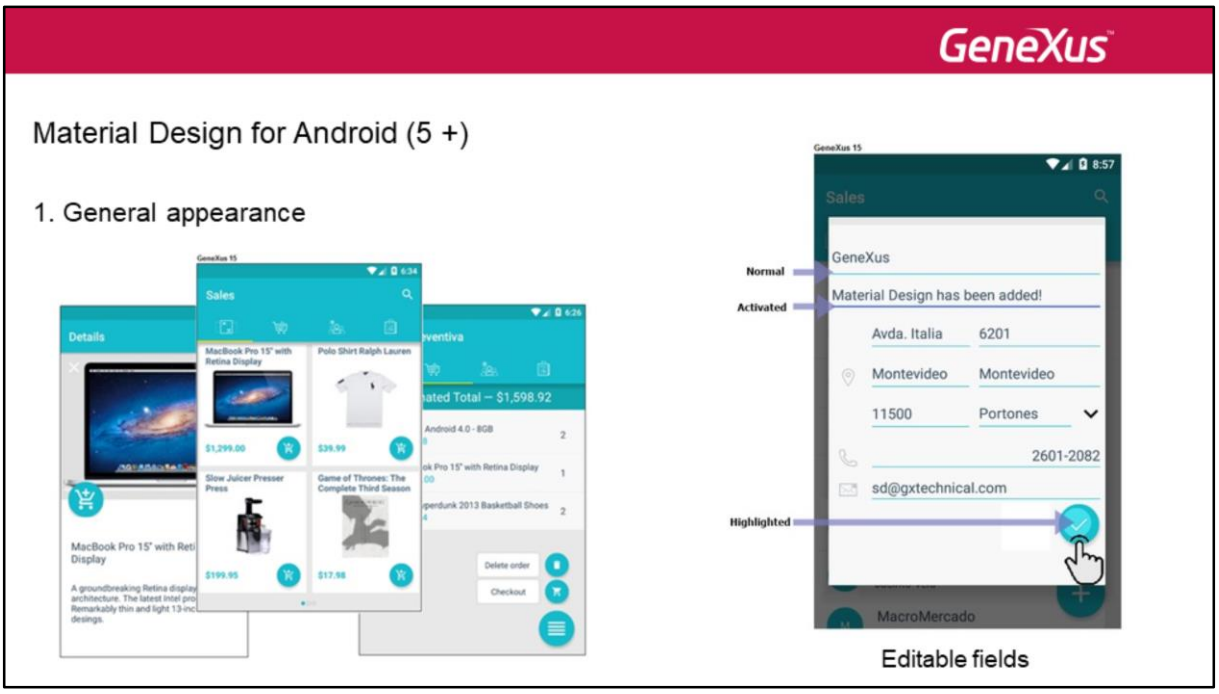


Como sabemos uno de los motivos por los que necesitamos que las aplicaciones sean nativas es que conserven el look & feel de todas las demás apps del dispositivo del usuario.

Cada plataforma define guías de diseño para sus aplicaciones, tanto en lo que hace a la UI como a la UX.

Desde Android 5 se cuenta con las guías conocidas como Material Design, y para iOS son las publicadas en el sitio de Apple (no reciben un nombre especial).

Design Guides: Android

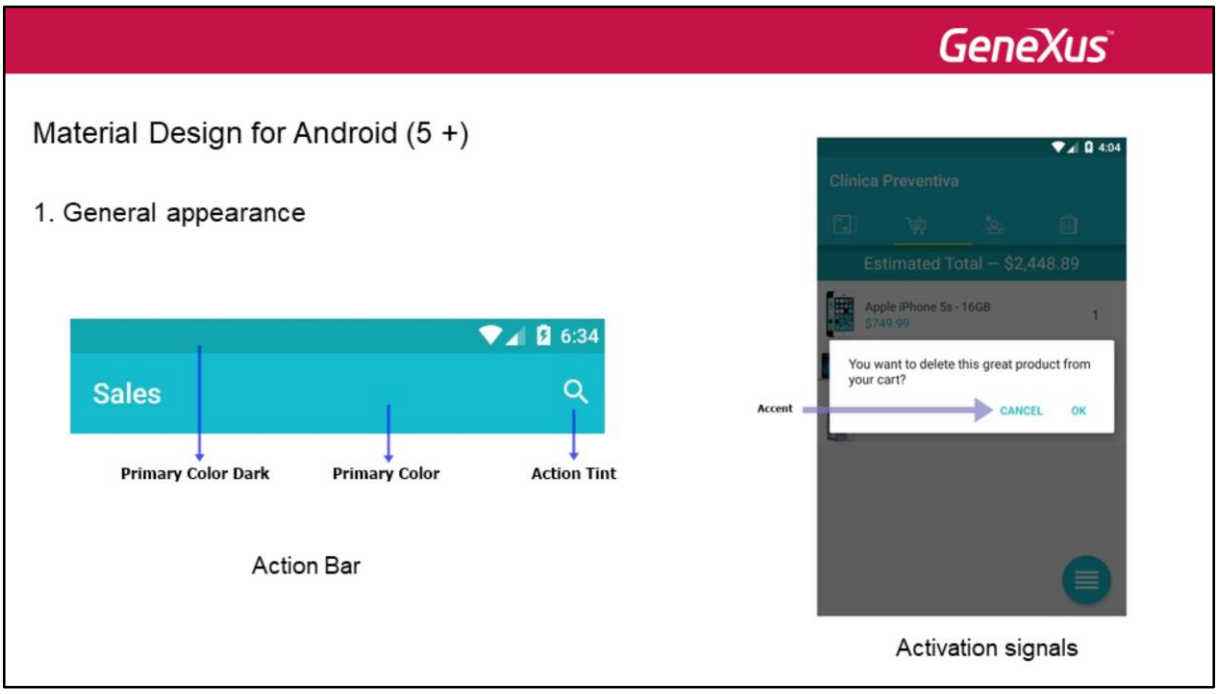


La plataforma Android, desde la version 5 ha incorporado las guías de diseño conocidas como Material Desing, para crear un ecosistema particular y característico. Esto permite a los desarrolladores customizar el look & feel de sus aplicaciones, haciéndolas más user-friendly con una UX optimizada.

Como veremos, estas funcionalidades serán ofrecidas por GeneXus de dos maneras: a través de **propiedades personalizables**, y también serán provistas por **default**.

Aquí mostramos una aplicación de ventas desarrollada con GeneXus 15, que sigue los lineamientos de Material Design, donde la **uniformidad en el uso de los colores para determinados elementos comunes de toda app** se torna fundamental.

Por ejemplo, a la hora de editar campos, vemos que se subrayan con un color uniforme los que no están activos en un momento dado, y con otro el campo con el que se está trabajando. Asimismo al hacer tap sobre un botón, se muestra resaltado siempre con el mismo color, en toda pantalla que haga lo mismo.



La Action Bar es uno de los elementos más importantes de toda aplicación. Su color se convierte en el “primary color” de la aplicación, que distingue a la marca. Las guías del Material Design indican que el color que debe asumir la Status Bar, es decir, la barra de arriba del todo (la que contiene la hora, indicadores de batería, wifi, notificaciones, etc.) debe tener un color 700 tints más oscuro que el primario.

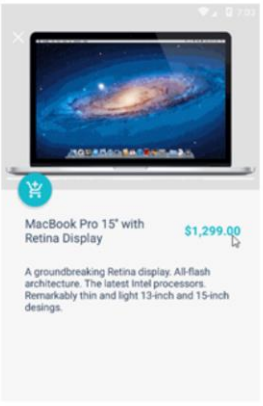
Además, el color de los íconos embebidos en al application bar deben ser más claros que el “primary color”, armónicos con éste, y debe ser siempre el mismo cuando se navega por distintas pantallas de la app.

También se establece de manera uniforme en toda la aplicación el color de los controles que son activados por el usuario (ej: tap sobre un botón o un texto para confirmar en una pantalla modal, radio buttons, checkboxes, datepicker, etc.).

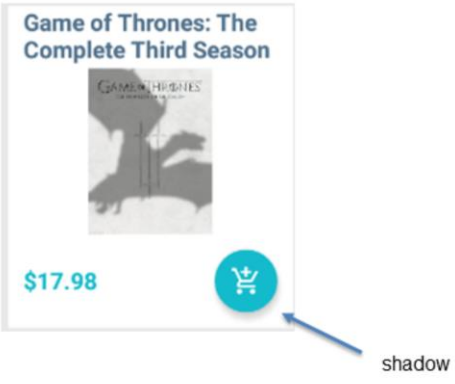
GeneXus

Material Design for Android (5 +)

2. Status Bar color



3. Elevation



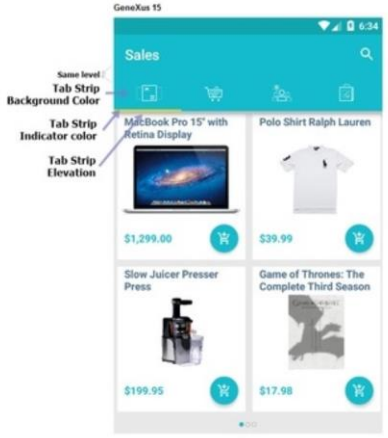
2. Status Bar color: la status bar podría ser escondida bajo ciertas condiciones, por ejemplo, cuando el usuario desliza hacia arriba la pantalla para ver lo que hay más abajo.

3. Una sombra bajo el control lo destaca, indicándole al usuario que es un control seleccionable. Por eso se le dice “elevation” a la propiedad que permite configurar este efecto de altura.

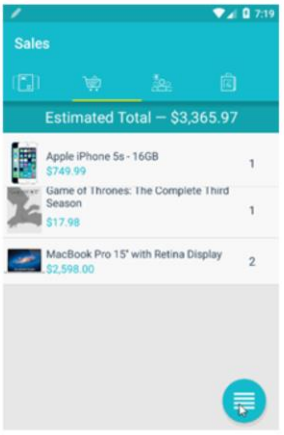
GeneXus

Material Design for Android (5 +)

4. Tab strip



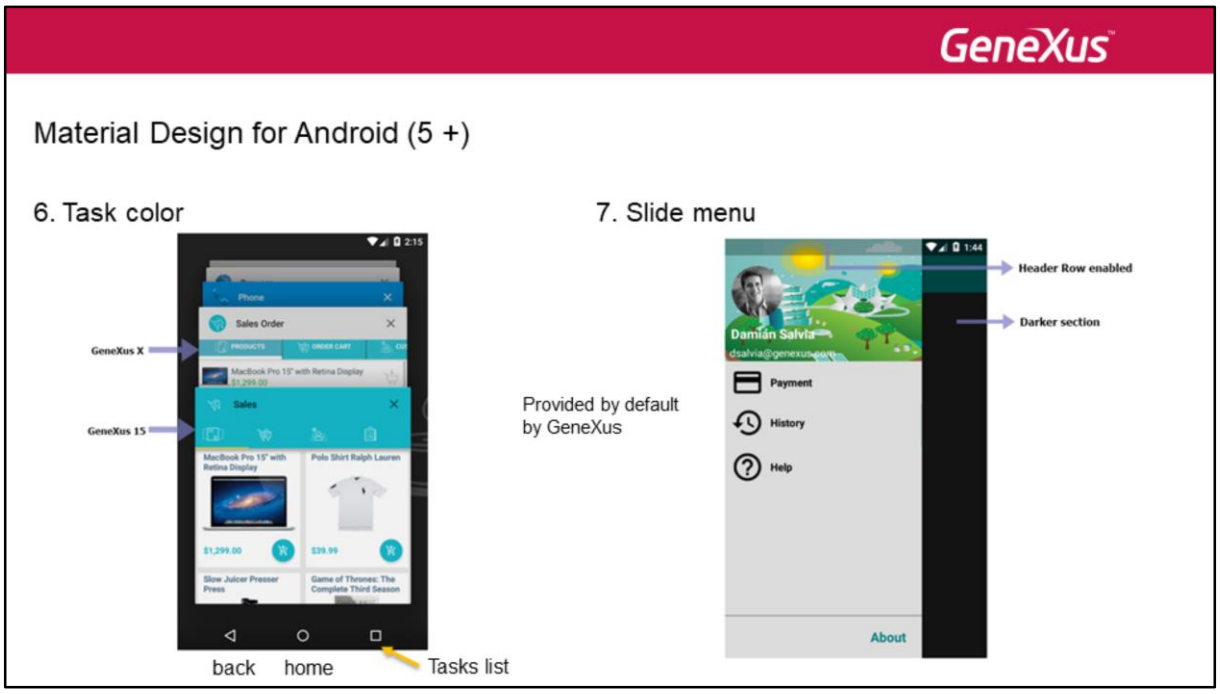
5. Touch ripples



Provided **by default** by GeneXus

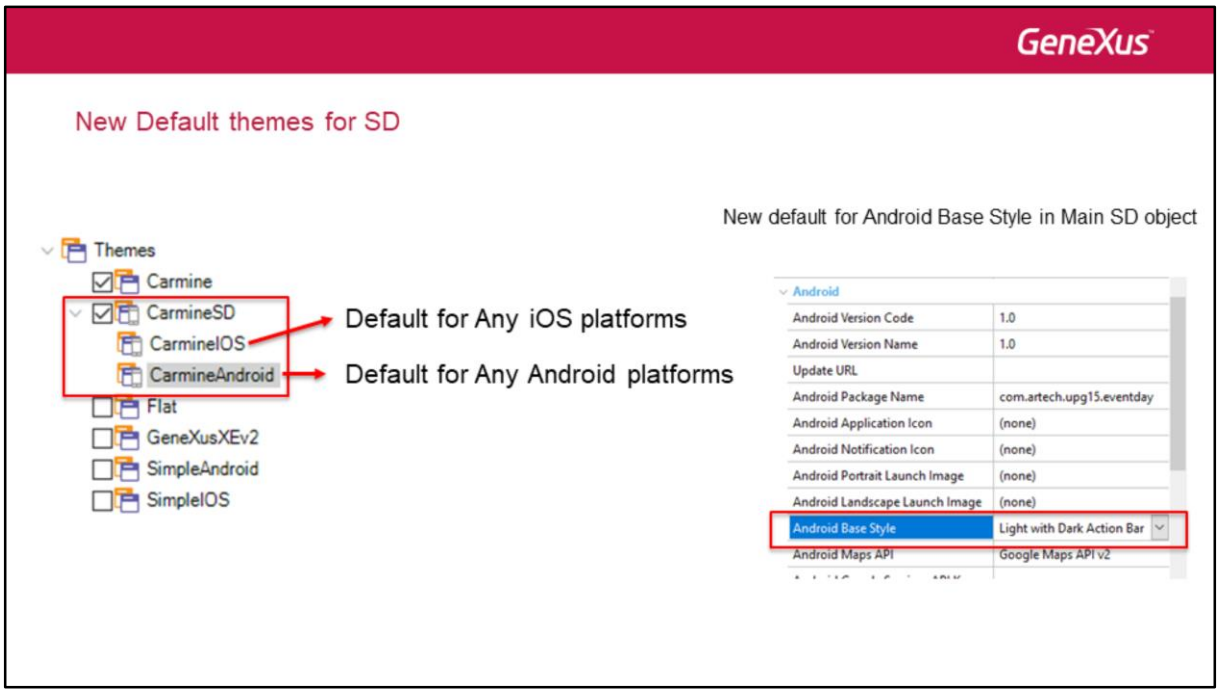
4. Tab strip: las pestañas se muestran con una imagen y un color de fondo, un indicador del tab activo con un indicador de color y con cierta elevación.

5. Touch ripples: es utilizado para notificar visualmente al usuario que el tap que ha realizado ha tenido efecto. Cuando el usuario final hace tap sobre un control que tiene un evento asociado, un círculo destacado se expande desde el punto donde hizo el tap hasta los bordes del control, rellenándolo por completo, mostrando que el evento se está disparando.



6. Task color: los dispositivos Android proveen tres botones físicos/"capacitivos", cada uno dedicado a una acción particular. Además de los botones de back y home, tenemos el botón que muestra con un tab switcher una cascada de thumbnails con las apps recientemente cerradas por el usuario, para poder volver a ellas. De estos thumbnails se muestra su application bar, sus íconos y etiqueta.

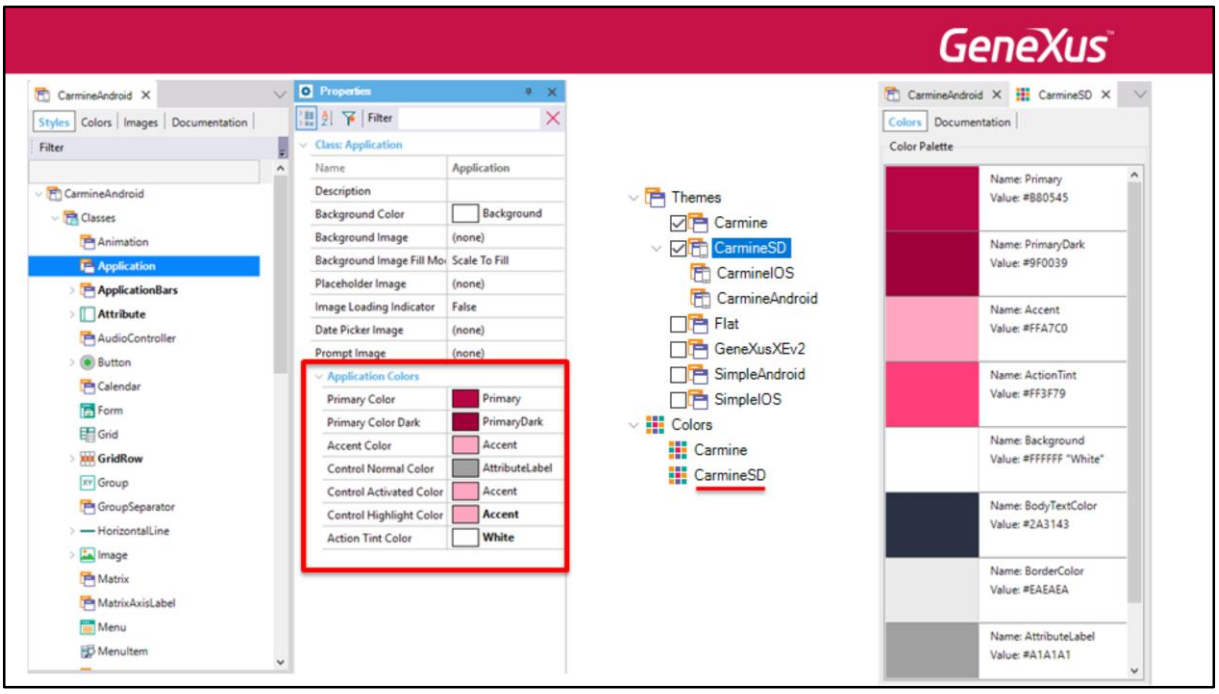
7. Slide menu: cuando el estilo de navegación es Slide, es decir, cuando el menú principal se despliega como una ventana desde la izquierda, su tamaño tiene que seguir las guidelines, y dejar una sombra en la sección derecha cuando está despelgada. Además, si tiene Hero image (es decir, la imagen que aparece arriba del todo ocupando todo el ancho), obsérvese que la ventana del menú debe alcanzar la status bar, preservando su opacidad.



Nuestras aplicaciones deben respetar las guías de diseño para Android y para iOS que mencionábamos. Gran parte del comportamiento ya se da por default sin que el desarrollador deba hacer nada. Pero otras veremos que se implementan mayormente en los themes y otras como propiedades de los controles.

A partir del upgrade 6 de GeneXus 15 hay un nuevo theme, CarmineSD, con dos sub-themes, CarmineiOS y CarmineAndroid que serán los defaults de estas plataformas.

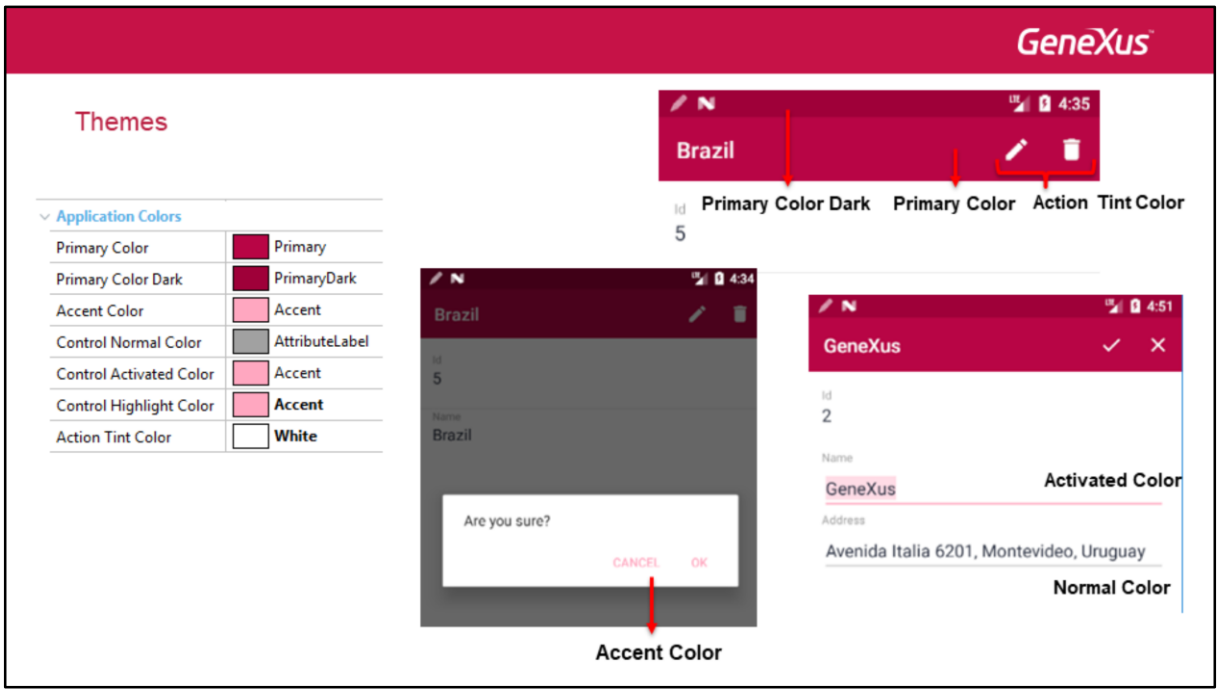
Además para Android se ha modificado el default de la propiedad Android Base Style pasando a “Light with Dark Action Bar”. Esta propiedad es del objeto main de la app SD.



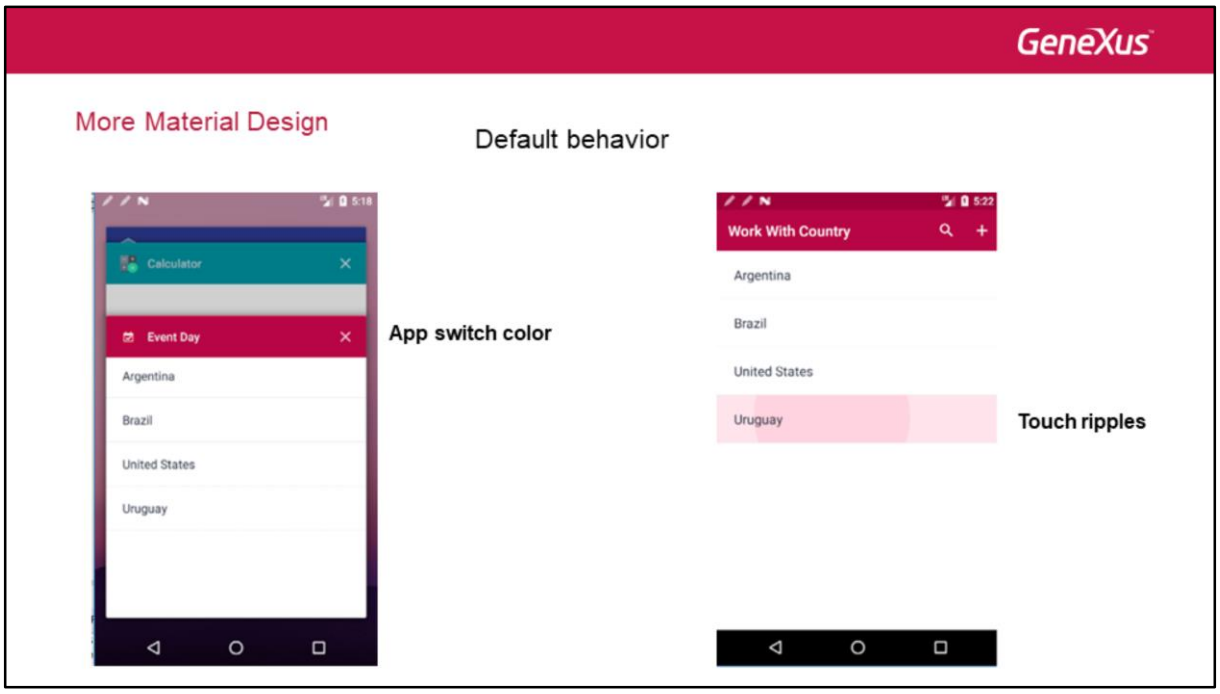
Decíamos que nuestras aplicaciones deben respetar las guías de diseño de Android y de iOS.

Si vamos al theme default para Android, vemos que en la clase **Application** aparecen un conjunto de propiedades agrupadas bajo el nombre Application Colors, que serán las que permitirán definir los colores de acuerdo a los criterios de Material Design que vimos.

¿Qué colores son los definidos como Primary, PrimaryDark, Accent, etc? Los que vienen por defecto en la paleta Colors / CarmineSD.



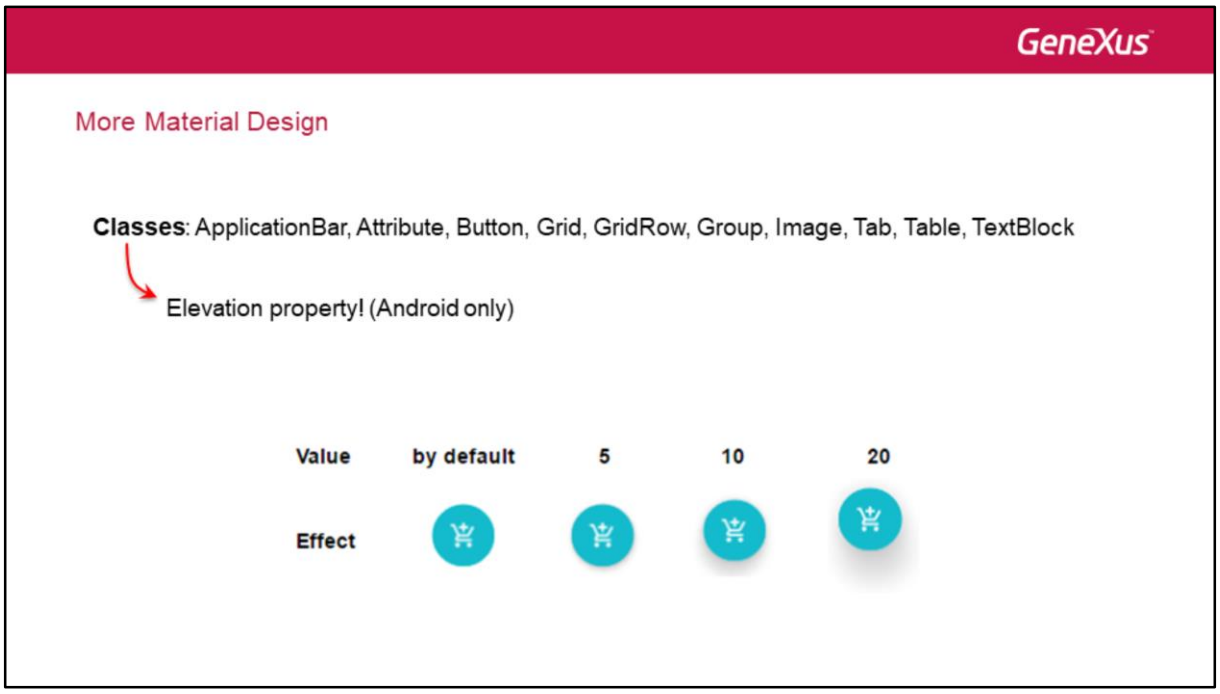
Acá vemos cómo esos colores están siendo utilizados para respetar el Material Design en la app generada.



GeneXus también provee por default dos features que ya mencionamos de Material Design:

La primera es que el app switch color coincide con el brand color de la app.

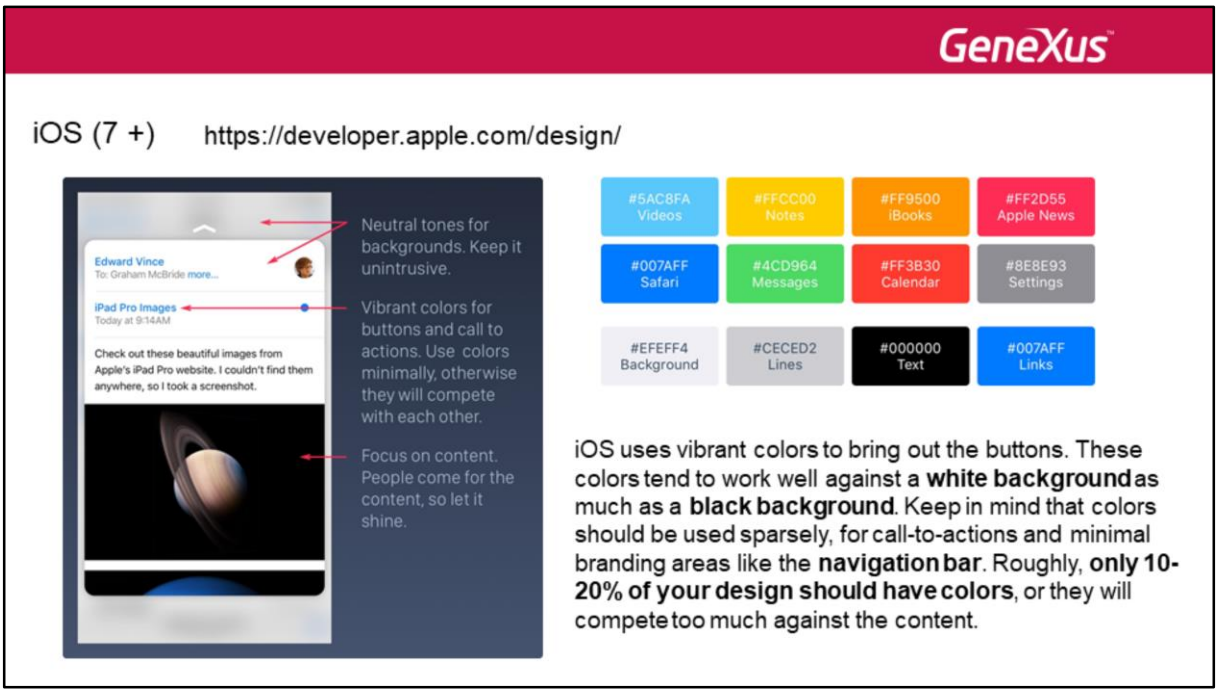
La segunda es la conocida como el efecto Touch Ripples, que produce un tipo de onda expansiva cuando el usuario hace tap sobre el control. En el ejemplo, sobre la fila Uruguay, para poder ver su detalle.



Podemos especificar que controles de la app Android asociados a las clases que se señalan arriba, o subclases de estas, puedan tener un sombreado, de acuerdo al valor de la **propiedad Elevation**, de estas clases. De este modo se estarán siguiendo las guías del Material Design.

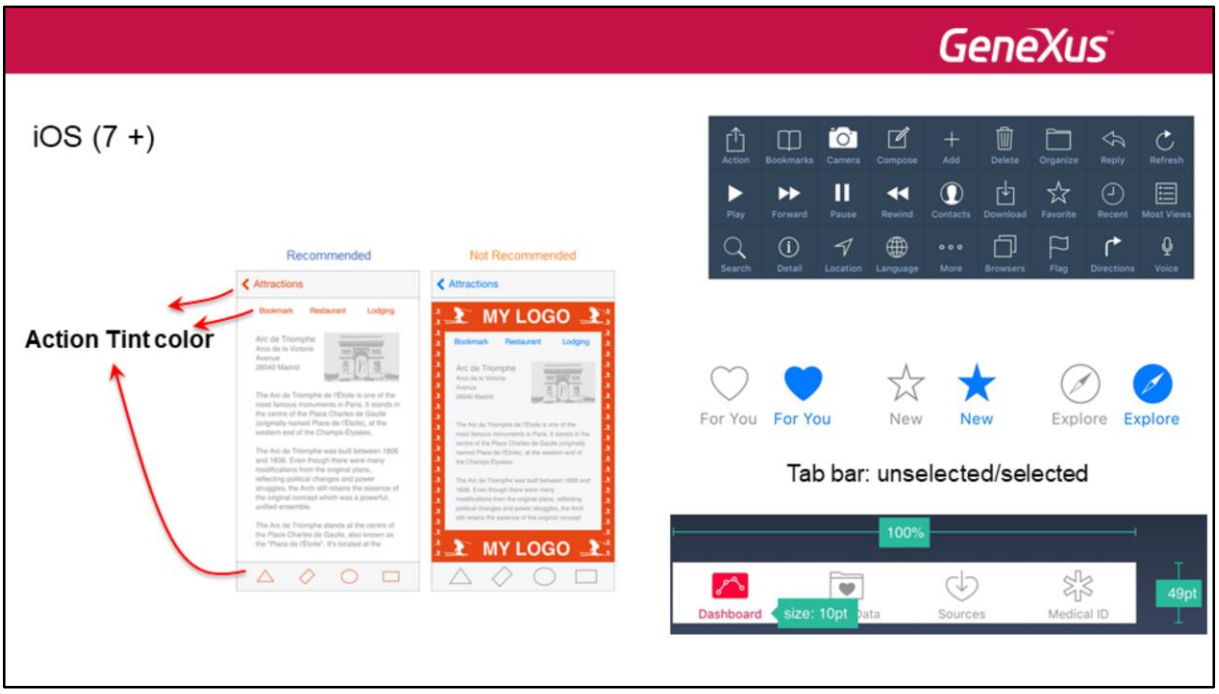
Luego vamos a mencionar cómo se combinan clases de los themes con otras propiedades a nivel de los controles para lograr efectos como el paralaje, motion en iOS, la hero image que mencionamos antes en el caso de Android, etc.

Design Guides: iOS



En el sitio de Apple se encuentran las distintas guías de diseño para las diferentes versiones de iOS. El cambio fundamental se dio con la liberación de iOS 7.

Una de las particularidades de sus guías de diseño tiene que ver con el uso minimalista de los colores y la información de pantalla.

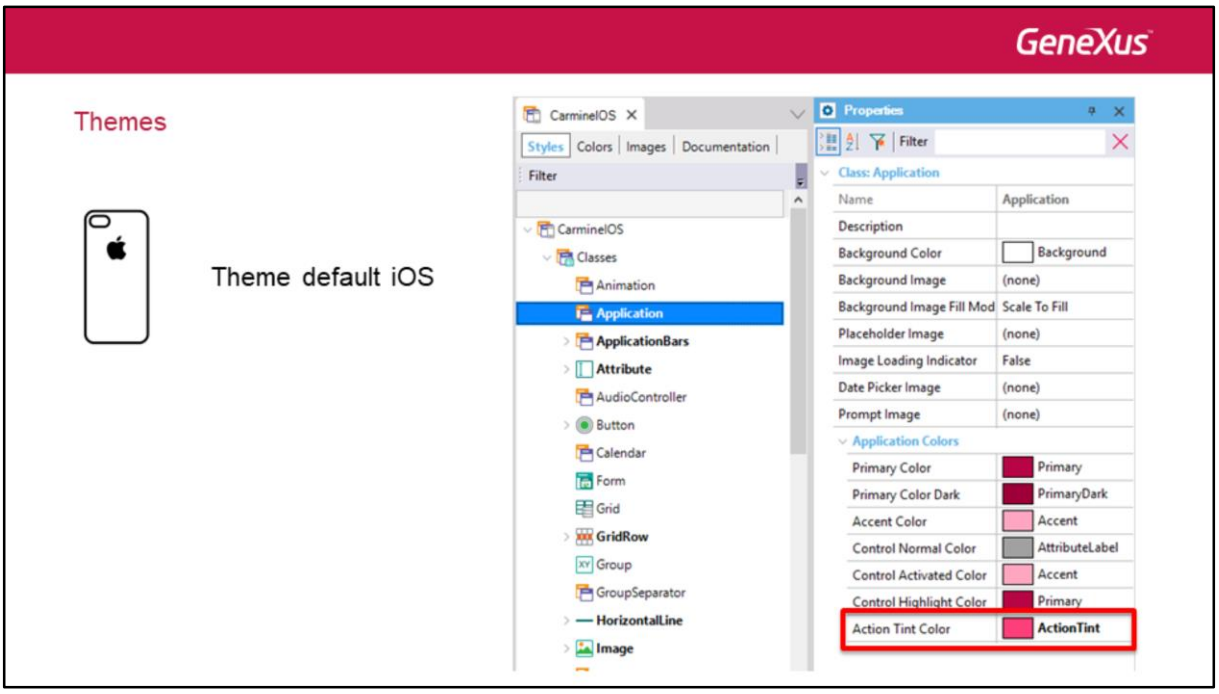


Se recomienda utilizar siempre un único color (preferentemente el de la marca) para todas las acciones que se le brindan al usuario, y no utilizar ese color para otras cosas. De este modo el usuario sabrá que siempre que aparezca ese color, se tratará de un elemento sobre el que podrá ejecutar una acción.

Encontraremos, por otro lado, la recomendación de utilizar los íconos generales (que se muestran en la imagen) solamente (y siempre) para esas acciones, de modo de no confundir al usuario.

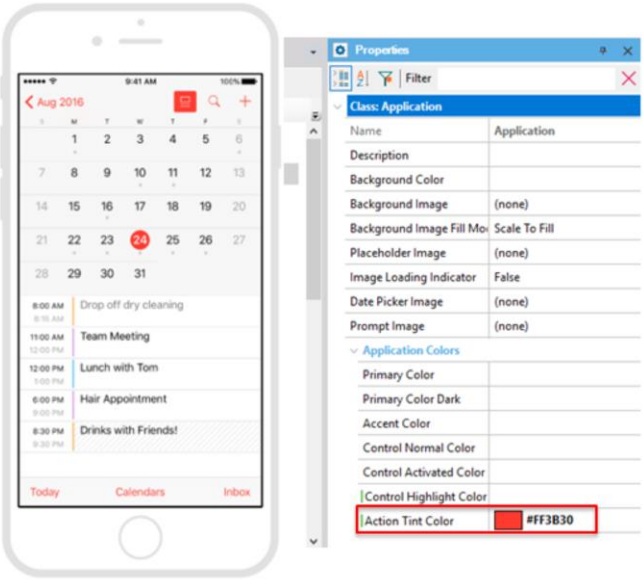
En la tab bar, que provee la navegación principal entre pantallas, se recomienda evitar el menú de hamburguesa si los ítems son pocos y agregar texto a los íconos de forma discreta cuando no son los universales. Si no están activos, se sugiere que los íconos se muestren con un contorno sin rellenar, para que reciban menos atención (deben proveerse dos versiones de cada ítem: para cuando está seleccionado y cuando no).

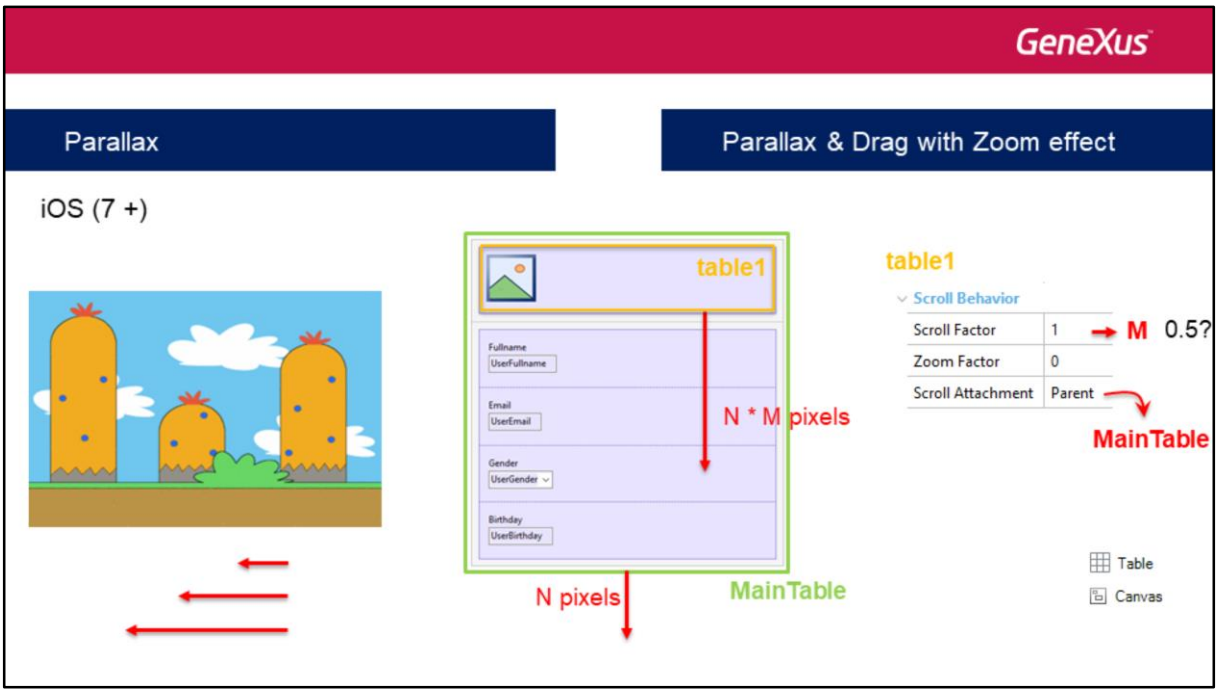
También podemos ver que hasta los tamaños que deben ocupar los controles están establecidos. Todos los íconos de la app deberían ser iguales en términos de tamaño, nivel de detalle y borde.



La única propiedad que aplica para iOS es Action Tint Color, para pintar íconos y los controles estándar (como Control Normal y Control Activated colors en Android).

Themes


9:41 AM
100%
Aug 2016
Drop off dry cleaning
Team Meeting
Lunch with Tom
Hair Appointment
Drinks with Friends!
Today
Calendars
Inbox
Properties
Filter
Class: Application
Name
Application
Description
Background Color
Background Image
(none)
Background Image Fill Mo
Scale To Fill
Placeholder Image
(none)
Image Loading Indicator
False
Date Picker Image
(none)
Prompt Image
(none)
Application Colors
Primary Color
Primary Color Dark
Accent Color
Control Normal Color
Control Activated Color
Control Highlight Color
Action Tint Color
#FF3B30



El efecto Parallax es un efecto visual utilizado frecuentemente en aplicaciones para Smart devices y también en páginas web. Consiste en desplazar un objeto por sobre otro que está por detrás, a distinta velocidad uno del otro, dando la sensación de profundidad.

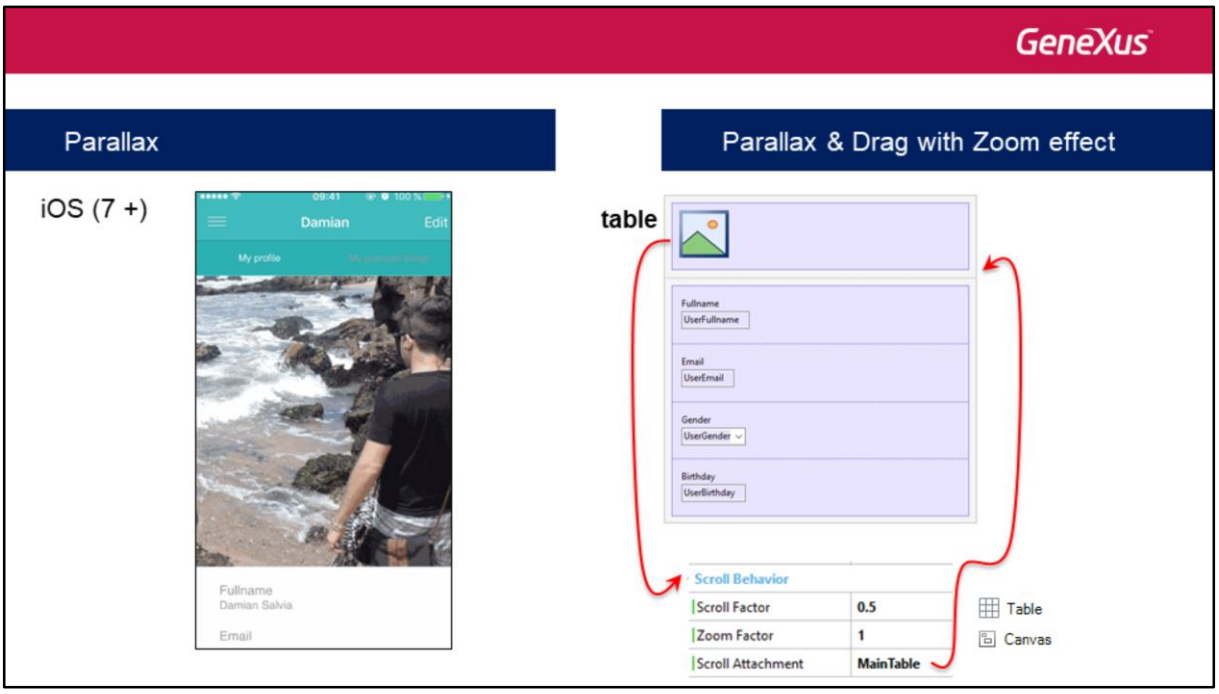
En el ejemplo de la izquierda vemos un efecto de paralaje con tres capas que se mueven a distintas velocidades. La más interior es la que se mueve más lento.

En GeneXus el efecto de paralaje se consigue para una **tabla** o **canvas** y su contenido, en relación al contenedor u a otros controles especificados en la propiedad Scroll Attachment.

Por ejemplo, si tenemos una table1 con una imagen, y esa table1 está contenida en una MainTable, y queremos que cuando se hace scroll sobre esta MainTable, por ejemplo hacia abajo, por cada N pixels que esa MainTable y todo su contenido se desplacen hacia abajo la table1 se desplace N*M pixels, entonces debemos editar las propiedades de la table1 y bajo el grupo Scroll Behavior modificar el valor por defecto 1 de la propiedad Scroll Factor por M.
Ese factor de scroll lo es en relación al control o controles que se especifiquen en la propiedad Scroll Attachment. Por defecto es el contenedor de esta tabla, que en nuestro caso coincide con MainTable, pero pueden ser una lista de controles.

Observemos que si M > 1 entonces al hacer scroll sobre MainTable la table1 se desplazará en la misma dirección pero más rápido que el resto del contenido de MainTable. Si M < 1, entonces será al revés.

Si M = 0.5 entonces table1 y su contenido se desplazarán a la mitad de velocidad que el resto de los controles de MainTable, por lo que si el scroll es hacia abajo, quedará un espacio vacío entre la imagen y los atributos que le siguen. Allí podemos utilizar la otra propiedad, Zoom Factor, para hacer que la table1 y su contenido hagan un zoom-in para ocupar más espacio en ese caso.



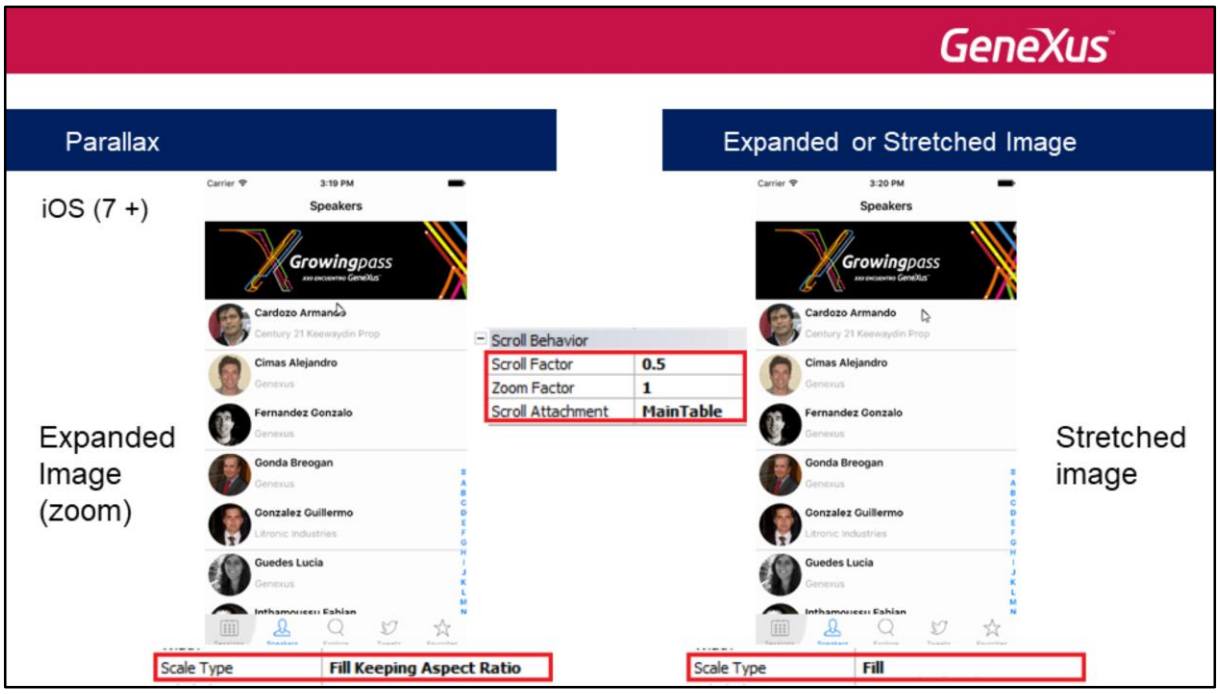
El efecto de Drag with Zoom consiste en expandir un control (por ejemplo una imagen) de fondo, cuando el usuario hace un Drag hacia abajo (o Pull down), al control que está sobre él. Como el control que está por arriba al desplazarse hacia abajo deja lugar, la imagen del fondo puede expandirse con un zoom, ocupando ese lugar.

Para conseguir estos efectos, usamos la propiedad Scroll Attachment y combinamos las propiedades Scroll Factor y Zoom Factor de las tablas y los canvas.

La propiedad Scroll Factor ya la vimos. El Zoom Factor es un valor decimal (por defecto 0, es decir, no zoom) que indica qué tanto zoom in/out hay que aplicar al control cuando el usuario hace scroll sobre los controles de la propiedad Scroll Attrachment. Valores positivos o negativos provocarán que sea un zoom in o out.

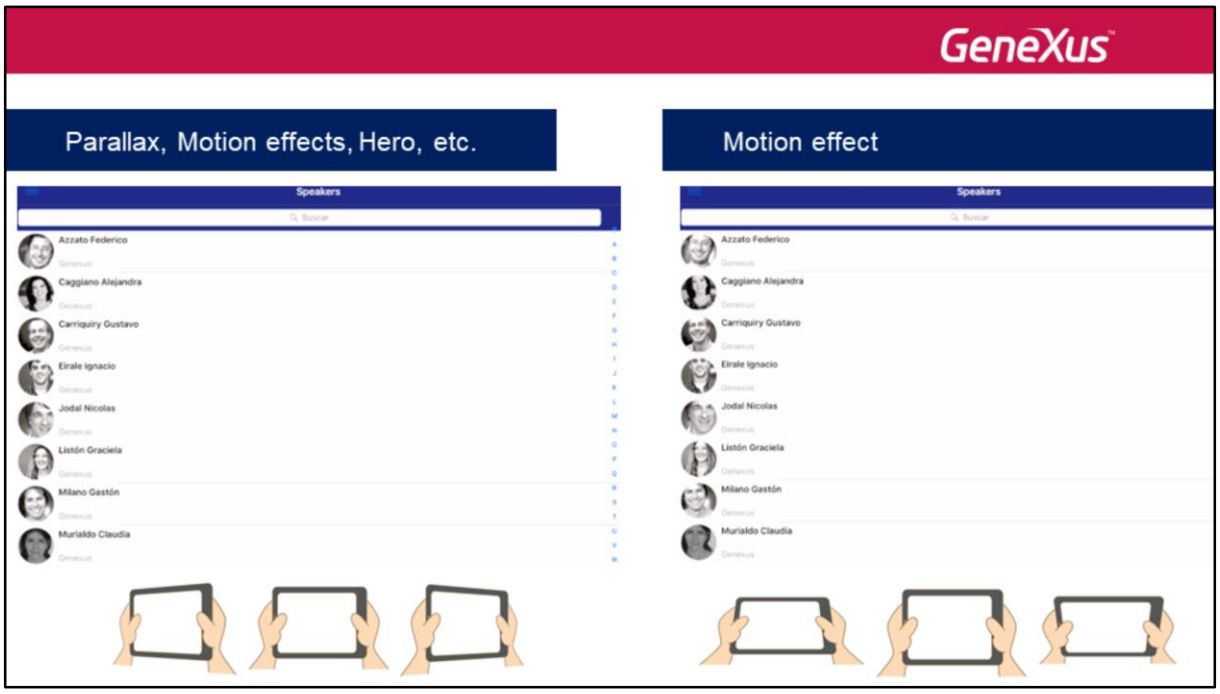
Este efecto solamente es soportado en iOS 7 o superior.

Mas información:
https://wiki.genexus.com/commwiki/servlet/wiki?31174,Scroll+behavior+properties+group,



Un caso particular es al hacerse un drag o pull down con una tabla o canvas que contiene una imagen. Podemos querer que en lugar de expandirse en todas direcciones (hacerse un zoom) como vimos antes, solamente se estire.

Esto se logra utilizando los mismos valores de las propiedades Scroll Factor y Zoom Factor que para hacer un zoom en la imagen, sólo que para que la imagen se estire en lugar de expandirse cambiamos el valor de su propiedad Scale Type al valor Fill.

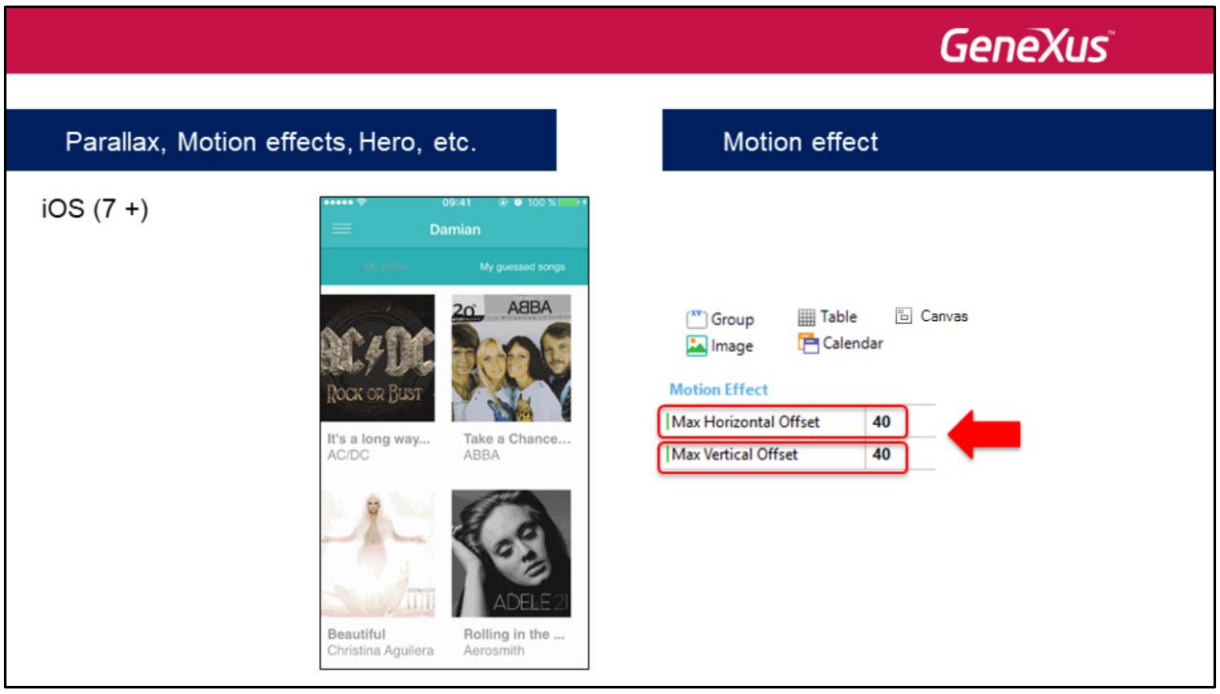


De la misma manera en que Android da la sensación de profundidad con la propiedad elevation, iOS lo hace a través de capas y usando el giroscopio. Este efecto es llamado Motion Effect.

El efecto de movimiento se logra rotando el dispositivo sobre su eje horizontal o vertical, lo que provoca un movimiento en el control, haciendo que éste se mueva como si no estuviera fijo al dispositivo.

Si el movimiento es horizontal, la imagen se desplaza hacia la derecha o la izquierda, en sentido contrario al movimiento, dando sensación de profundidad.

Si el movimiento es vertical, el desplazamiento es hacia arriba o hacia abajo. También es posible combinar ambos movimientos, cambiando la posición del dispositivo en ambas direcciones a la vez.



Para lograr el efecto de movimiento, se utilizan las propiedades Max Horizontal Offset y Max Vertical Offset de cualquiera de las clases listadas. Por defecto están en 0.

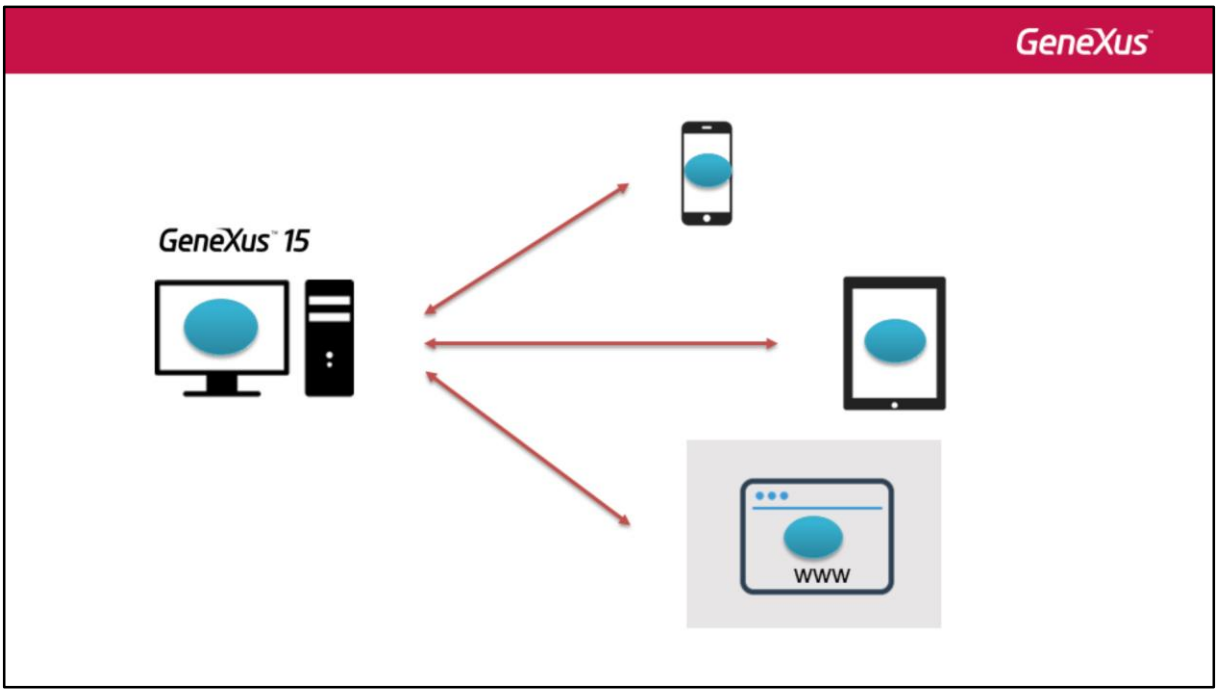
La propiedad Max Horizontal Offset indica el máximo offset que el control puede alcanzar, cuando el usuario inclina el dispositivo horizontalmente hacia la derecha o hacia la izquierda. Se interpreta como un factor de movimiento, a mayor valor, mayor movimiento.
Análogamente, la propiedad Max Vertical Offset indica el máximo offset del control, cuando el usuario inclina el dispositivo hacia adelante o hacia atrás.

Un valor negativo en estas propiedades, indica que el control se mueve en dirección opuesta al movimiento del dispositivo físico.
Un valor de cero implica que no hay efecto, y un valor positivo hará que el movimiento sea en la misma dirección que toma el dispositivo.

Este efecto aplica a controles imagen, grupos, tablas, canvas y calendarios. Y solamente es soportado en iOS 7 o superior.

Más información:
https://wiki.genexus.com/commwiki/servlet/wiki?30953,Motion+Effect+properties+group,

Live editing

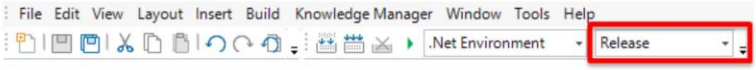


Hasta GeneXus 15 cada vez que queríamos probar un cambio de diseño debíamos volver a especificar, generar y compilar la aplicación.

Pero en GeneXus 15 contamos con la herramienta de Live editing, que permite realizar cambios de diseño y algunos de comportamiento y ver instantáneamente esos cambios en la app en el dispositivo, o en el browser de la app web, sin tener que compilar nada, y sin que siquiera esos cambios deban ser grabados.

GeneXus

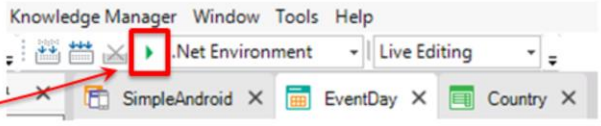
Step 1



Step 2

Run a SD main object

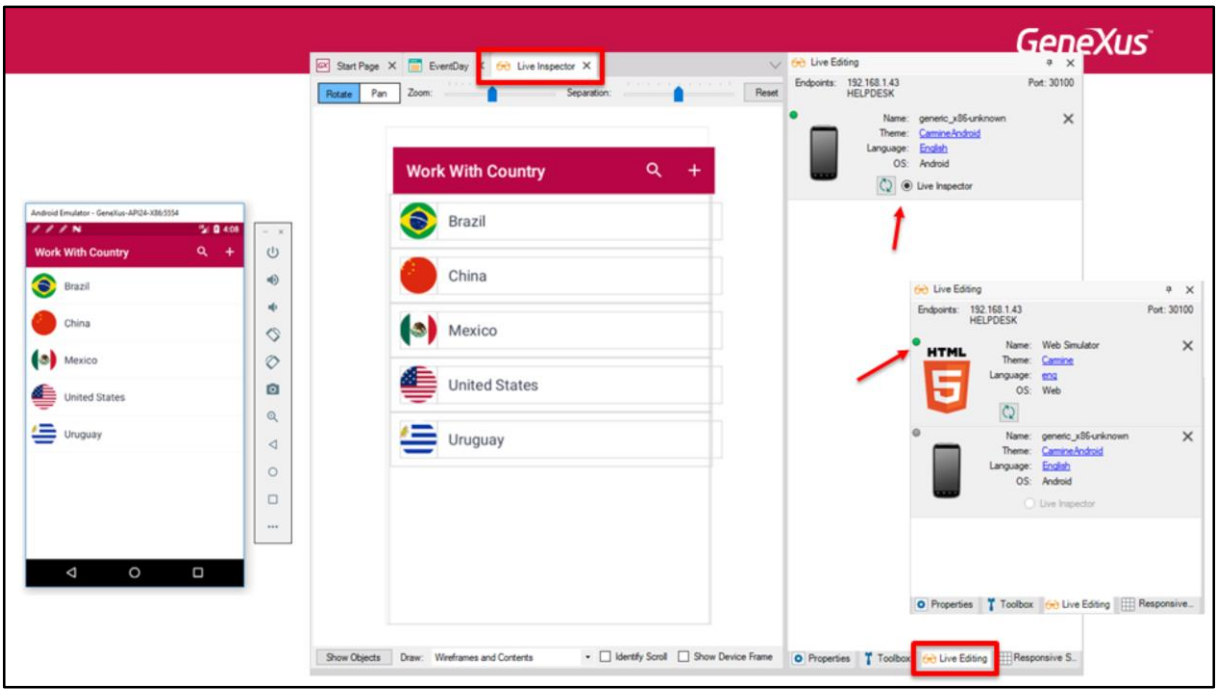
Having a SD main object as Startup Object:



Para usar el Live Editing se necesitan 2 pasos:

- primero cambiar el combo del valor Release al valor Live Editing
- Luego ejecutar la aplicación (asegurándonos de que se compile. Live editing no funciona con el KBN)

De esta manera el servidor de Live Editing quedará “escuchando” los cambios que hagamos a la aplicación y replicará estos cambios sobre la metadata a la que accede el dispositivo móvil (en nuestro caso el emulador) y que se utiliza para dibujar la pantalla en el dispositivo y definir su comportamiento.



Como consecuencia del Run se abrirá la app SD en el emulador en nuestro caso (si tuviéramos un dispositivo conectado a la computadora se abriría en el dispositivo), y en el IDE veremos que en el sector de las propiedades aparece una nueva solapa llamada Live Editing, y en el sector principal una ventana Live Inspector, que se puede habilitar/deshabilitar desde la primera.

En la solapa Live Editing veremos una lista de todos los dispositivos con los que estamos prototipando (todos aquellos sobre los que hayamos Compilado y tengamos la app abiertas) y por tanto con los que el IDE está conectado, listos para trabajar con Live Editing. En este caso, solo uno.

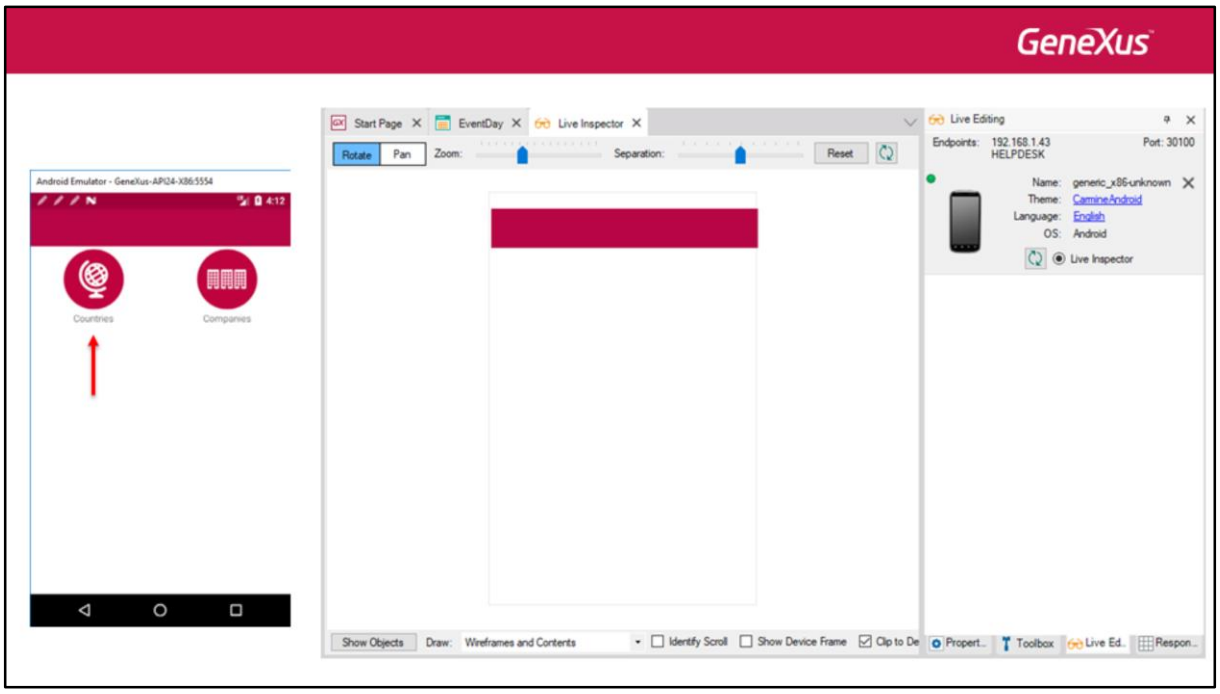
Para Live Editing usamos un canal bidireccional entre el dispositivo que está ejecutando la aplicación y el IDE, que funciona como server.

En el ejemplo, el IDE está escuchando en el 192.168.1.43:30100 (en la computadora llamada HELPDESK) y el dispositivo que ejecuta la aplicación es el de nombre generic:_x86-unknown que corresponde al emulador de Android.

Podemos ver el nombre del dispositivo, el theme que está utilizando, el lenguaje y el sistema operativo que es Android.

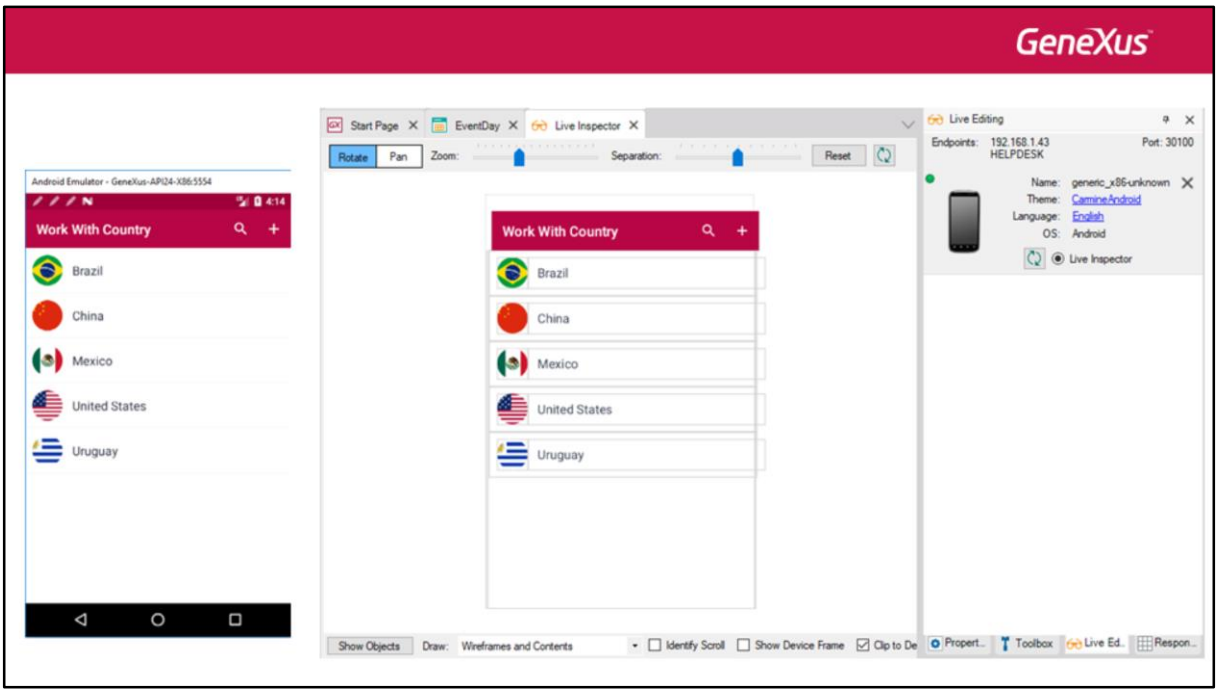
Si hiciéramos F5 sobre el objeto Home web, por ejemplo, mostramos a la derecha cómo se verá la ventana de Live Editing cuando se abra el browser como consecuencia del proceso de Run. Con el punto verde se muestra cuál es el dispositivo que se está escuchando en ese momento.

Además tenemos una opción Live Inspector, que al seleccionarla abrirá una ventana donde podremos visualizar la aplicación tal cual la vemos en el emulador, pero que, a diferencia del emulador, nos permite desgranar la información y ver las diferentes capas que componen el layout, los nombres y clases de cada control, etc., y realizar cambios directamente sobre ella y verlos impactados automáticamente.

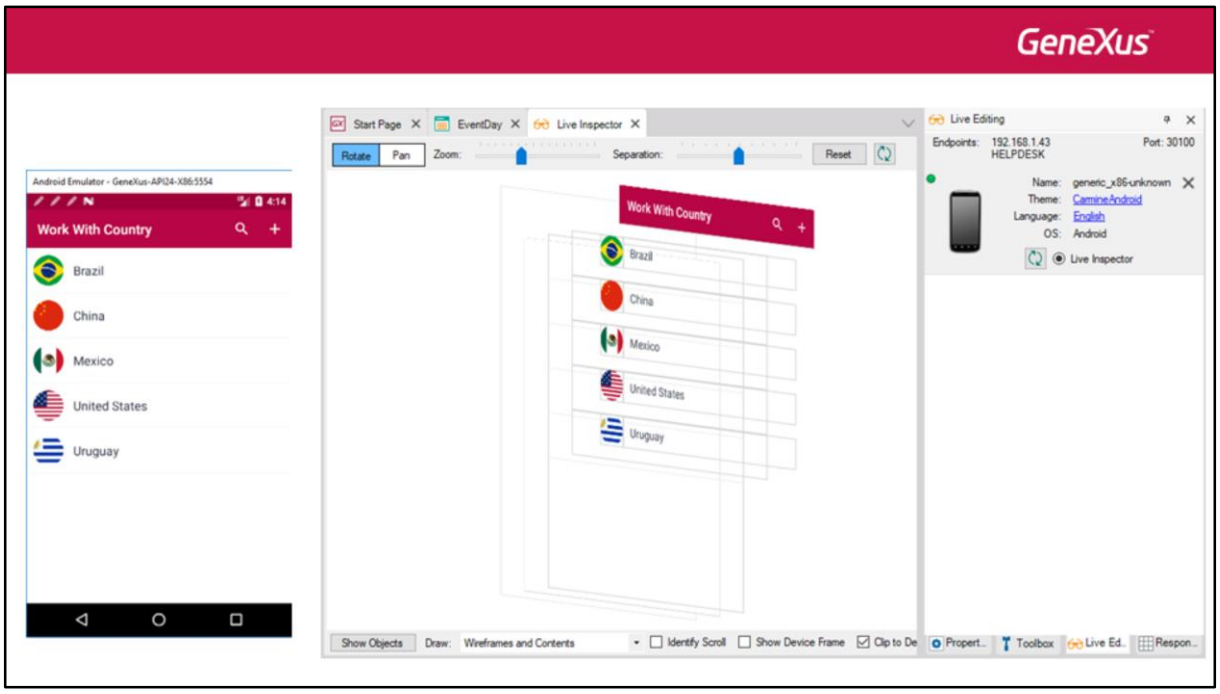


El objeto de tipo Menu for Smart Devices no es *inspeccionable*, razón por la que aparecerá vacío en la ventana de Live Inspector.

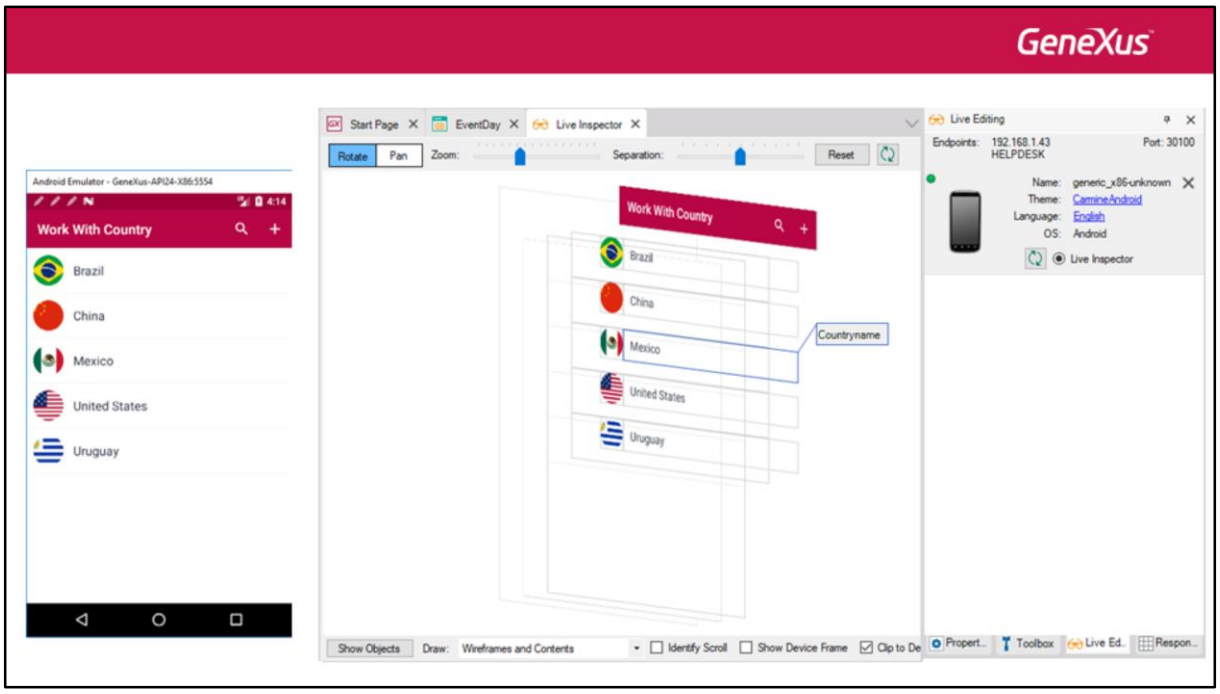
Seleccionemos, por ejemplo, el List de países en el dispositivo.



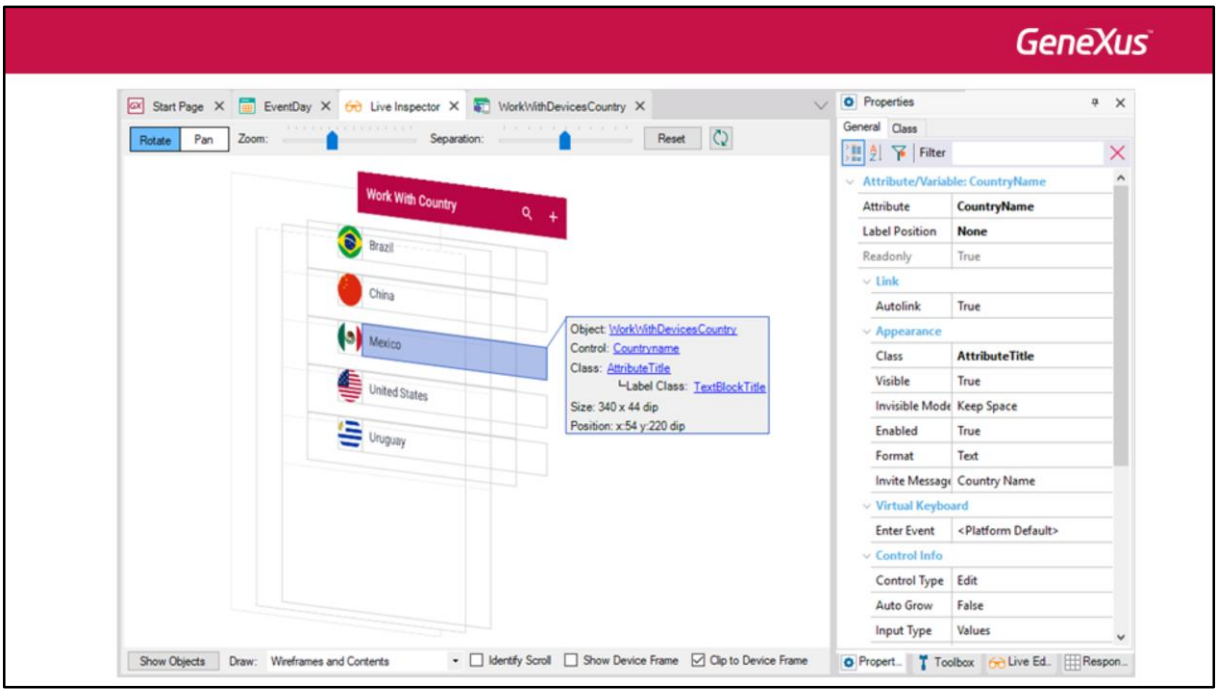
La ventana Live Inspector nos permite inspeccionar los controles de los que está compuesto ese layout. x



Como primera cosa, observe que haciendo botón izquierdo o derecho con el mouse sobre cualquier punto del contenido de la ventana, dejándolo presionado y moviendo el mouse logrará girar el layout para ver sus capas en 3 dimensiones.

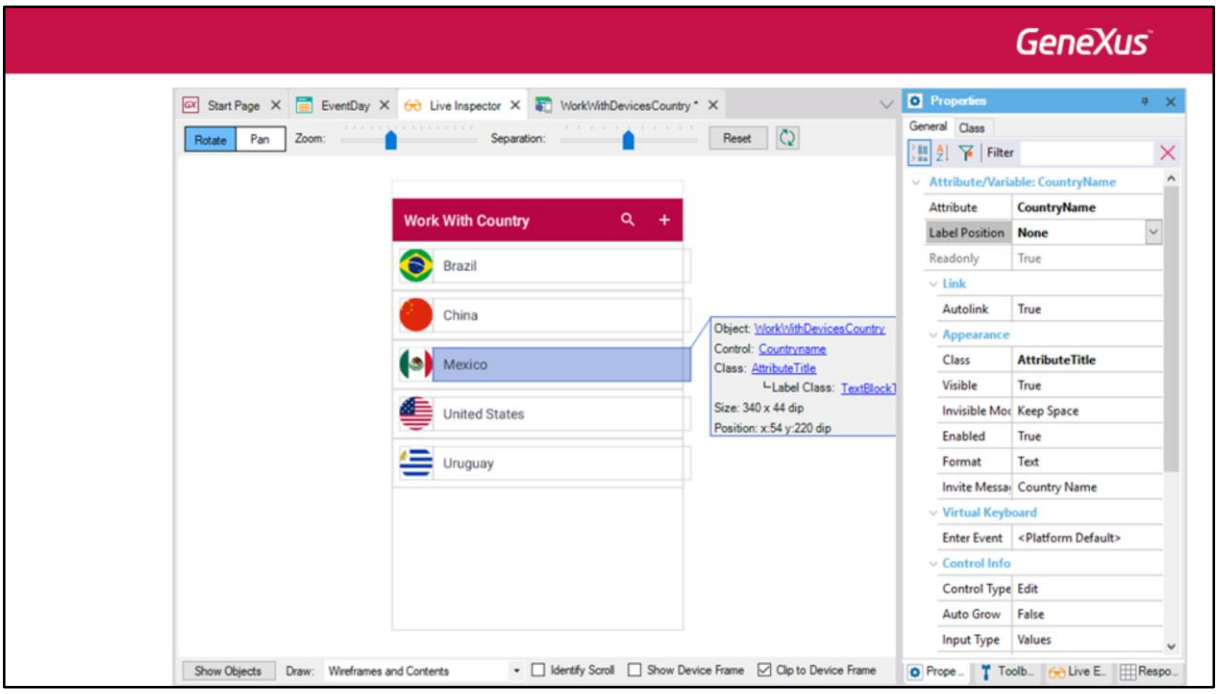


Y si desplaza el mouse por la ventana irá viendo que se le muestran los tipos de controles por los que está pasando el mouse.



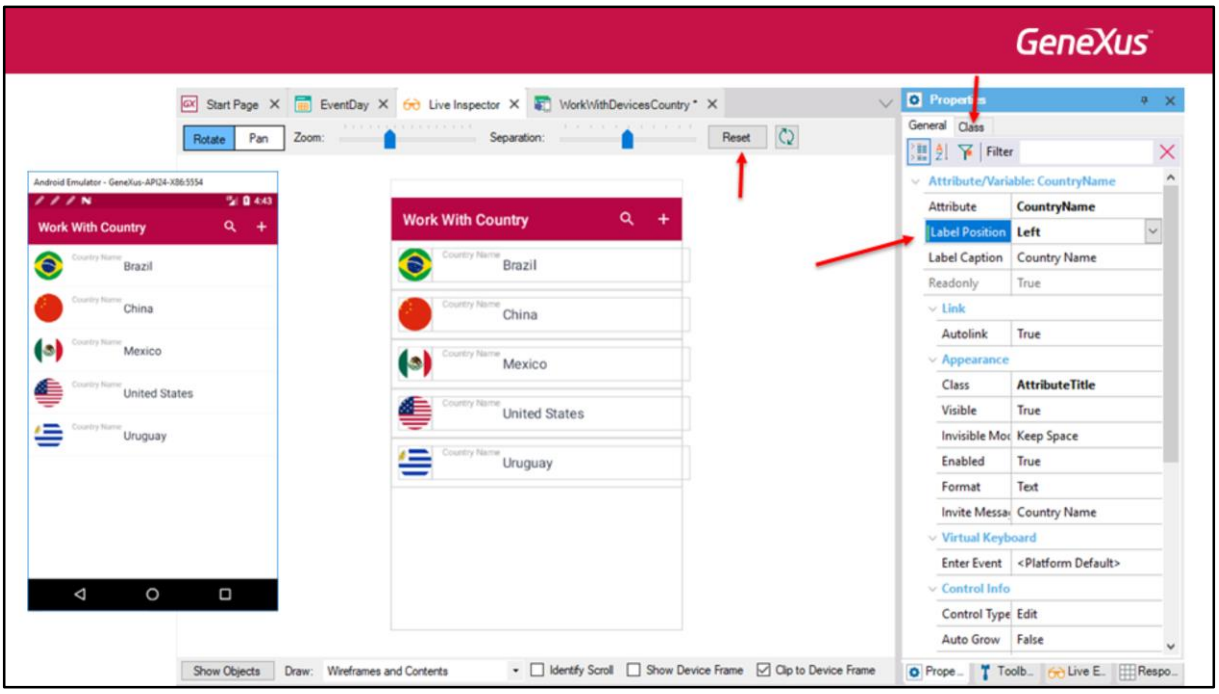
Si hace clic sobre uno, se le muestra la información de en qué objeto se encuentra, el nombre del control, la clase y subclases si es que las tiene, además del tamaño y posición en la pantalla.

Simultáneamente se le muestra en la ventana de propiedades, como para que pueda modificar cualquiera de sus propiedades, incluyendo la clase, o incluso las propiedades de la clase.



Para verlo mejor, presionemos Reset para que nos vuelvan a aparecer las capas superpuestas.

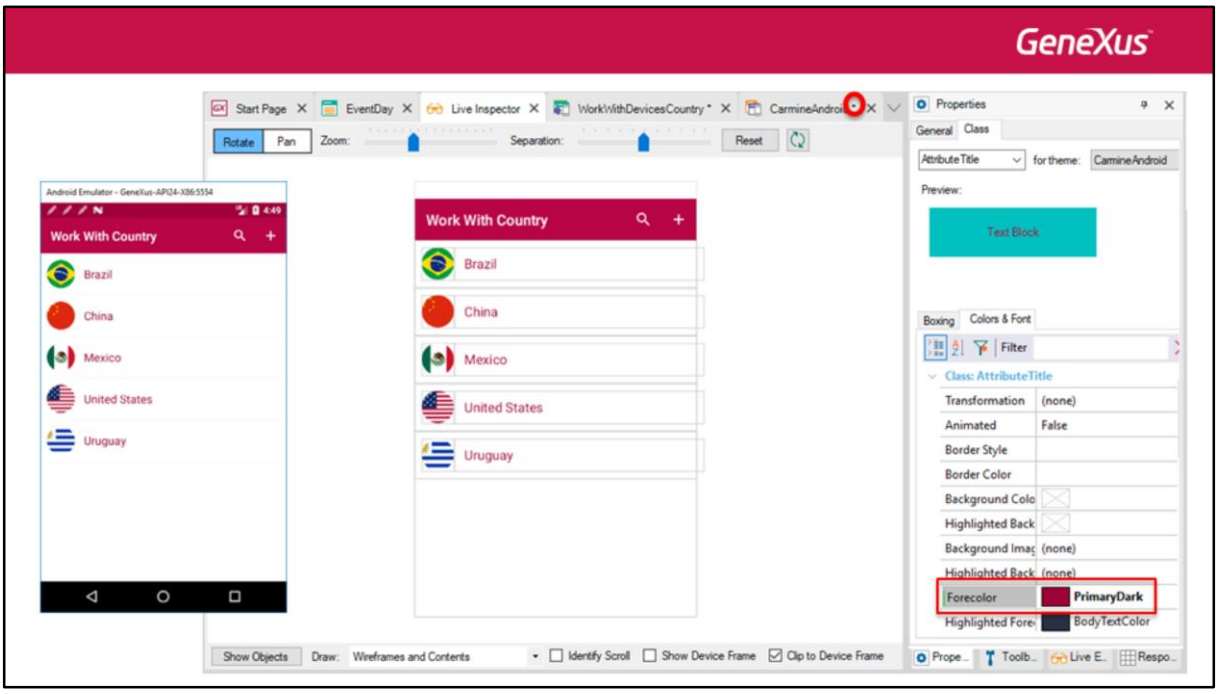
Y ahora modifiquemos Label Position por “Left”.



Para verlo mejor, presionemos Reset para que nos vuelvan a aparecer las capas superpuestas.

Y ahora modifiquemos Label Position por “Left”. Observemos que se refleja automáticamente en el dispositivo y en la ventana de Live Inspector.

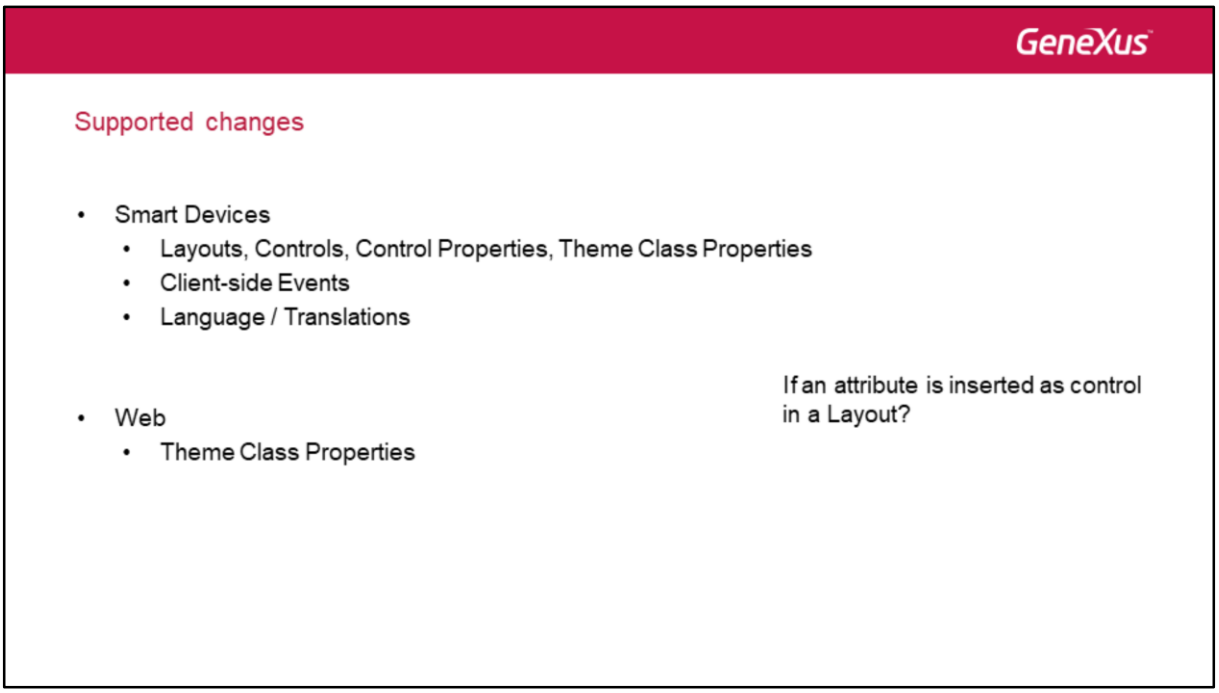
Dejémoslo como estaba, con Label Position “none” y ahora modifiquemos alguna propiedad de la clase del atributo. Por ejemplo el color.



Observe cómo se abrió automáticamente el theme CarmineAndroid y está indicando que hubo un cambio pero que aún no se ha grabado.

Si estos cambios son los que deseaba, entonces puede grabar y en el próximo Run la app se verá de esta forma.

Con esto vimos un ejemplo sencillo de lo que podemos lograr con Live Editing.

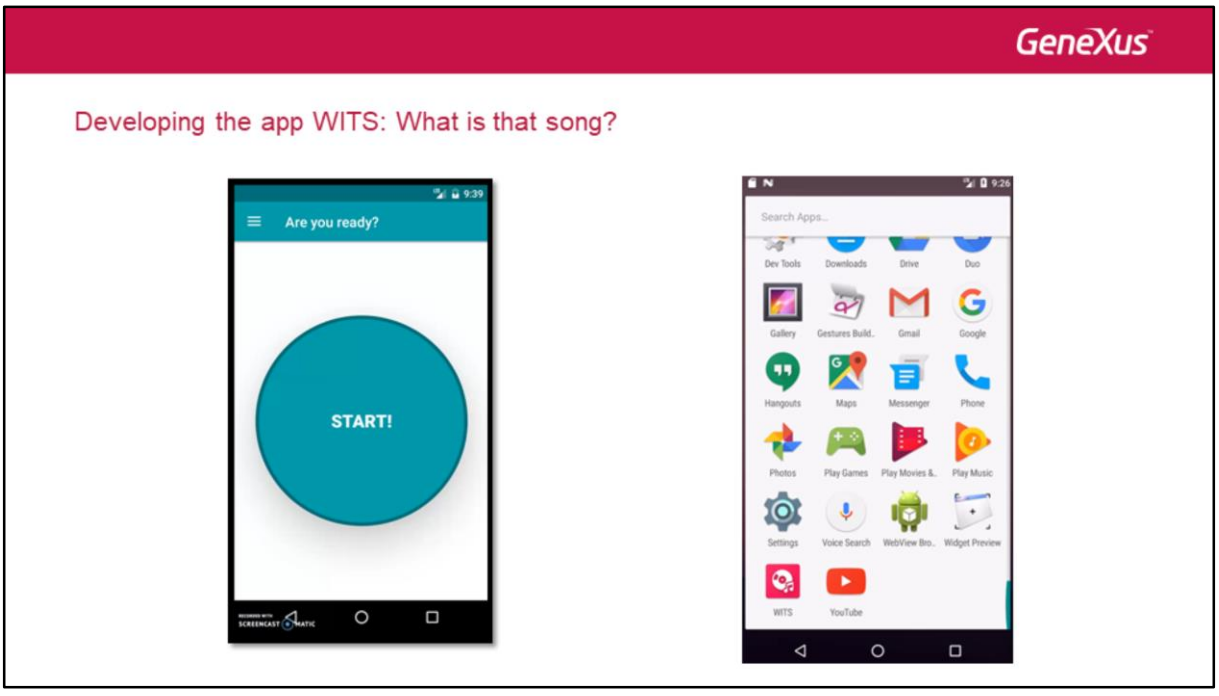


¿Qué tipo de cambios pueden verse “en vivo” en el dispositivo y la pantalla de Live Inspector?

En aplicaciones Smart Devices, cambios en layouts (filas, columnas de tablas), en propiedades de diseño de los controles, propiedades de las clases del theme de los controles, cambios en eventos que se ejecutan en el cliente, cambios en los textos relacionados a la traducción.

Note que hay cambios tales como en los eventos en el server en Smart Devices (Start, Refresh, Load), o atributos que se agregan como campos a un layout, que no pueden verse en vivo con Live Editing. Esos cambios siguen requiriendo el ciclo normal de Build.

User/Developer Experience



Se ha desarrollado una aplicación que es un juego con cinco etapas, en cada una de las cuales suena una canción y se le dan en pantalla al usuario los nombres de 5 canciones para que elija en menos de 15 segundos cuál es la canción que está sonando en ese momento.

Tenemos dos versiones de la app. La primera es anterior a la salida del theme Carmine para Smart Devices. La segunda es posterior, y además tiene funcionalidades que fueron apareciendo en upgrades posteriores de GeneXus 15.

Observemos que más allá de los colores la app contiene un menú slide, que ofrece loguearse con Facebook. Para comenzar el juego debe presionar un botón Start que tiene el efecto de ripple que decíamos que era automático en las apps Android posteriores a la versión 5.

Se carga una pantalla con la ronda actual del juego, donde pueden verse los nombres de cinco canciones y una que comienza a sonar. El usuario deberá hacer tap sobre el nombre de la canción que está sonando. Puede verse una barra que actualiza los éxitos y los fracasos a medida que van pasándose rondas del juego. Son cinco rondas. En cada una un temporizador mostrará el tiempo que resta para adivinar la canción de la etapa actual.

Finalizadas las cinco rondas, se nos muestra una pantalla con el resultado, que puede compartirse, por ejemplo por Twitter. Al tener la app de Twitter instalada en el dispositivo, la abre y nos permite compartirlo directamente allí. En caso contario la abrirá en el navegador.

En la segunda app, la de estética Carmine, vemos una secuencia de artistas en un especie de carrusel. Se tratará de un nuevo tipo de grid que no es el Horizontal sino uno llamado SD Flex Grid. Vemos básicamente el mismo menú. Aquí ya se está logueado con Facebook.

Hay un panel con el perfil del usuario, que básicamente muestra el Detail de un Work with con la info general del usuario, una lista de canciones que ya ha adivinado, y chats que ha enviado.

Desde el menú también podemos acceder a una Playlist con las canciones que hemos adivinado en el juego, donde aparecen listadas y abajo vemos un reproductor de audio que permite reproducir esa playlist, tanto en miniatura como en modo full, ocupando toda la pantalla.

Luego tenemos un buscador, que nos permite buscar las canciones con las que trabaja la app por artista, o nombre de la canción. El contenido va modificándose a medida que el usuario tipea en el campo.

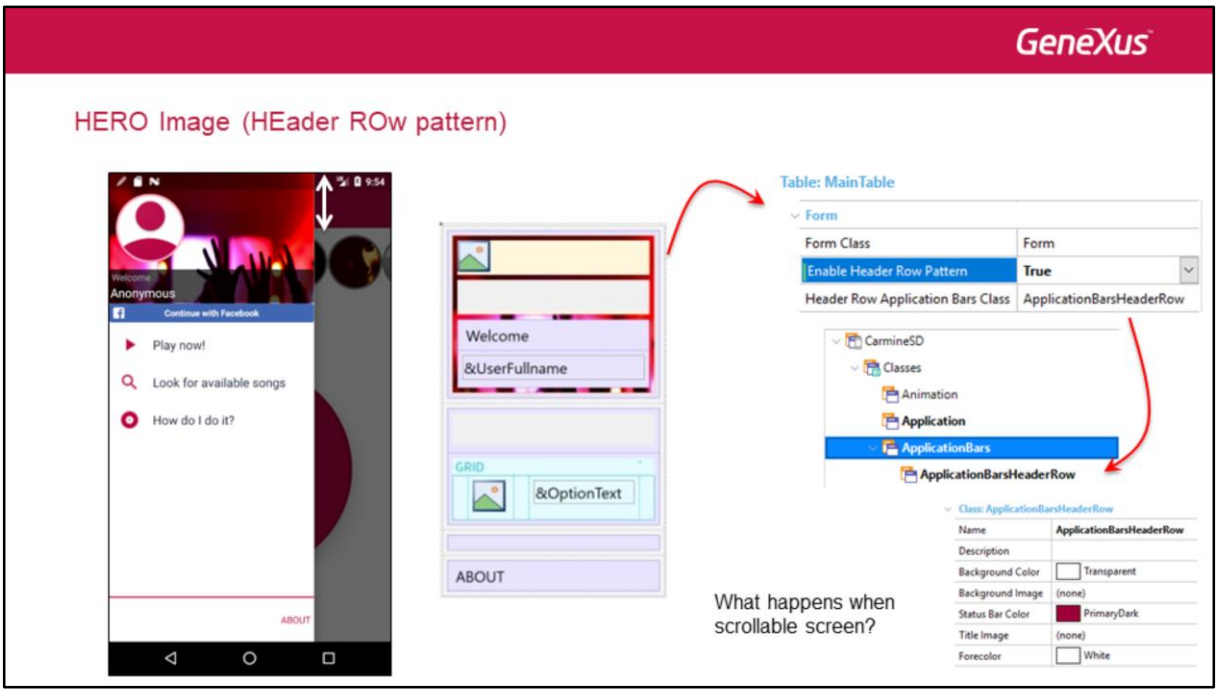
Luego tenemos un chat donde los usuarios podrán compartir mensajes de texto o audio con otros usuarios de la app. Observemos que cuando el usuario aún no ha tipeado nada, aparece el ícono del microfonito para permitirle grabar un audio (se creó un EO AudioRecorder), y que cuando el usuario empieza a tipear cambia el iconito pues será texto lo que se enviará. Observemos que al enviarlo sale una notificación que es la que le llegará al resto de los usuarios.

Y observemos que en esta segunda versión de la app al iniciar el juego, aparece una animación que sustituye a la típica del circulito que indica que la pantalla se está cargando. Estos tipos de animaciones aparecieron con el upgrade 8. Y luego aparece la misma pantalla que vimos antes, aunque con una estética diferente, mostrando las canciones e indicándole al usuario si acertó o no con un mensaje (en este caso el usuario no ha acertado ninguna). El video se grabó sin sonido, por lo que no podemos escuchar la canción que se está reproduciendo.

Veremos luego que al compartir el resultado del juego con esta nueva versión de la app, podremos enviar un link especial, lo que se conoce como Deep linking. Ya lo veremos.

Esta app nos permitirá ir presentando varias de las novedades de GeneXus 15 para Smart Devices. Usted las pondrá en práctica luego con otra aplicación, EventDay.

HERO Image
(HEader ROw pattern)

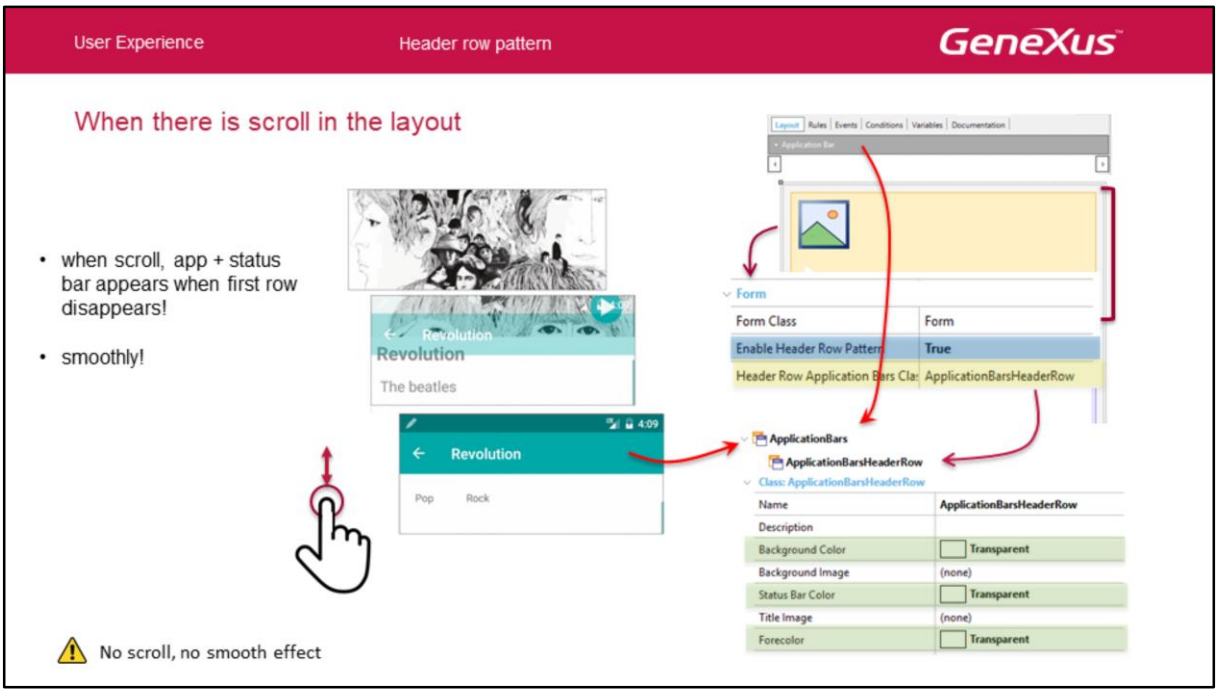


Hero Image, acrónimo de Header Row, se utiliza hoy en todas las apps. ¿Qué significa este patrón de diseño? Que lo que tenemos como primera fila del layout actuará de header, ocupando desde el borde superior, sin dejar espacio para la Status Bar y la Application Bar. Obsérvese que éstas quedan por debajo: la imagen está ocupando todo el espacio.

¿Cómo se consigue?

A nivel de la tabla main, bajo la sección Form tenemos la propiedad **Enable Header Row Pattern**. Cuando se habilita, otra propiedad es desplegada: **Header Row Application Bar Class**, donde el desarrollador puede indicar el nombre de una clase **Application Bar** del theme. Aquí se configura cómo se desea que sean application y status bars cuando la primera fila es una HERO image. Obsérvese que en el ejemplo el Status Bar color sigue siendo el PrimaryDark, y es por eso que se ve debajo ese color. El resto es transparente, por lo que no se ve la Application Bar debajo.

En este caso la pantalla no es scrolleable, por lo que la primera fila nunca desaparecerá de pantalla. Pero, ¿qué sucede en caso contrario?



Aquí tenemos una pantalla scrolleable, por lo que cuando el usuario quiere ver lo que hay más abajo en la pantalla y entonces hace el gesto correspondiente, cuando el espacio de la primera fila, –la que ocupaba todo el espacio de la application y la status bar–, empieza a desaparecer, la application y status bar empiezan a aparecer, suavemente.

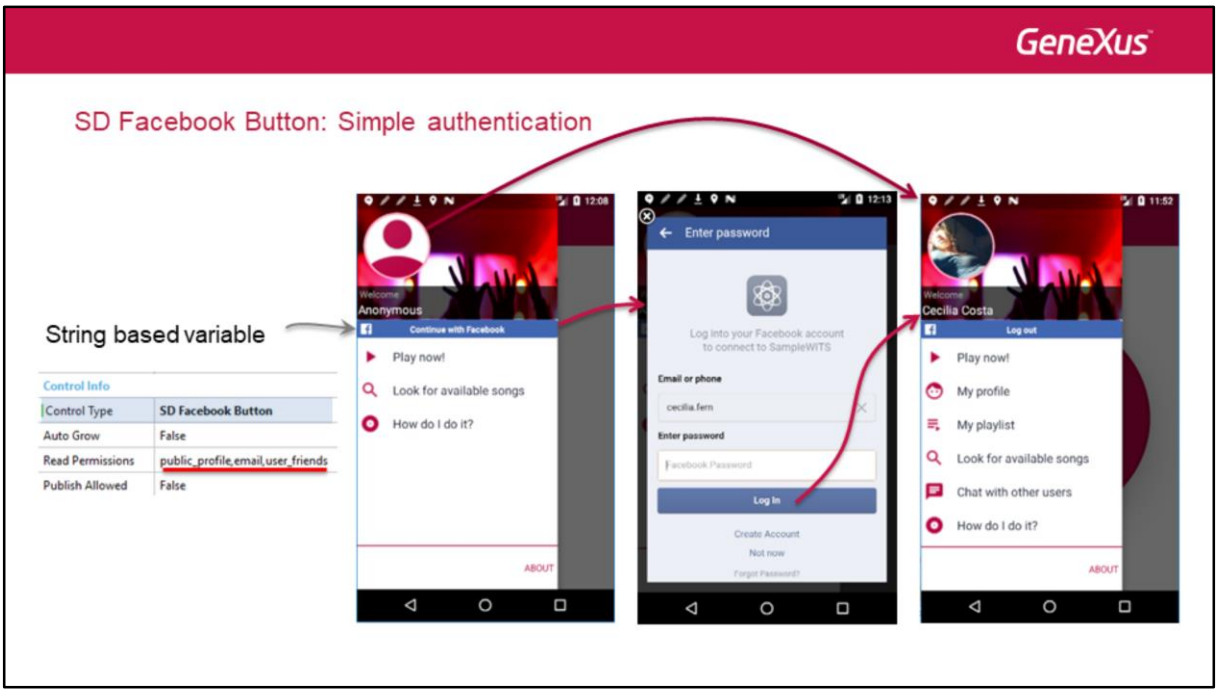
Hay dos Application Bar theme classes en juego: la propia del panel (hacer clic en la zona de la Application Bar del panel y verá que tiene configurada por defecto la propiedad del theme ApplicationBars) y la configurada para el efecto HERO.

Entonces es aplicada la clase de HERO para la application/status bar cuando la primera fila (header row) sigue visible en pantalla y cuando desaparece por el scroll, la application/status bar adopta la clase del propio control Application Bar del layout.

Observemos que al haber configurado como transparentes los colores de Background, Status Bar y Forecolor en la clase de la Header Row, no se ve en absoluto Status/Application bar cuando la primera fila está visible.

Puede ver un ejemplo completo en nuestro wiki.

Facebook



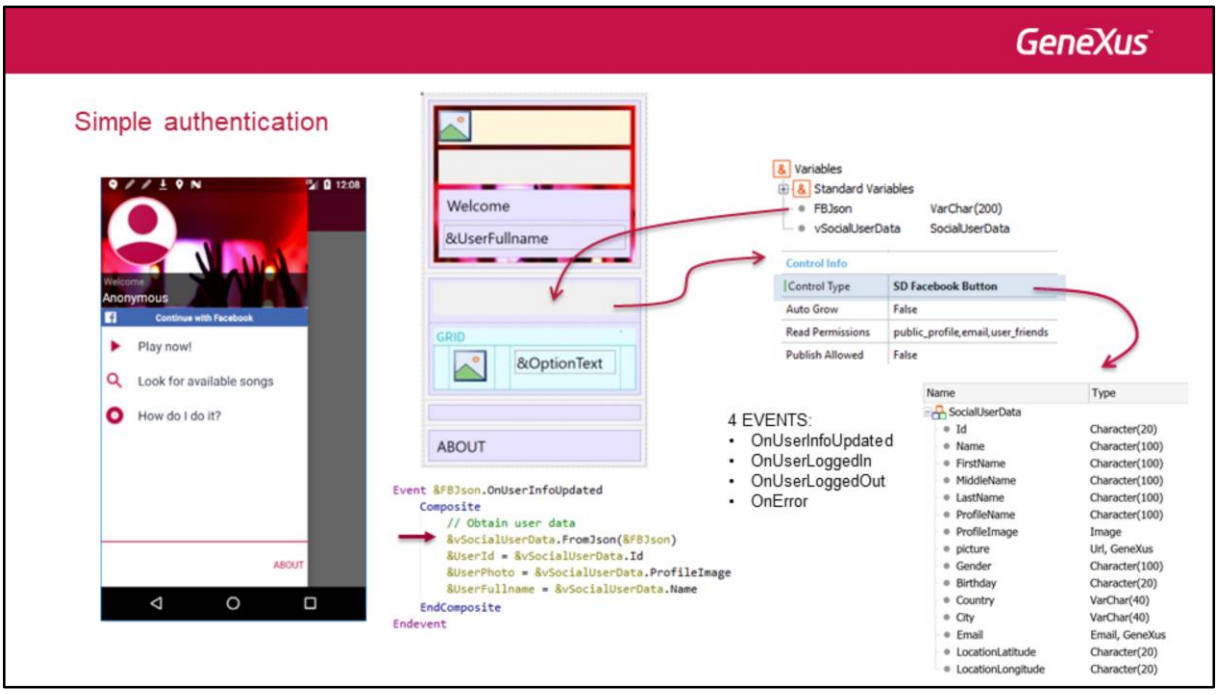
Podemos insertar un Facebook button para recuperar datos del usuario de Facebook y crear un esquema simple de autenticación en nuestra aplicación.

Como podemos ver en la imagen, inicialmente la app muestra un botón de “login” con Facebook. Obsérvese que en el menú, arriba, aparece el usuario anónimo, sin foto.

Una vez que el usuario hace tap sobre el botón, será automáticamente redirigido a Facebook para que ingrese sus credenciales y Facebook le indicará qué clase de información es requerida por la aplicación.

Finalmente, una vez que el usuario final es autenticado y acepta los requerimientos de información de nuestra app, Facebook devuelve esa información. En nuestro ejemplo solamente se pide la información del perfil público (Name, Last Name, Photo, Birthday, etc).

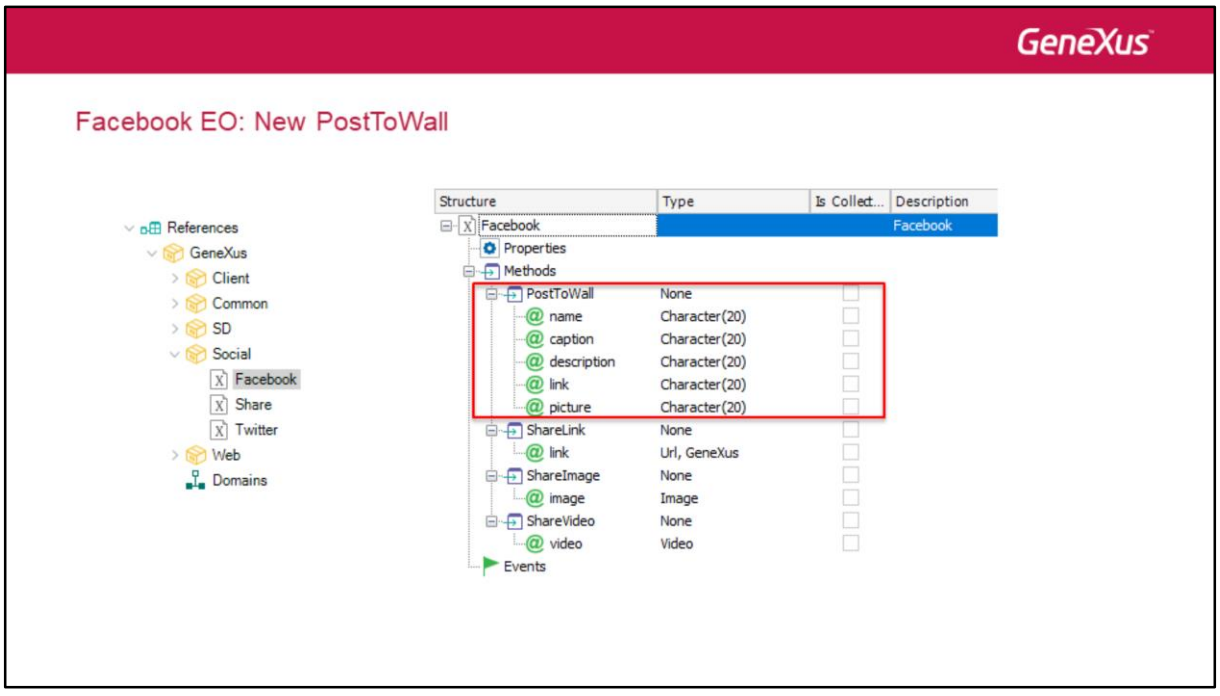
Nota: para poder utilizar esta feature, el desarrollador debe configurar la propiedad **App Id**, bajo el grupo **Facebook** del objeto main.



¿Cómo lo implementamos?

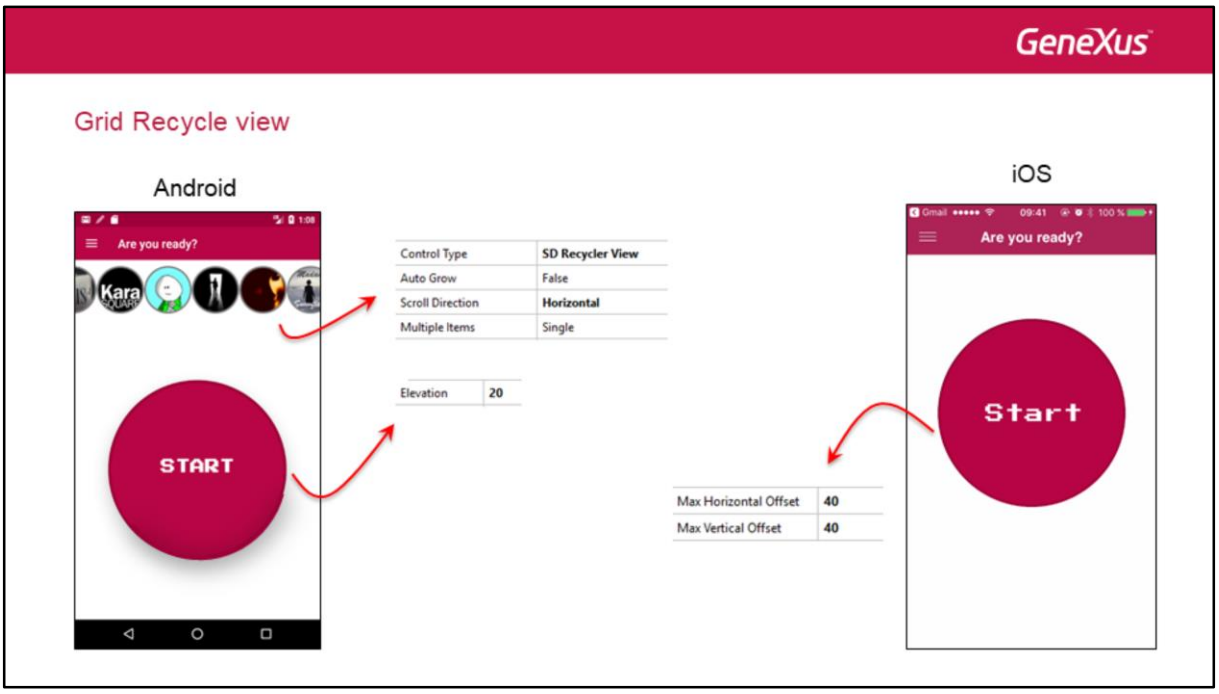
Es muy simple. Debemos seguir los siguientes pasos:

1. Declarar una variable (o atributo) de tipo Character o VarChar, en nuestro caso la que hemos llamado FBJson, para recuperar la información que Facebook nos devolverá en formato Json.
2. Insertar esa variable en el layout del Panel o Work With for Smart Devices donde queramos que aparezca el botón de login de Facebook.
3. Cambiar su propiedad Control Type a **SD Facebook Button**. Con esta configuración se nos abre la propiedad Read Permissions donde podemos indicar qué permisos necesitamos y que el usuario deberá aceptar al loguearse con Facebook. Por defecto, GeneXus pide información pública. Además se crea automáticamente un SDT SocialUserData, para poder devolver la info de Facebook en formato adecuado para GeneXus.
4. Declarar una variable de tipo el SDT SocialUserData para que contenga la estructura del json devuelto por Facebook para poder manipular su información dentro de GeneXus. Basta con que utilicemos el método FromJson para cargar esta variable.
5. Al definir el paso 3, se crean 4 eventos (que aceptan parámetros de input) asociados a la variable/atributo, que capturan los siguientes momentos:
    1. OnUserInfoUpdated: es el primer evento llamado cuando el usuario inicia una nueva sesión de Facebook. Una vez que finaliza y que la conexión ha sido establecida, la información del usuario es actualizada y queda disponible para el desarrollador. Observemos en el ejemplo cómo la hemos utilizado para mostrar la foto y el nombre completo del usuario que se está logueando a nuestro juego.
    2. OnUserLoggedIn: es un evento informativo que el desarrollador puede utilizar para actualizar alguna indicación del estado de la actual sesión de Facebook. En este evento la información del usuario aún no ha sido actualizada.
    3. OnUserLoggedOut: es un evento informativo que indica cuando el usuario se ha deslogueado de la sesión de Facebook.
    4. OnError: devuelve un error cuando algo ha ido mal durante el proceso de autenticación de Facebook.



A partir del upgrade 4 aparece el nuevo método PostToWall en el external object (EO) Facebook.

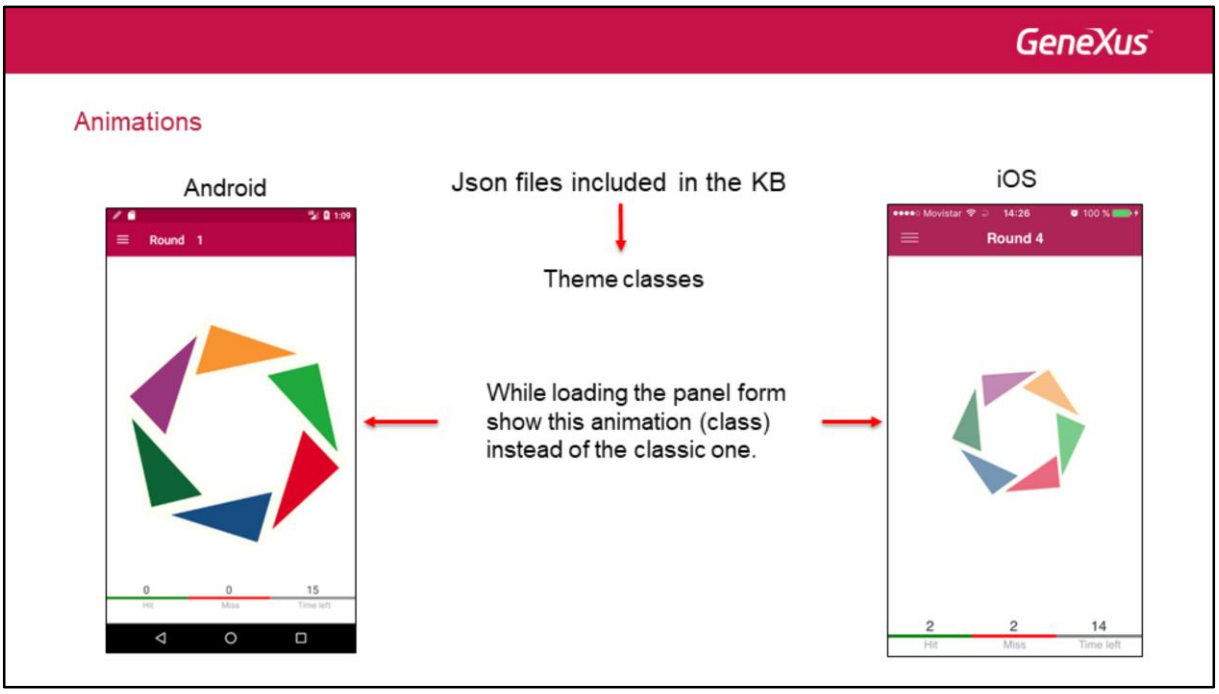
Grid Recycle View



Cuando vimos la pantalla de inicio del juego, vimos ese carrusel que nos iba mostrando los artistas. No se trata de un horizontal grid, que solamente nos permite mover en bloque, por página. Acá se dan movimientos que parecen continuos.

Esto se consigue con el nuevo control type de los grids, por ahora solo para Android, SD Recycler View. Ya de paso podemos ver que el botón de Start tiene asociada una clase con la propiedad Elevation configurada en 20 y por eso vemos ese efecto de profundidad. Efecto de profundidad que como ya mencionamos, en iOS se consigue con las propiedades que vemos a la derecha, Motion effects, que utilizan el giroscopio del dispositivo.

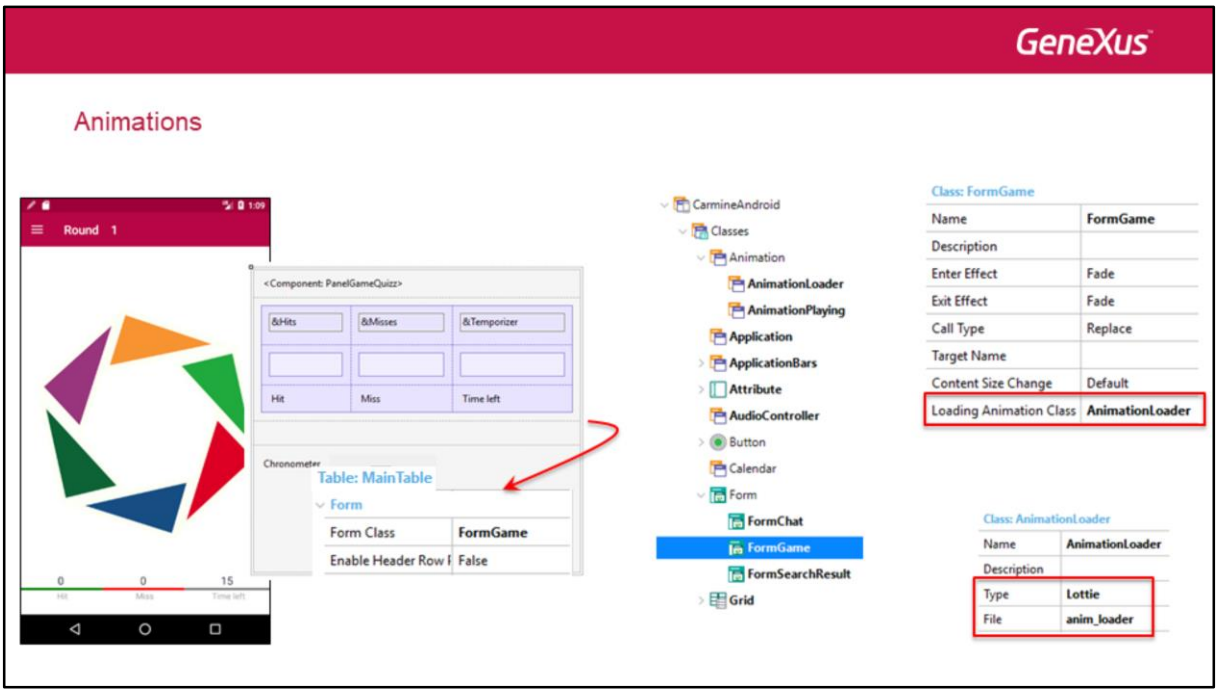
Animations



A partir del upgrade 8 pueden integrarse animaciones, específicamente animaciones Lottie (una library creada por Airbnb, que provee una api Json para integrar las animaciones en aplicaciones mobile). Con esos archivos json podemos incluir animaciones en la KB.
Hay muchos repositorios de animaciones de los que pueden descargarse en formato Json.

Habrá que incluirlos en la KB y crear clases para cada uno de ellos.

En el ejemplo, estamos utilizando una animación para personalizar el circulito que por defecto aparece cada vez que se está haciendo un Loading de una página o grid. Para ello utilizaremos



Vemos el form del panel que carga la pantalla con la ronda actual del juego. Si observamos las propiedades de la MainTable del Layout, veremos que a la Form Class le hemos asociado una clase FormGame que creamos en el theme, y a la que a su vez le hemos configurado la propiedad “Loading Animation Class” con el valor de una Animation class que también creamos.

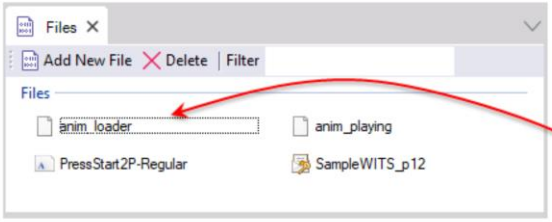
A esa animación, a la que llamamos AnimationLoader, la definimos del tipo Lottie, y tuvimos que indicarle el nombre del archivo json de la animación, que previamente hubimos de insertar en la KB.

En la siguiente página vemos los pasos de forma más ordenada.

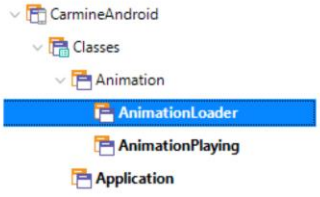
GeneXus

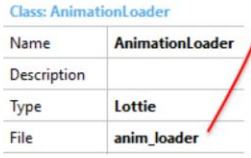
Animations: how to incorporate to the KB

1. Add the Lottie Json files as File objects in the KB



2. Create Animation classes in the Theme



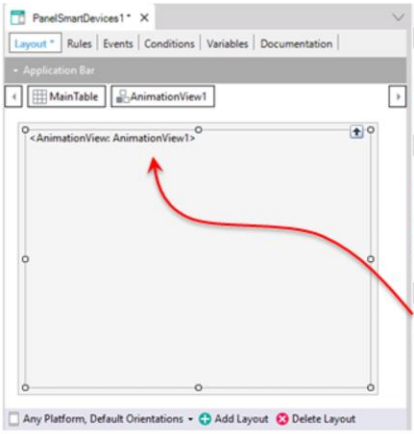


GeneXus

Animations

Animation View

3. Use it in some theme class property where an animation is expected (eg: **Loading animation class** of Form, Grid and Matrix theme classes) or use it throught the **Animation View control**:



AnimationView methods:

- SetAnimation(AnimationClass, loop)
- SetProgress(progress)
- Play()
- Pause()

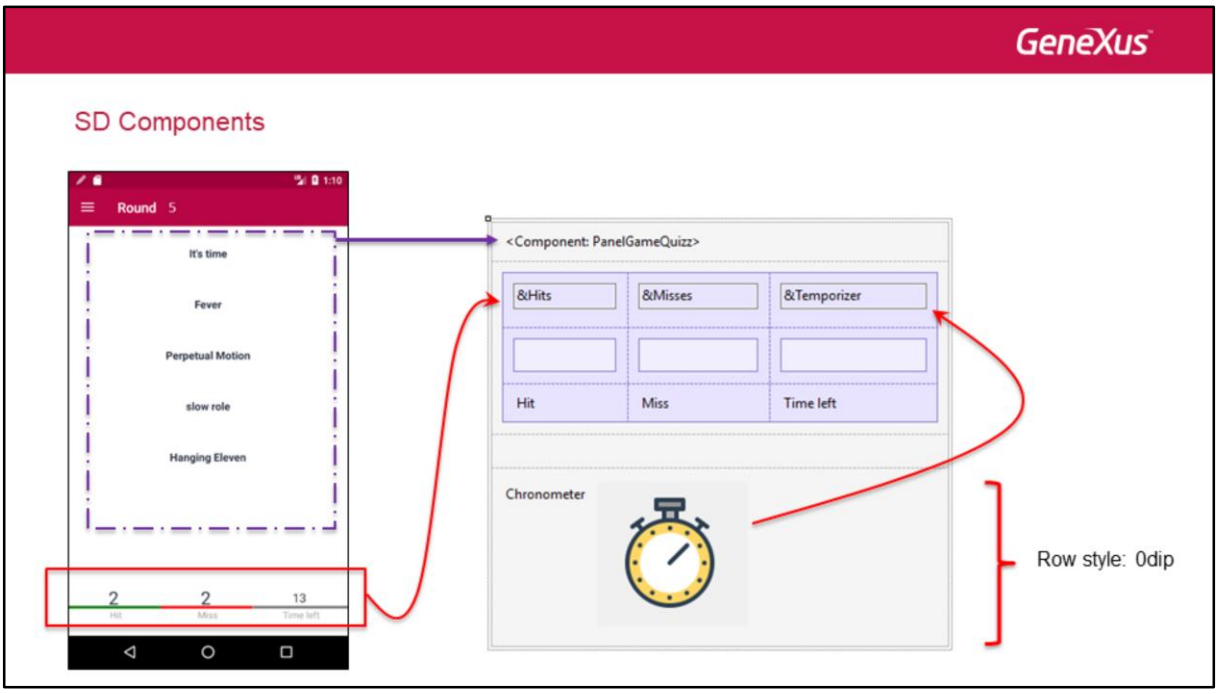
eg.

```
Event ClientStart
    composite
        AnimationView1.SetAnimation(ThemeClass:AnimationPlaying, true)
        AnimationView1.Play()
    endcomposite
endevent
```

Existe un nuevo control que puede insertarse en un layout, Animation View, con cuatro propiedades.

En la primera seteamos cuál será la clase de tipo Animation que será cargada en el control y si queremos que la reproduzca en loop o no. SetProgress permite indicar un número entre 0 y 100 para que desde allí arranque la animación. Las demás son obvias.

SD Components and Global Events



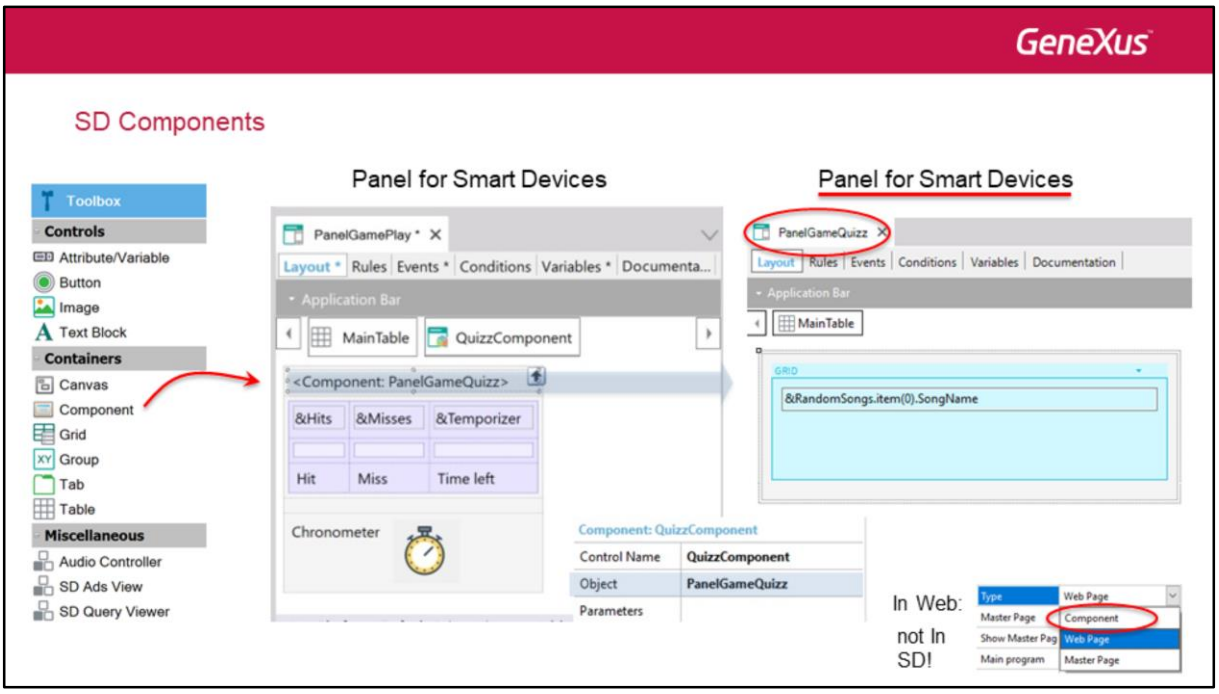
Aquí tenemos el panel PanelGamePlay que para cada ronda muestra las cinco canciones para que el usuario elija la que está sonando, y abajo se le muestra la cantidad de aciertos que tuvo hasta el momento en las rondas anteriores, de desaciertos, y el tiempo que le resta para adivinar la canción de la ronda actual.

Si vamos a observar el layout del panel, veremos que consta de un control nuevo, componente, que es como los componentes web, con algunas mínimas diferencias.

Básicamente, el concepto de Componente involucra algún tipo de “ventana” en un panel, a través de la cuál puede verse otro panel. En otras palabras, esto permite que el desarrollador pueda diseñar un panel que pueda ser ejecutado independientemente y pueda ser embebido en otro panel como componente de él.

Deberemos mantener sincronizados panel padre y panel hijo, puesto que cuando el usuario haga un tap en el componente (el panel que muestra las cinco canciones de la ronda actual), deberán actualizarse los contadores Hit y Miss, y además al cargarse una nueva ronda deberá resetearse el temporizador. De la misma manera, cuando se termine el tiempo de una ronda, deberá darse por finalizada y perdida.

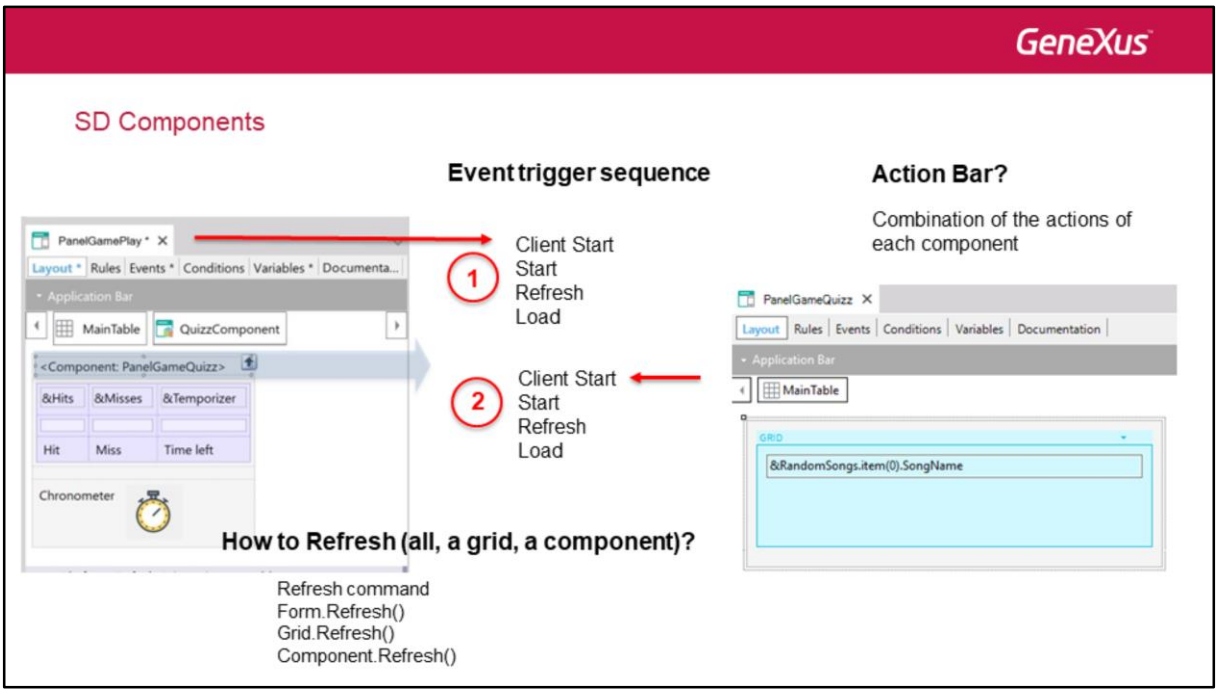
Para tener un timer necesitamos un nuevo tipo de control conocido como Chronometer, que deberá estar en la pantalla, aunque no quereamos visualizarlo. Como no puede utilizarse para él la propiedad Visible en 0, alcanzará con colocarlo en una fila con 0dips.



Observemos que se ha incorporado a la Toolbox el control Component. Una vez arrastrado al layout del Panel del que se trate (por supuesto puede ser también el layout de un Work With for Smart Devices), accediendo a sus propiedades se encontrará la de nombre **Object**, que es la que permite especificar qué objeto se cargará allí como componente. En nuestro caso cargaremos el Panel for Smart Devices PanelGameQuizz, que es el que carga las cinco canciones de la ronda actual. Nuestro PanelGameQuizz no tiene parámetros, pero si los tuviera, es en la propiedad **Parameters** del control Component donde éstos se especifican.

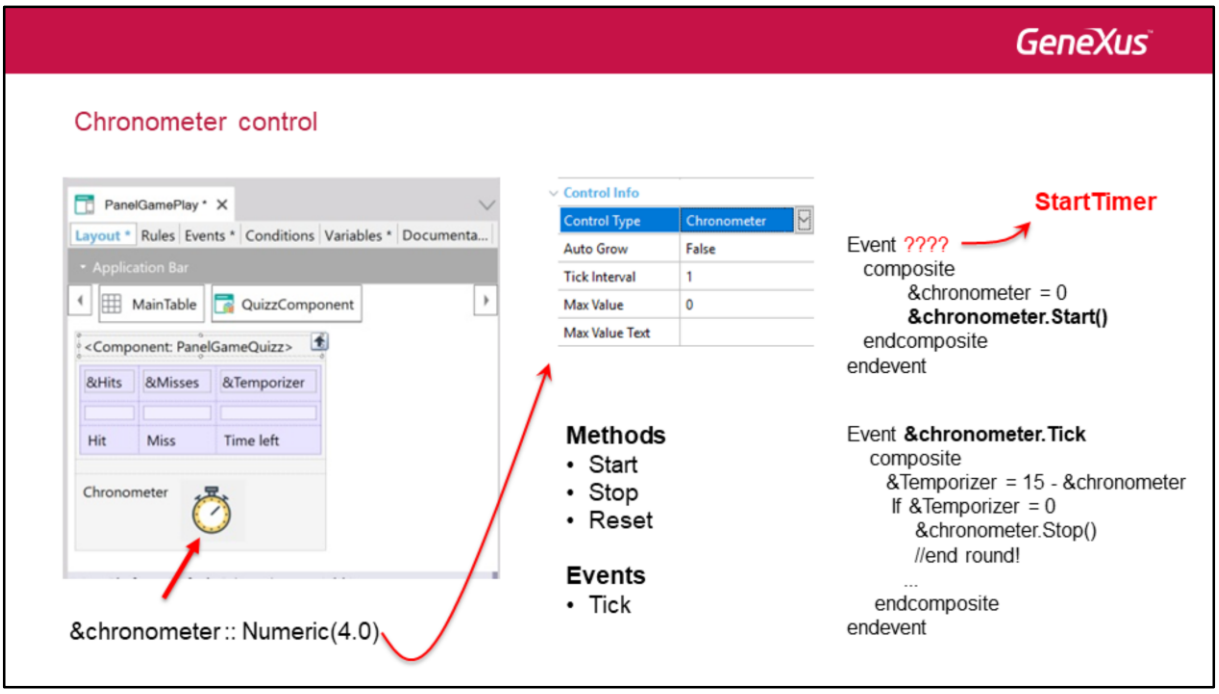
Estas propiedades, a diferencia de en Web, sólo pueden especificarse aquí, es decir, en el diseño. No pueden especificarse dinámicamente, en runtime.

Por otro lado, si bien en Web un panel componente debía especificarse explícitamente de ese tipo, en SD no. Cualquier Panel for Smart Devices, e incluso Work with for Smart Devices podrá ser utilizado como componente.



El orden de ejecución de los eventos cuando un panel tiene uno o más componentes es el mismo que el de la sección Detail de un Work with que tiene una o varias secciones.

Primero ejecuta los eventos asociados con el panel contenedor y luego los eventos de cada componente embebido, secuencialmente (uno por uno, en el orden en el que aparezcan en el layout).



El control Chronometer nos da la posibilidad de ejecutar un evento luego de un cierto tiempo o simplemente mostrar un cronómetro en la pantalla. Basta con definir una variable o atributo numérico y cambiarle el tipo de control a Chronometer. Es válido tanto para Web como para SD.

Una vez hecho esto, se habilitan tres propiedades:

- Tick interval: indica la frecuencia en segundos en que se disparará el evento **Tick**.
- Max value: el máximo valor que el cronómetro puede tomar
- Max value text: el texto asociado a al variable cuando alcanza el valor máximo.

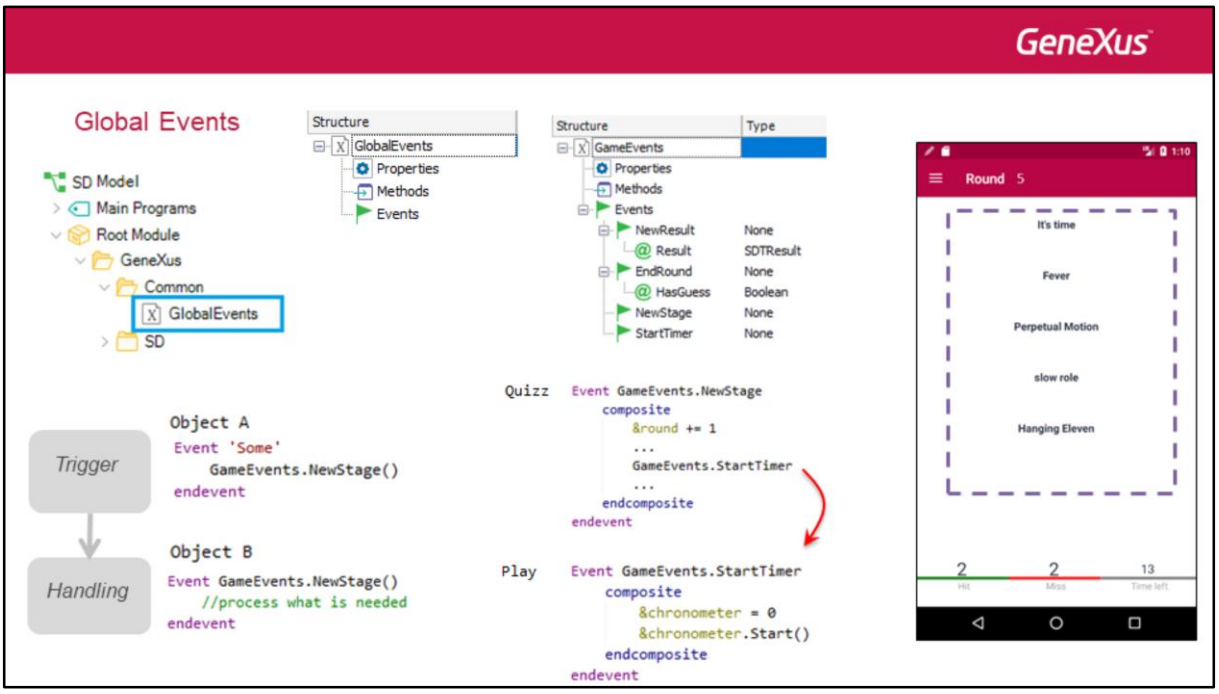
Además se habilitan tres métodos:

- Start: Inicia el cronómetro. Empezará por el valor (en segundos) contenido en el atributo/variable asociado al control.
- Stop: Detiene el cronómetro.
- Reset: Coloca en 0 el valor del cronómetro.

Y además y bien importante, se habilita el evento Tick que será ejecutado cada vez que el valor indicado en la propiedad Tick interval haya trascurrido.

En nuestro ejemplo, cuando inicia una nueva ronda del juego hay que iniciar el timer, es decir, el cronómetro debe arrrancar en 0 y al llegar a 15 detenerse porque se terminó el tiempo para esa partida.

A continuación veremos en qué evento colocar el Start del cronómetro. Será en un evento global que definiremos. Lo llamaremos StartTimer.



Como vimos cuando estudiamos lo nuevo en Web, en GeneXus 15 se ha incorporado un external object especial llamado **GlobalEvents**, que a diferencia de la mayoría de los external objects predefinidos, no viene en el módulo GeneXus, bajo References, sino dentro del folder GeneXus. La razón: será modificado por el desarrollador. Para eso fue hecho.
No obstante, podemos salvarlo con otro nombre, y tener un external object de nombre específico, por ejemplo GameEvents.

Aquí el desarrollador podrá definir eventos que lograrán comunicar a paneles, por ejemplo para sincronizar acciones entre sí.
En nuestro ejemplo hemos definido cuatro eventos globales para poder sincronizar al panel PanelGamePlay con su hijo PanelGameQuizz.

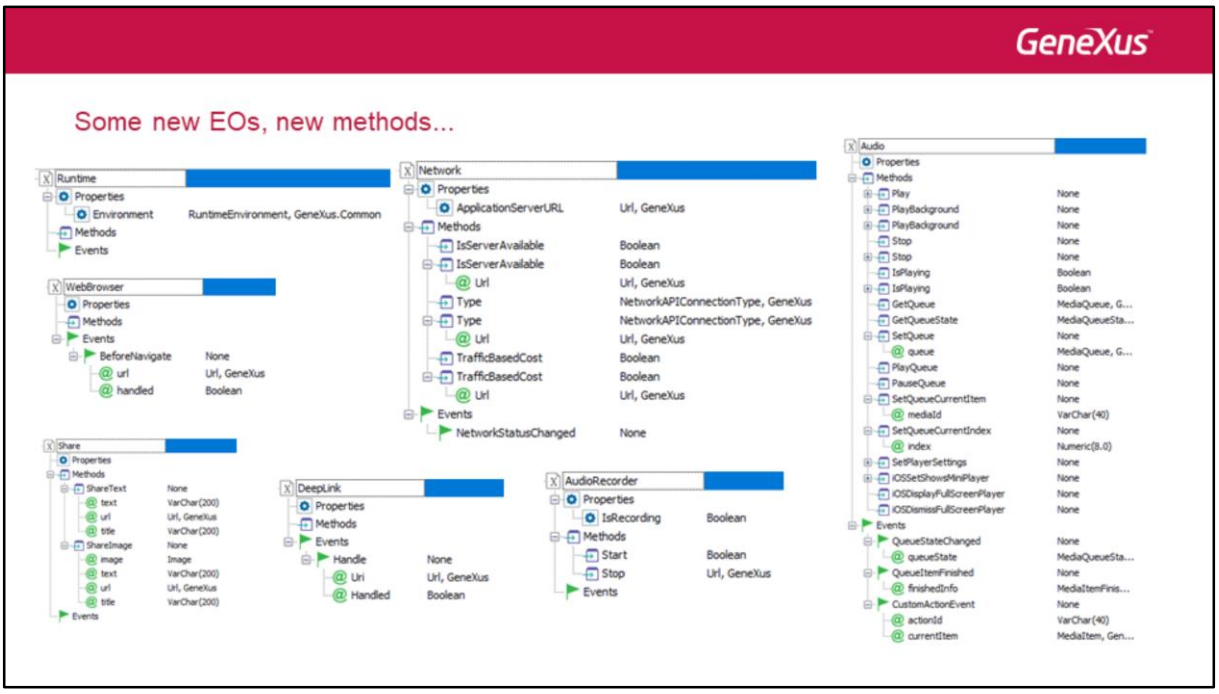
Alguno o algunos de los paneles dispararán el evento, que será escuchado por el o los paneles que lo tengan definido como evento en su sección Events. Estos eventos pueden o no tener parámetros.

De hecho, un mismo panel puede disparar un evento y manejarlo (¡Object A y Object B pueden ser el mismo!). Así, desde un evento de un panel puede ejecutarse otro evento del mismo panel, invocándolo.

Este es un uso simple de la potencia de los GlobalEvents, pero también pueden ser utilizados en paneles independientes (no necesariamente para sincronizar componentes y el panel padre).

A la derecha vemos un ejemplo: el panel componente PanelGameQuizz en un evento global NewStage que será disparado desde el contenedor PanelGamePlay (aquí no lo mostramos) cuando hay que iniciar una nueva ronda de canciones, incrementa el número de ronda en 1 y dispara la inicialización del timer, evento global que es manejado por el PanelGamePlay, que es el que tiene el cronómetro y muestra el tiempo restante (a partir del evento Tick que vimos antes).

More about external objects



Algunos de estos external objects son nuevos, otros tienen nuevos métodos. Aquí simplemente listamos algunas de las novedades, e iremos viendo en más detalle algunos casos.

Runtime: para determinar dónde se ha ejecutado alguna acción (en el server, o dispositivo –o browser-)

WebBrowser: para manejar links en web pages embebidas.

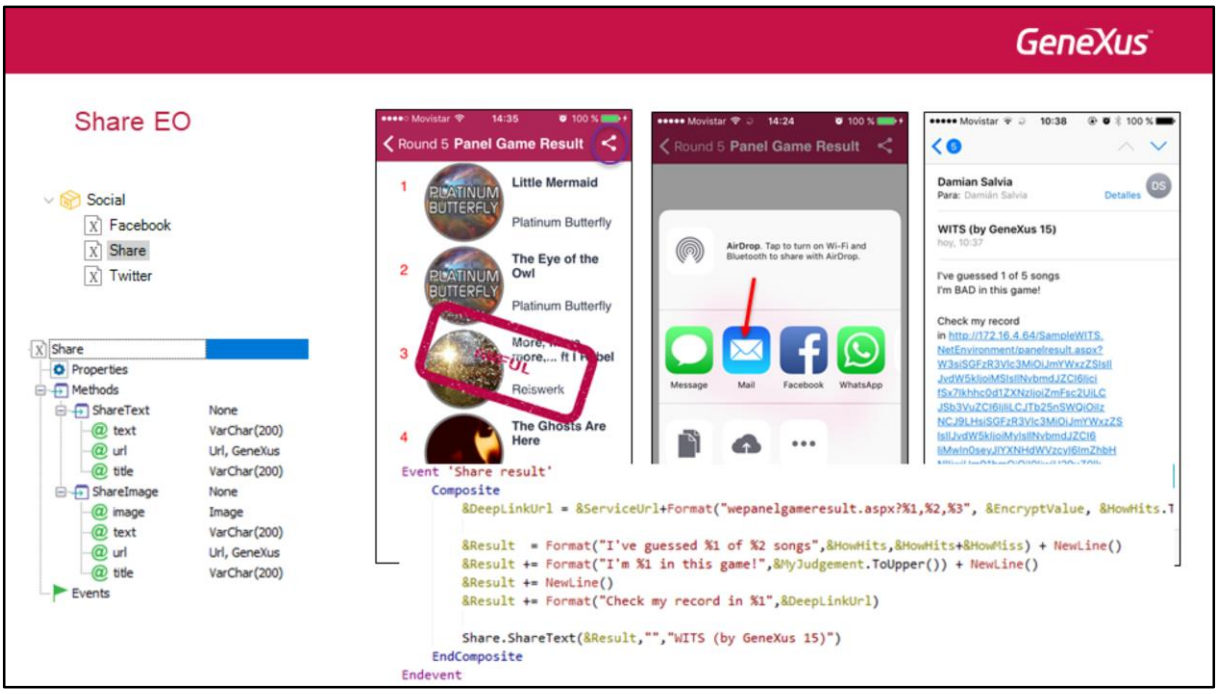
DeepLink: para linkear web y mobile apps, abriendo el link en la app nativa si está instalada, y en el navegador mobile si no lo está, de forma más compleja que la automática que ya ofrece GeneXus15 y que veremos.

Network: nuevos métodos.

Etc.

A continuación veremos un ejemplo del external object Share y la forma automática de Deep linking, que no necesitará utilizar el EO DeepLink.

Share EO and Deep Linking



Para poder compartir contenido a través de otras apps instaladas en el dispositivo contamos con el external object Share con los dos métodos que vemos:

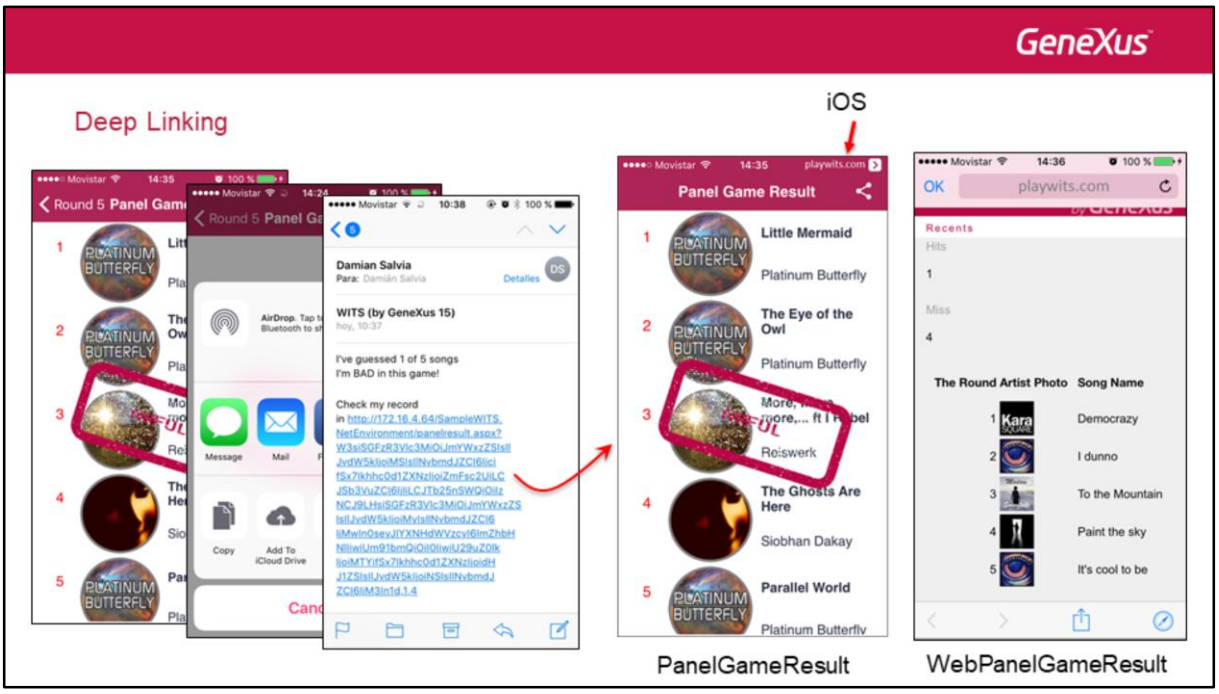
- Uno que permite compartir texto
- Otro que permite compartir una imagen

Además es posible compartir información extra, tal como una url y un título.

En el ejemplo, cuando termina la última ronda del juego se invoca al panel que vemos a la izquierda, que muestra los resultados y ofrece un botón de compartir en la application bar. Cuando el usuario lo elige, se le abre la lista de apps instaladas que permiten compartir textos e imágenes, para que el usuario elija a través de cuál desea compartir sus resultados.

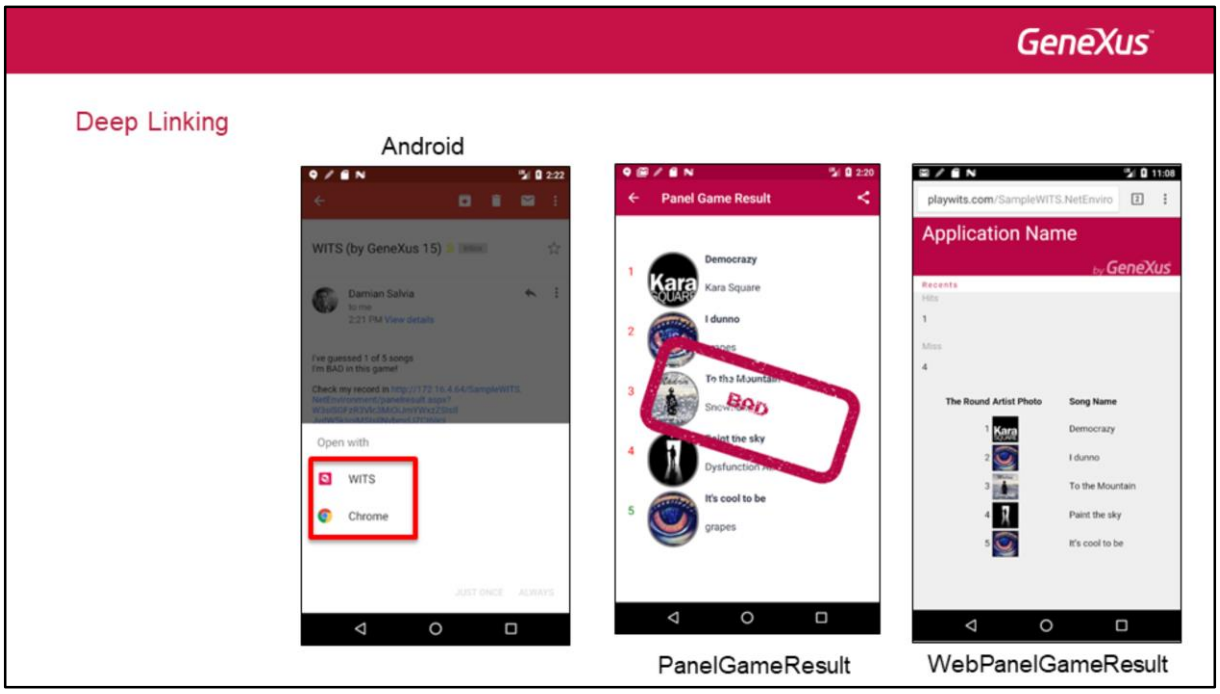
Por ejemplo, supongamos que elige hacerlo por mail. Entonces se abre esa app, con el texto especificado en el evento, en la variable &Result en su cuerpo, y el tercer parámetro, el título en el subject.

Observemos que dentro de la variable &Result hemos cargado un link, que dirige a un web panel análogo al panel de resultados. Pero es un link particular. Lo veremos en la siguiente página.

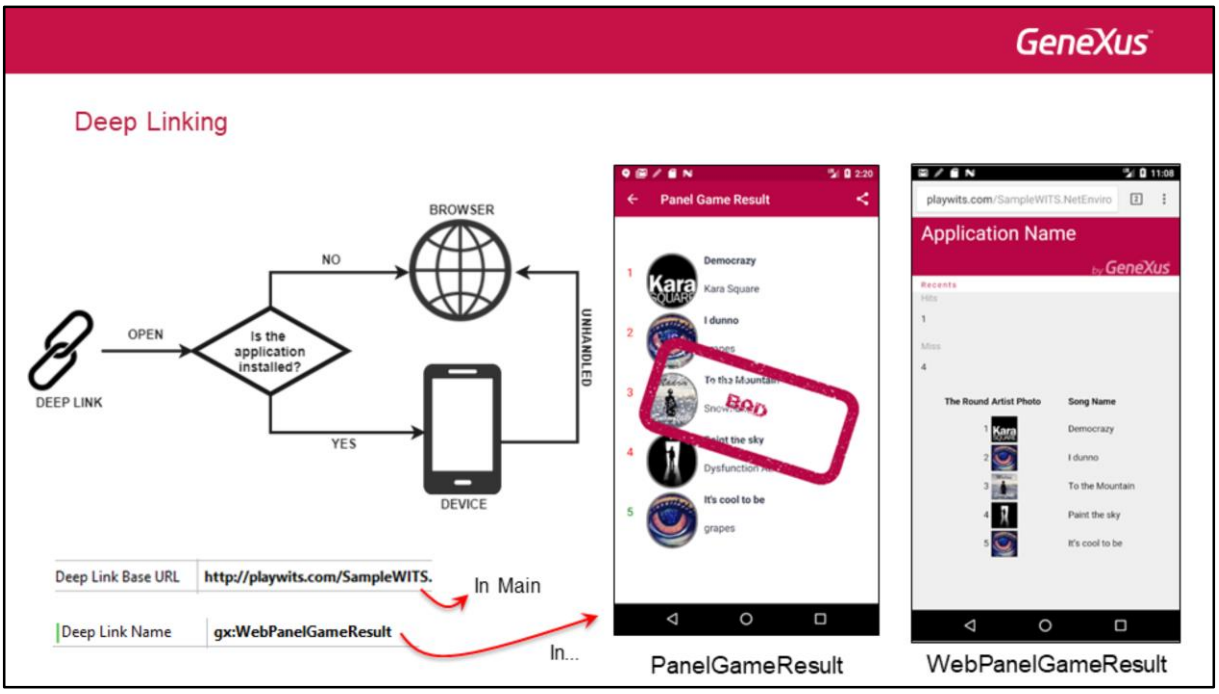


En el marco de iOS, si el usuario hace clic en el link y el dispositivo tiene instalada la app WITS (nuestra app) entonces se le abrirá directamente el panel for Smart Devices de resultados, con el resultado, en vez de el web panel del link. De todas formas se ofrece arriba la posibilidad de abrir el web panel con el navegador.

Si el dispositivo no tiene instalada nuestra app WITS, entonces de entrada se abrirá el navegador con el web panel de resultados.



En Android, en cambio, cuando se hace clic sobre el link, si la app está instalada en el dispositivo se ofrece abrirla con ella, o con el navegador, y dependiendo de la elección del usuario se abrirá el panel en la app o el web panel en el navegador.



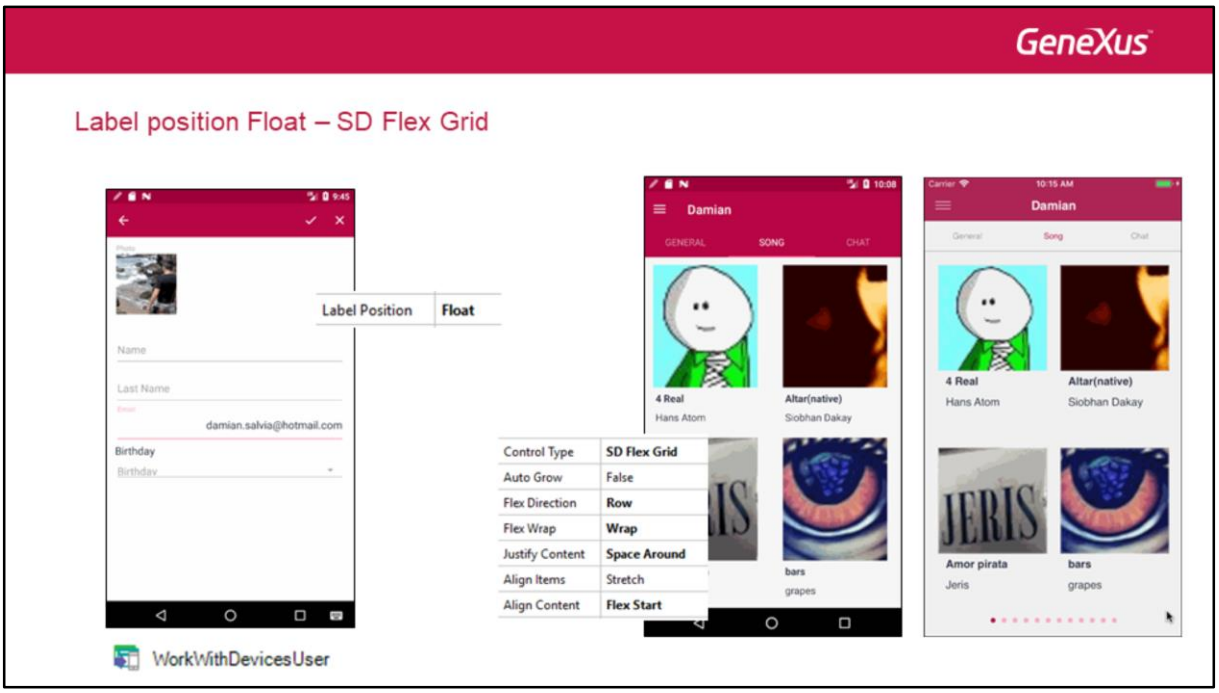
Este comportamiento se consigue con dos propiedades.

Al main SD se le configura la propiedad **Deep Link Base URL**. Cuando se instala la app en el dispositivo, el dispositivo “escucha” todo link que tenga esa (o esas, pues pueden configurarse varias) Base URL, porque desde cualquier otra app (como el programa de correo) se puede querer abrir un objeto web que sin embargo tenga un análogo nativo.

En GeneXus se implementa un web panel y además un SD panel (o Work With for Smart Devices) que será el análogo del web pero para la app nativa. A ese SD object se le configura la propiedad **Deep Link Name** con el nombre del web panel análogo que se deberá abrir en caso de link (por eso se llama Deep link).

Entonces, cuando un link en el dispositivo tiene la Base URL y es del recurso que aparece como Deep Link Name en algún SD panel, es que se ofrecen ambas alternativas.

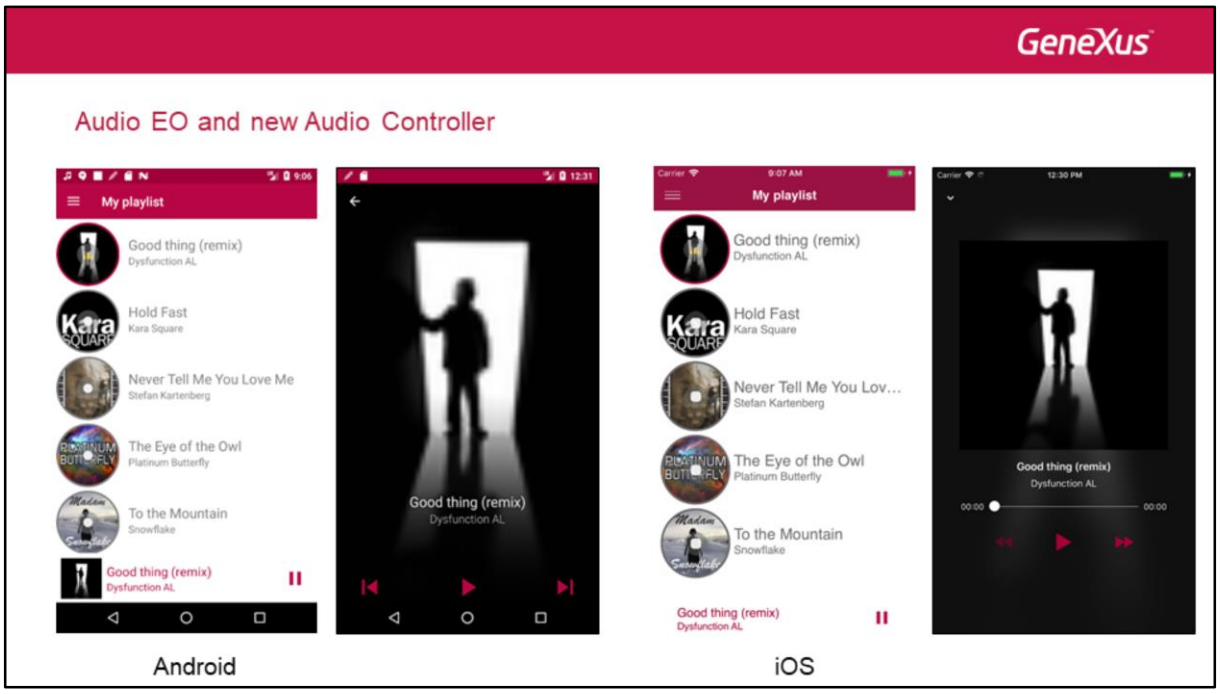
More UI features



Aquí vemos una nueva Label Position con el comportamiento que se aprecia. Cuando el usuario va a digitar en el campo, la etiqueta flota arriba.

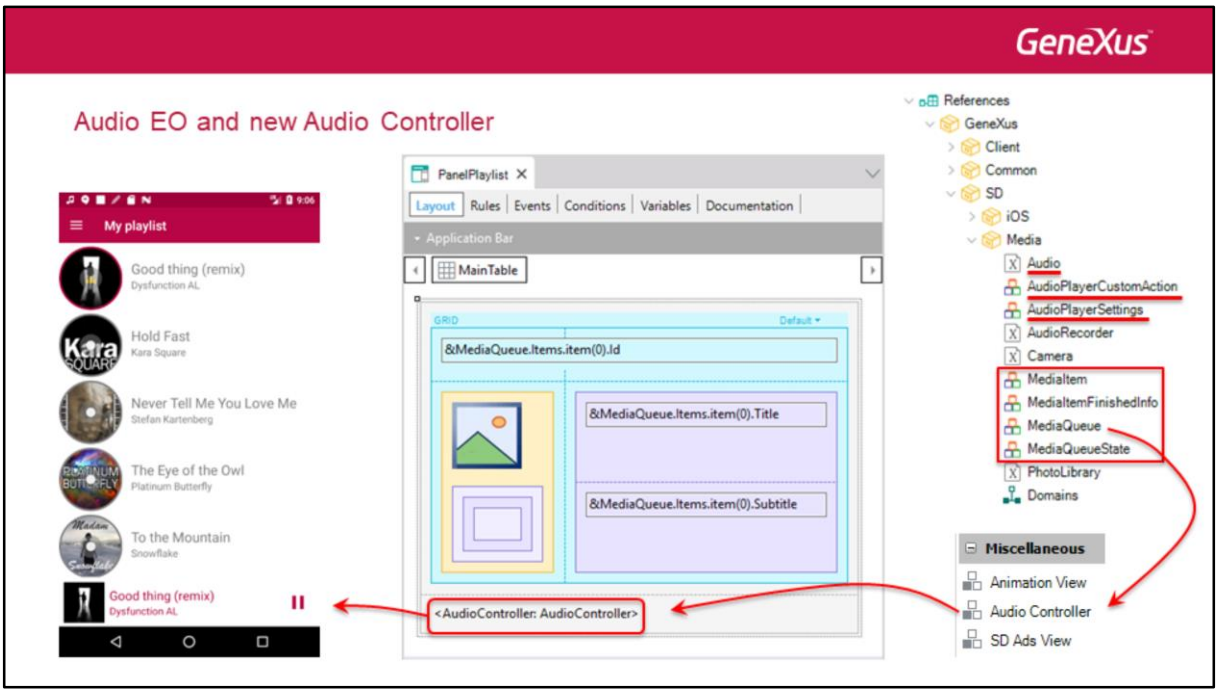
A la derecha vemos un nuevo tipo de control para grids, que se parece al Horizontal Grid pero no es. Observemos en la implementación de la derecha, con horizontal grid, que nos encontramos en un panel que tiene tres tabs. En el segundo es que se encuentra este horizontal grid que muestra cuatro canciones por página. Para pasar a la siguiente página hay que mover la pantalla para la derecha o izquierda con el dedo.
Pero si quisiéramos que ese comportamiento sea el default en el que nos podemos mover de tabs, este grid no nos sirve. Contamos con un nuevo tipo de grid, el SD Flex Grid, que permitirá scrollear como se ve en el video, y conservar, entonces los desplazamientos a derecha e izquierda para movernos de tab.

Audio



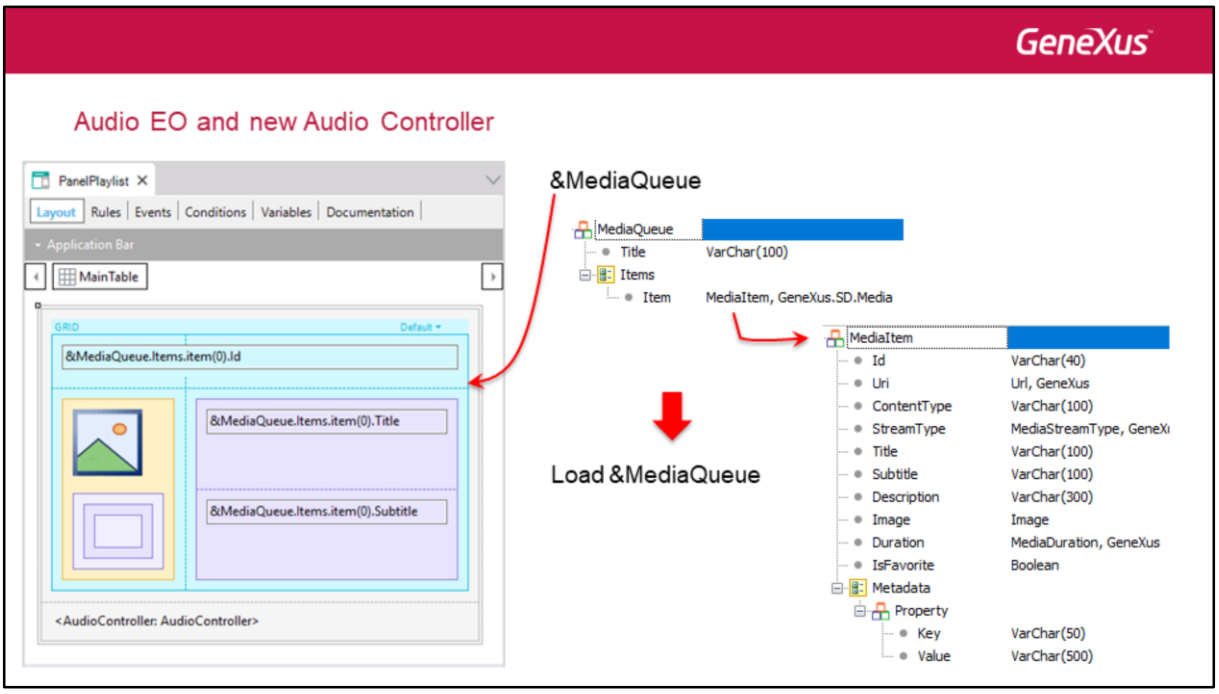
A partir de GeneXus 15 podemos manejar playlists e incorporar un audio player que puede mostrar la lista de reproducción en modo “mini” o en modo “fullScreen”, como podemos ver en las imágenes.

Aquí tenemos el panel Playlist, que lo que hace es mostrar todas las canciones que el usuario logueado ha adivinado hasta el momento, y abajo vemos un reproductor de audio que permite escucharlas.



El reproductor de playlists está implementado con el nuevo control AudioController. ¿Cómo sabe qué playlist reproducir, de dónde sacar los datos?

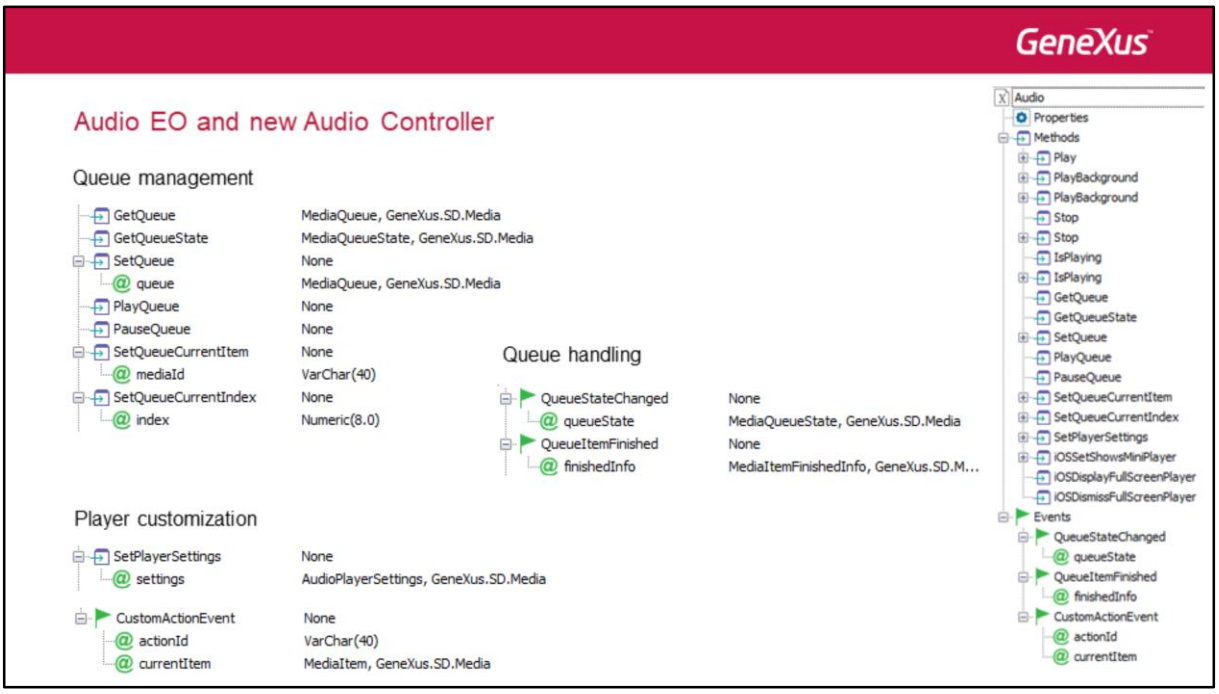
Si observamos el contenido del módulo Media, vemos que relacionados con la api Audio tenemos todos los SDTs que marcamos arriba. Uno de ellos, MediaQueue es el que permite almacenar los datos necesarios para la cola de reproducción que será manejada por el AudioController.



Basta entonces con cargar una variable de tipo el SDT predefinido MediaQueue, que tiene una colección de ítems del tipo MediaItem, que almacena la info de una canción. En nuestro caso lo cargaremos de la base de datos, a partir de una transacción User que contiene un subnivel Song con las canciones adivinadas.

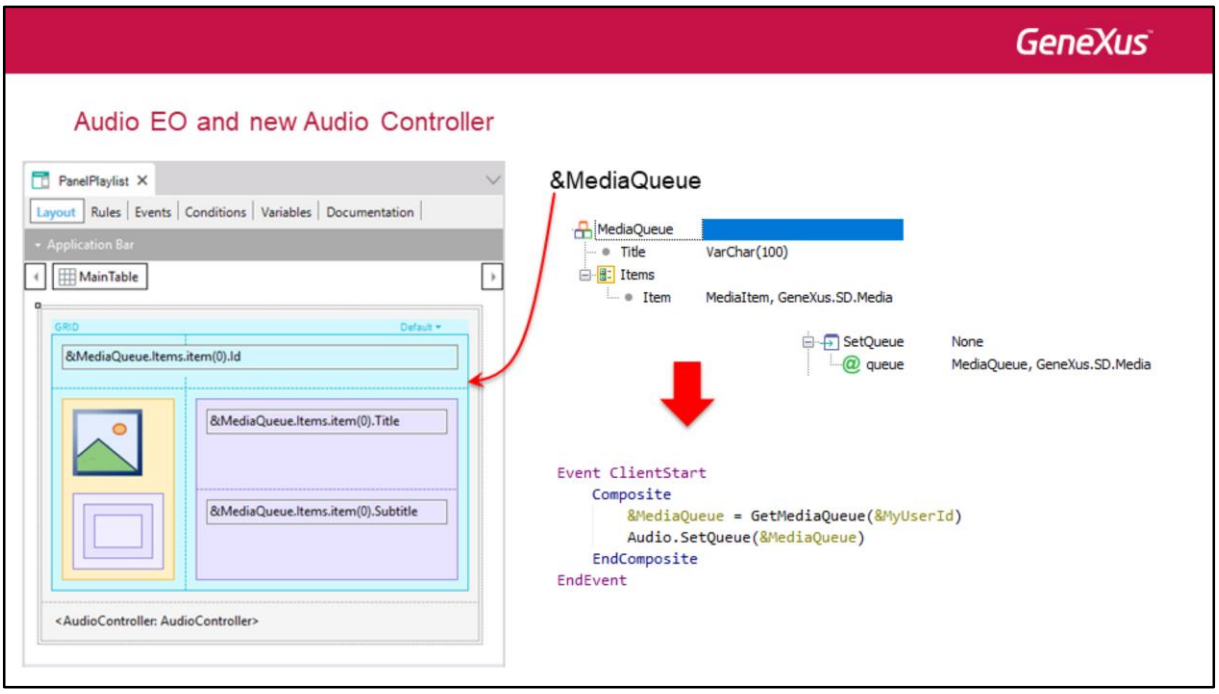
En el panel que vemos estamos utilizando la variable &MediaQueue tanto para mostrar su colección de ítems en un grid, como para reproducirlas con el audiocontroller.

Para poder manejar la repoducción, contamos con la api Audio.



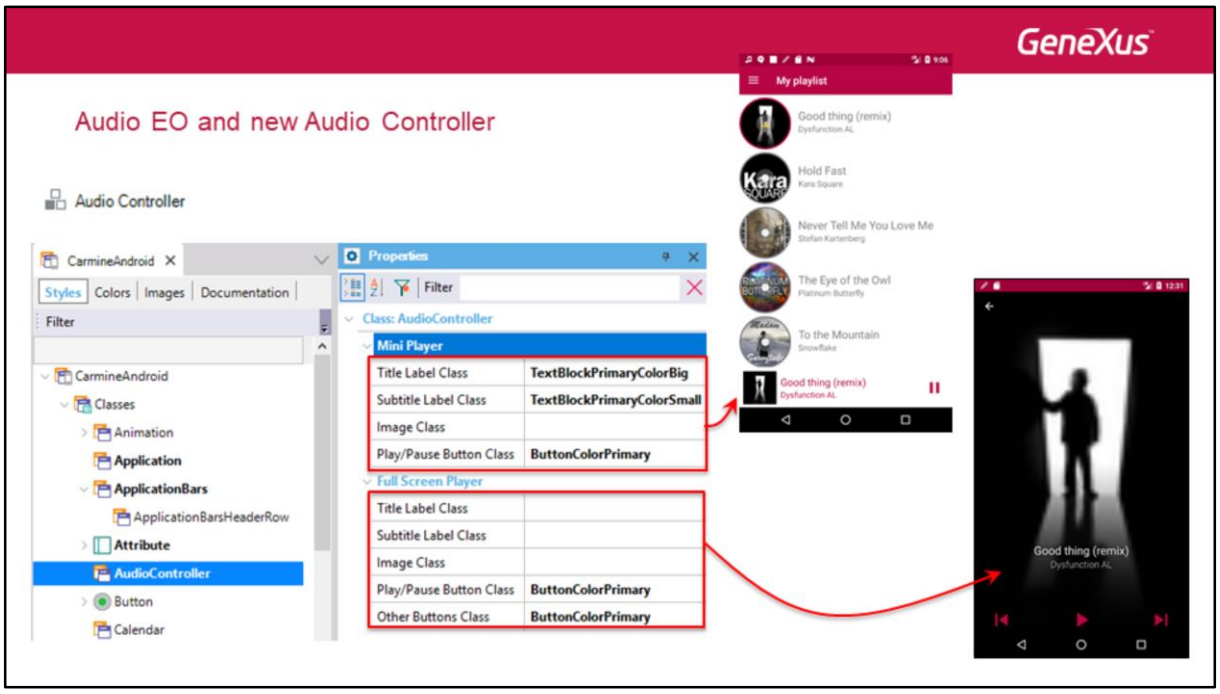
A los métodos que ya tenía la api, vemos que agrega los que son para el manejo de colas de reproducción. Además agrega eventos.

Aquí presentamos separados los que son para la administración de las colas, su manejo y los de personalización del reproductor (la posibilidad de agregar acciones distintas de las predefinidas (stop, pause, etc.).



Volviendo a nuestro ejemplo, si solo vamos a dar la posibilidad de reproducir la cola con el reproductor, entonces alcanzará con programar el evento ClientStart como mostramos.

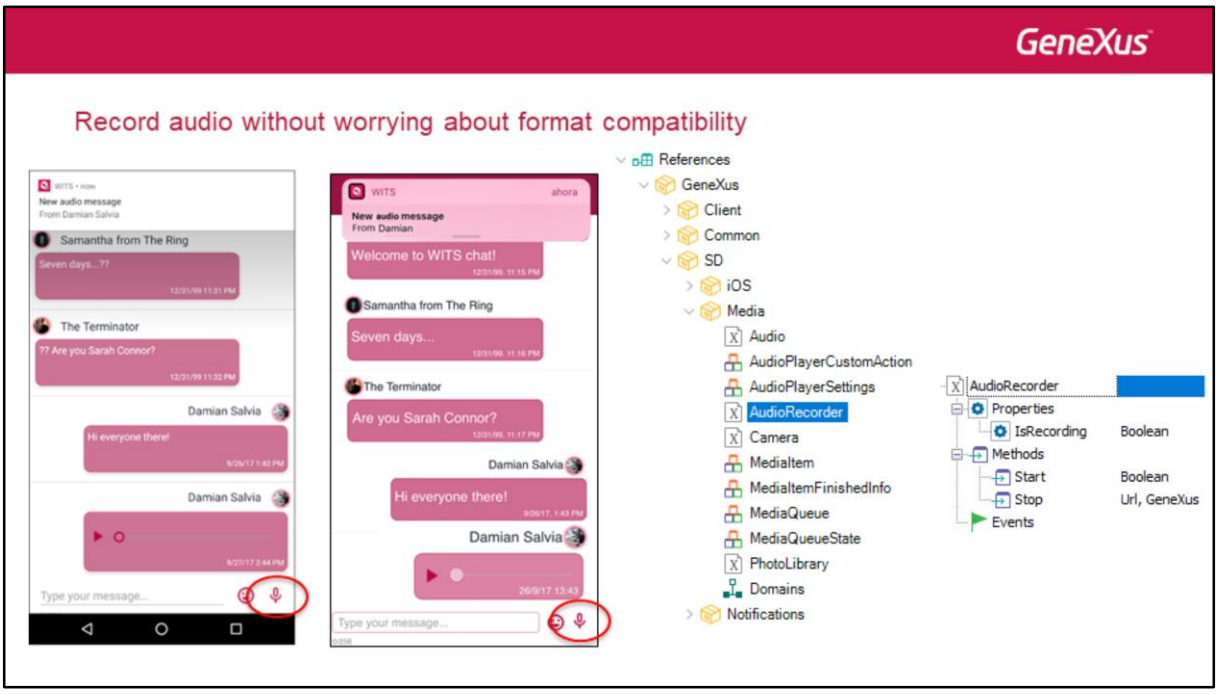
Pero, ¿cómo configuramos el diseño del repoductor?



El reproductor permite configurar en diseño las propiedades que vemos en la clase AudioController del theme.

En nuestro ejemplo, vemos que en modo “mini” se ven en colores rosados la etiqueta del título y del subtítulo, con distintos tamaños de letra, y en modo “full” vemos en blanco título y subtítulo y los botones con los colores rosados de la app.

Audio Recording

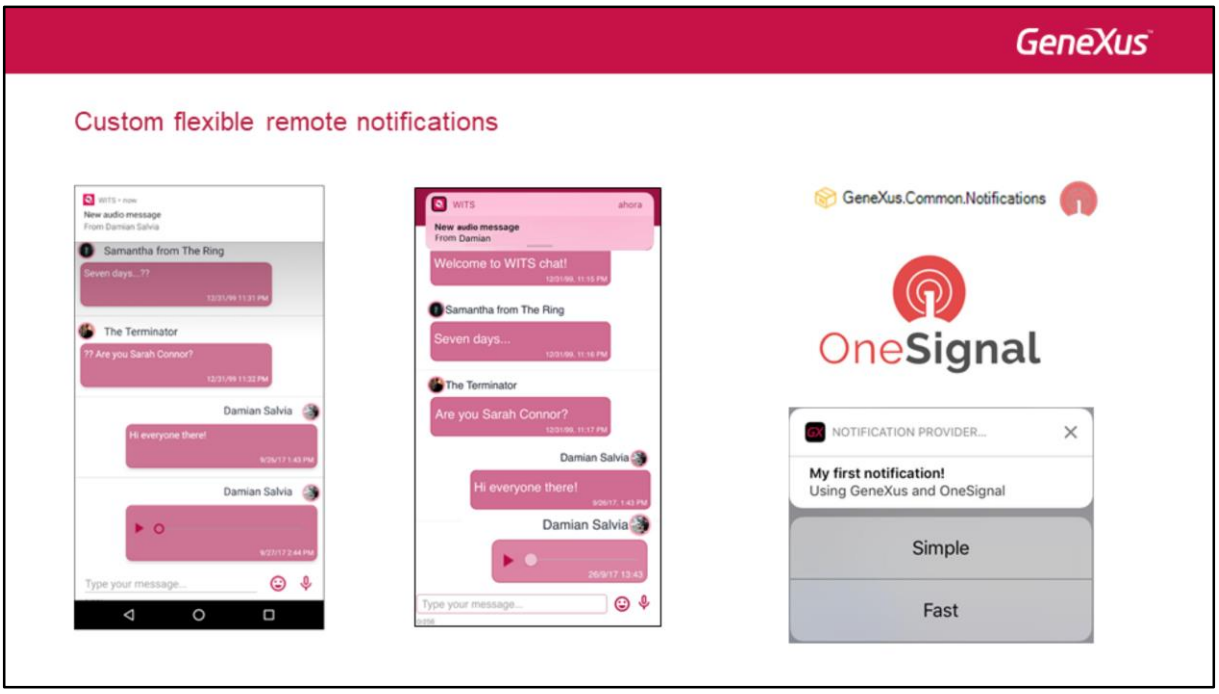


A partir de GeneXus 15 aparece un nuevo external object llamado AudioRecorder, que permite que el audio que se grabe en una plataforma sea reproducido sin problema en la otra (compatibilidad en los audios entre iOS y Android).

- **Start method**, comienza una grabación de audio y retorna el éxito o no de la operación.
- **Stop method**, finaliza la actual sesión de grabación y retorna el path donde el archivo de audio fue grabado.

Este mecanismo simple permite al desarrollador grabar audio en un procesamiento batch (un audio atrás de otro).

Push notifications



Se provee un módulo para enviar notificaciones utilizando OneSignal. ¿Por qué OneSignal?

OneSignal es un proveedor de notificaciones que permite el envio masivo de mensajes a los dispositivos registrados. Además de ser robusto, es altamente personalizable.

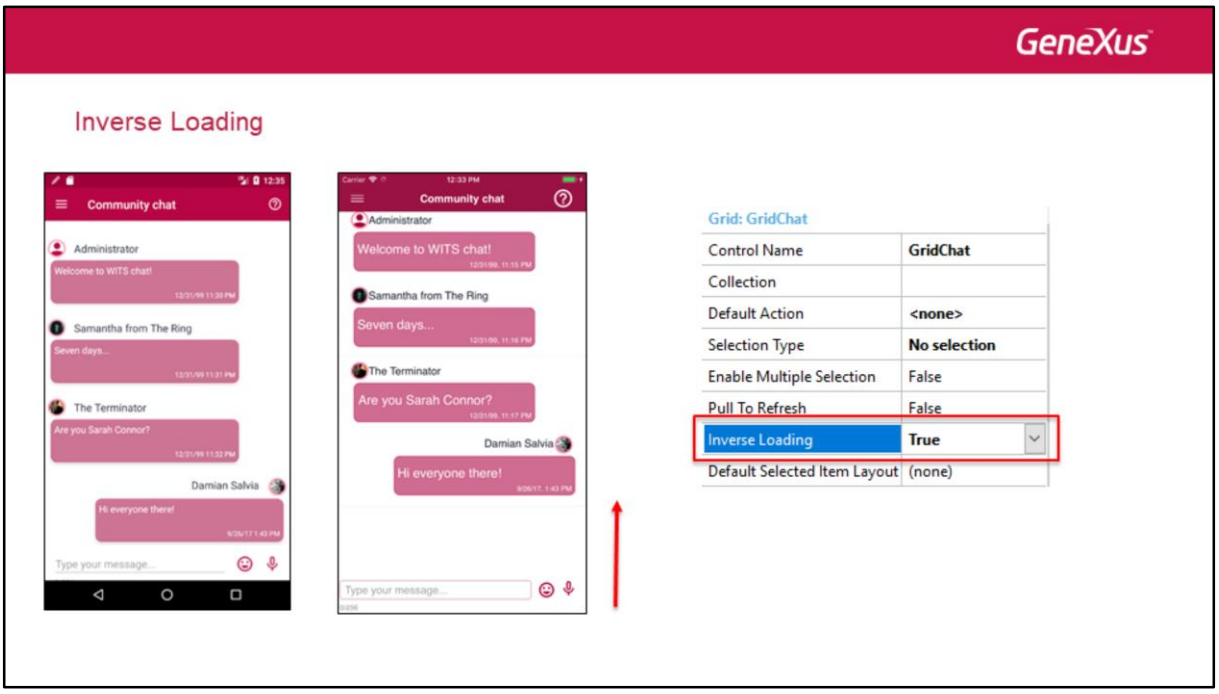
Por ejemplo podemos enviar:

- Un título
- Un subtítulo
- Un ícono
- Un conjunto de acciones

También pueden agregarse otras features.

Si se aprecia en la notificación del chat se indica que ha llegado un nuevo mensaje de audio.

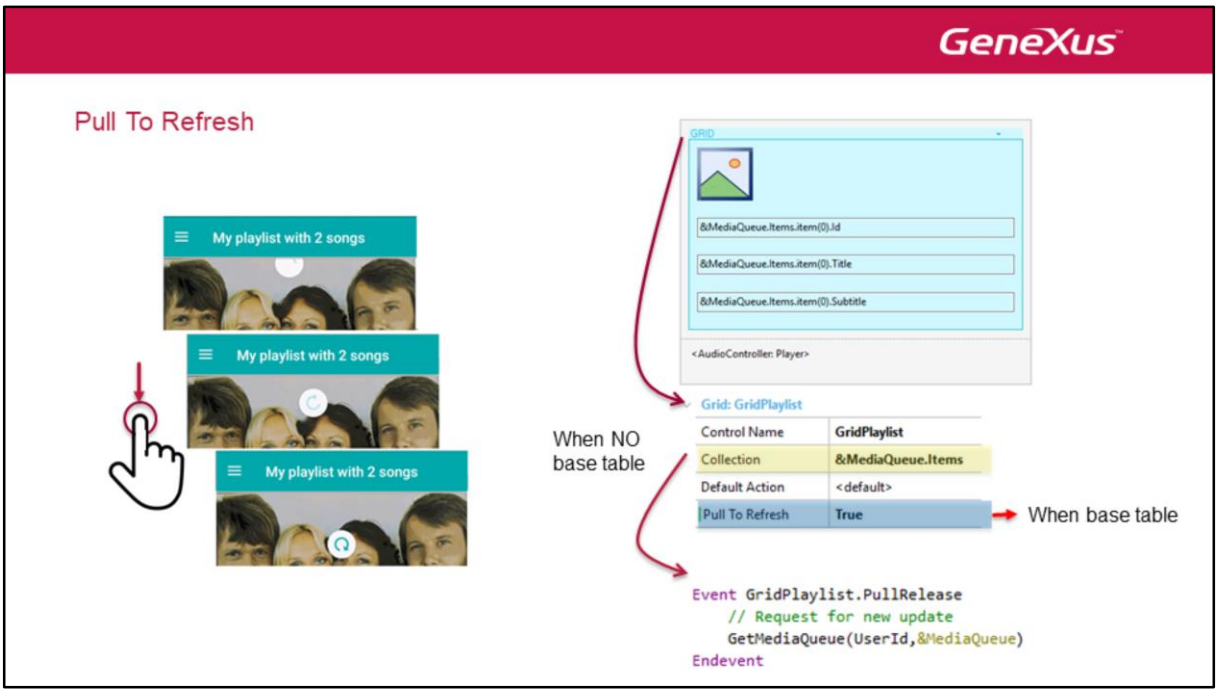
Grids:
Inverse Loading
Pull to Refesh



En el panel que implementa el chat de la app vemos una nueva propiedad a nivel del grid, llamada Inverse Loading, que permite que el grid se cargue en sentido inverso.

En el ejemplo, para simular el comportamiento de un chat simple, agregamos un order al grid por un atributo ChatTimestamp. De esta forma los registros más nuevos serán mostrados abajo del todo en el grid.

Aprovechemos y veamos la otra nueva propiedad, Pull To Refresh.

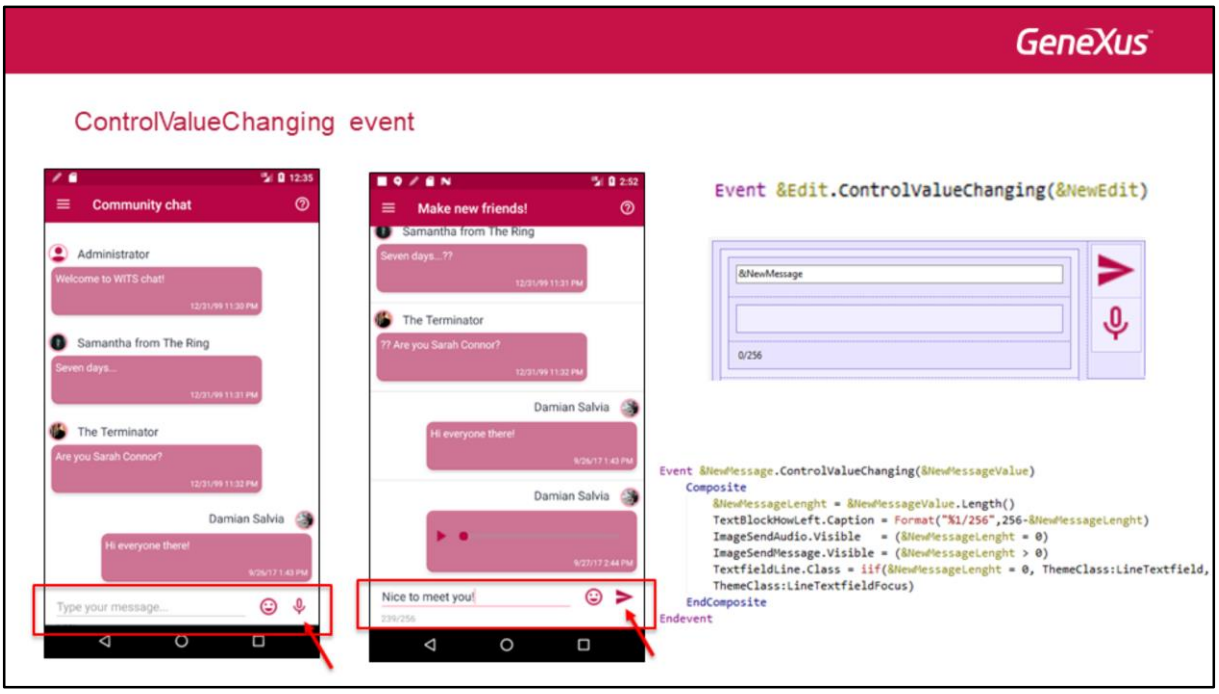


Todo usuario está familiarizado con esta feature. Si se tiene una scrollable screen que muestra datos que cambian (como las apps de Facebook o E-mail), el drag-down effect en la parte superior de la pantalla es una forma intuitiva de refrescarla para actualizar lo que el usuario está viendo.

Esto se consigue con la propiedad del Grid Pull To Refresh. Cuando está habilitada, el comportamiento default será preguntar al server si hay nueva información para devolver, en cuyo caso el grid de entrada será refrescado (y solo él, no el panel completo). Esto solo vale cuando el grid tiene table base. En otros casos (como el del ejemplo), podemos utilizar el nuevo evento **PullRelease**. Es disparado cuando el usuario final suelta la pantalla luego de hacer un pulling de ésta.

En el ejemplo simplemente se vuelve a cargar la variable colección &MediaQueue para que se refresquen las canciones que se muestran en el grid.

ControlValueChanging
new edit field event



Se ha incorporado un nuevo evento que se ejectua mientras el usuario final está ingresando datos de entrada. Su propósito es dar feedback constante al usuario, por ejemplo con efectos de UI.

En el ejemplo del chat, mientras el usuario no ingresa ningún texto aparece el ícono para grabar audio. Pero cuando el usuario empieza a escribir texto en el campo, el ícono cambia al de enviar texto, y además por cada letra que el usuario va ingresando/borrando, se lleva un conteo abajo, pues no se permiten textos de más de 256 caracteres.

Para capturar cada momento en que el usuario cambia algo del campo editable, tenemos el evento ControlValueChanging de ese control.

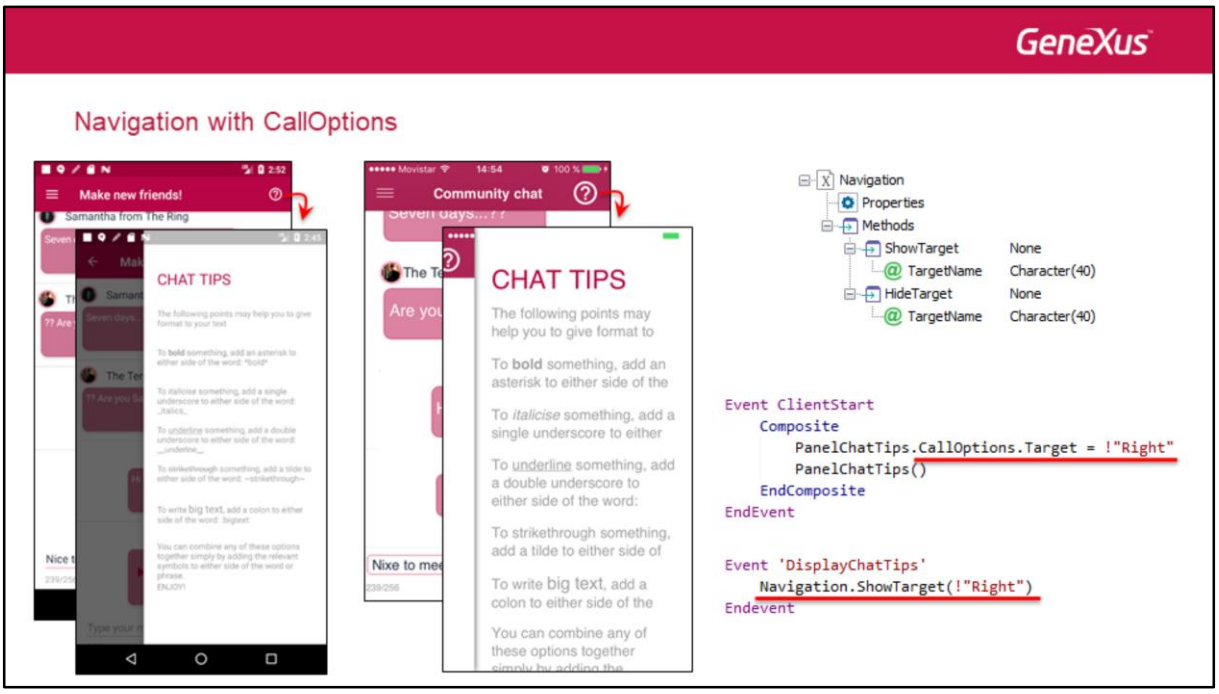
Event *VarOrAtt*.**ControlValueChanging(***&newVarOrAtt***)**
    Event_code
EndEvent

Donde:
VarOrAtt: es cualquier control en el layout que sea editable. El valor de este control será el valor anterior al cambio.

&newVarOrAtt: es una variable del mismo tipo que *VarOrAtt*. Su valor será el siguiente valor al de *VarOrAtt* una vez que ha sido cambiado.

Event_code: es el código que será ejecutado cuando se dispare el evento.

Navigation



Muchas apps utilizan el patron de Slide navigation para desplegar un menu haciendo tap sobre un ícono en la parte superior izquierda de la pantalla (familiarmente conocido como “ícono de hambuerguesa”).

Esquemáticamente, el panel principal (nuestro menú) es cargado en el costado izquierdo de la app y permanence escondido hasta que el usuario hace tap sobre el ícono. Esta manera de mostrar contenido ha sido expandida a otras secciones. Por esta razón GeneXus 15 incorpora el right navigation style utilizando el atributo **CallOptions** de un panel (para iOS también el bottom navigation style).

En el ejemplo, observemos que el ícono de hamburguesa tiene el comportamiento conocido, pero observe que hemos incluido un nuevo botón en el lado derecho de la Application Bar: un help icon. Su propósito es desplegar un panel de help contextual una vez que le usuario final hace tap en el ícono.

Para implementar esto, primero que nada necesitamos que el estilo de navegación de la app sea Slide. Luego:

1) Indicar a través del Target attribute de las **CallOptions** del panel que queremos invocar a la derecha... ¡que queremos invocarlo a la dercha!

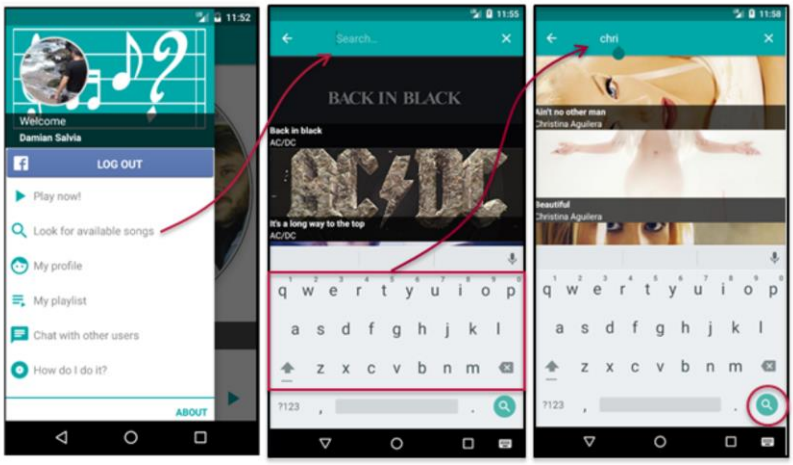
2) Realizar la invocación al objeto que queremos mostrar a la derecha. Por defecto esta acción simplemente carga el panel en el sector derecho pero no lo muestra.

3) Desplegar el menu contextual utilizando el nuevo **external object** denominado **Navigation**.

Define your own Search

GeneXus

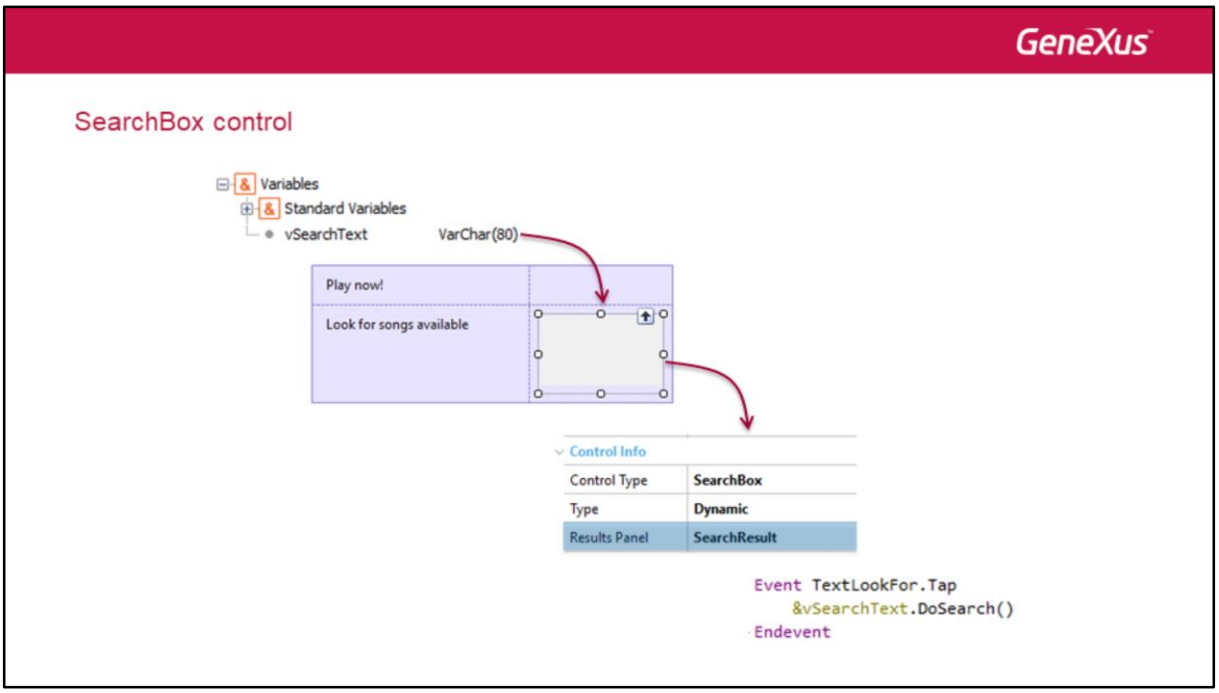
Define your own search



GeneXus 15 da la flexibilidad a los desarrolladores de definer y diseñar sus propios searchs.

En la primera imagen vemos una opción del menu que le permite al usuario final realizas una búsqueda en la app.
A partir de allí la Application Bar se transforma en un search box (en iOS luce un poco distinto pero es básicamente lo mismo). Cuando el usuario final va introduciendo su texto, el comportamiento del resultado puede ser configurado para refrescar el contenido automáticamente o esperar hasta que el usuario final seleccione el botón de search del teclado.

La diferencia con la propiedad Search de un grid es que este mecanismo es más flexible pues permite definer búsquedas globales en la app.



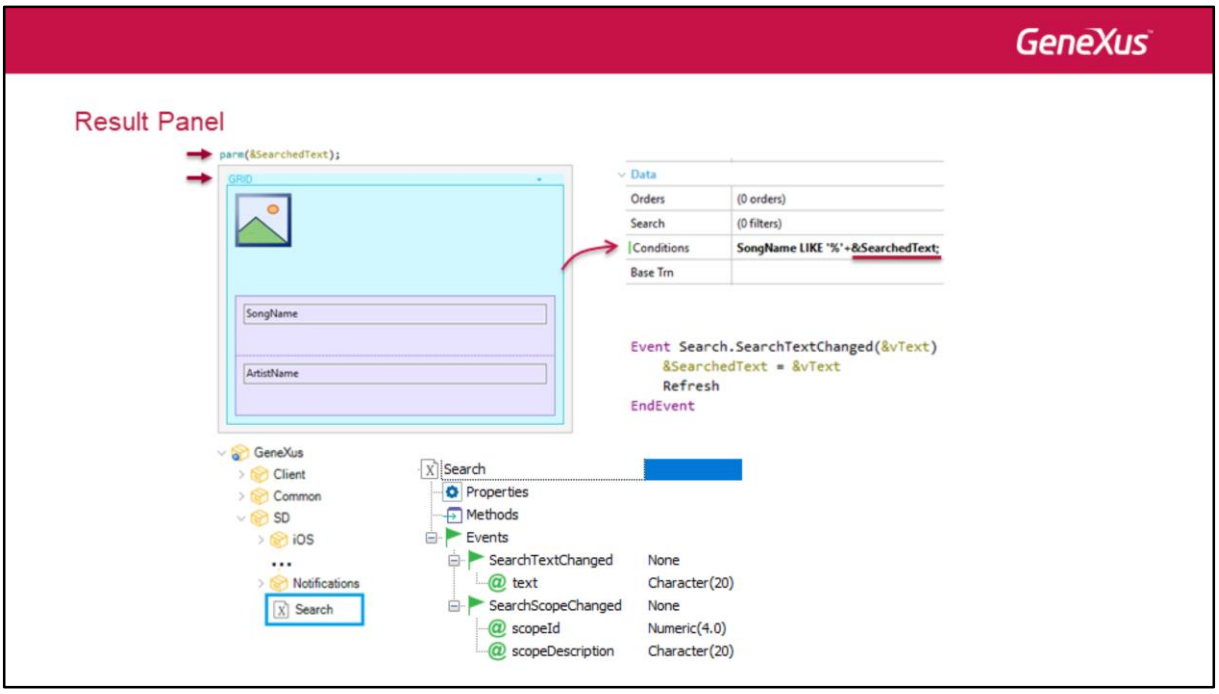
¿Cómo lo logramos?

Simplemente definiendo una variable string que alojará al texto a ser buscado, e insertándola en el layout, con el Control Type **SearchBox**.

Se desplegarán dos propiedades nuevas al elegir ese control type: Type y Panel Result.

Type: permite indicar si el texto a ser buscado será mostrado dinámicamente (a medida que el usuario tipea el texto) o explícitamente (cuando presiona la tecla de Search del teclado).
Panel Result: se indica el panel que contendrá el resultado de la búsqueda.

Por default, cuando el usuario final hace tap en la variable string desplegada en el layout, se habilitará la search-box (lista para que se empiece a tipear texto) y se desplegará el Panel Result. Pero también podemos invocar a la search-box en forma programática, a tavés del método **DoSearch** asociado a estos tipos de variables.



El Result Panel es la esencia del patrón de búsqueda.

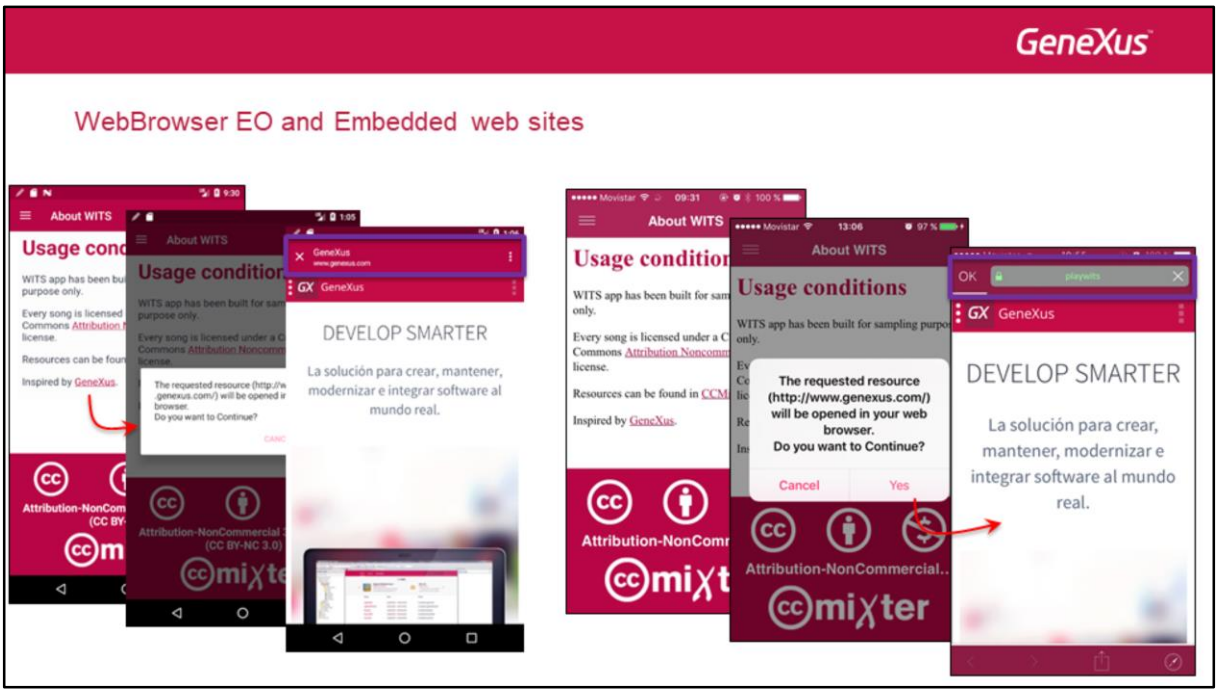
Debe:
1) Incluir la regla **parm** para recibir el texto a ser buscado
2) Incluir al menos un control **grid** con una **Condition** para filtrar los items.

Más aún, si queremos refrescar el contenido del grid cada vez que el usuario tipea un nuevo texto, podemos utilizar el **external object Search**, con el método **SearchTextChanged** que se dispara cuando el contenido de la search-box cambia y que recibe por parámetro el nuevo texto.

En nuestro caso, nuestro propósito es simplemente actualizar la variable que condiciona la carga del grid, y volver a refrescarlo.

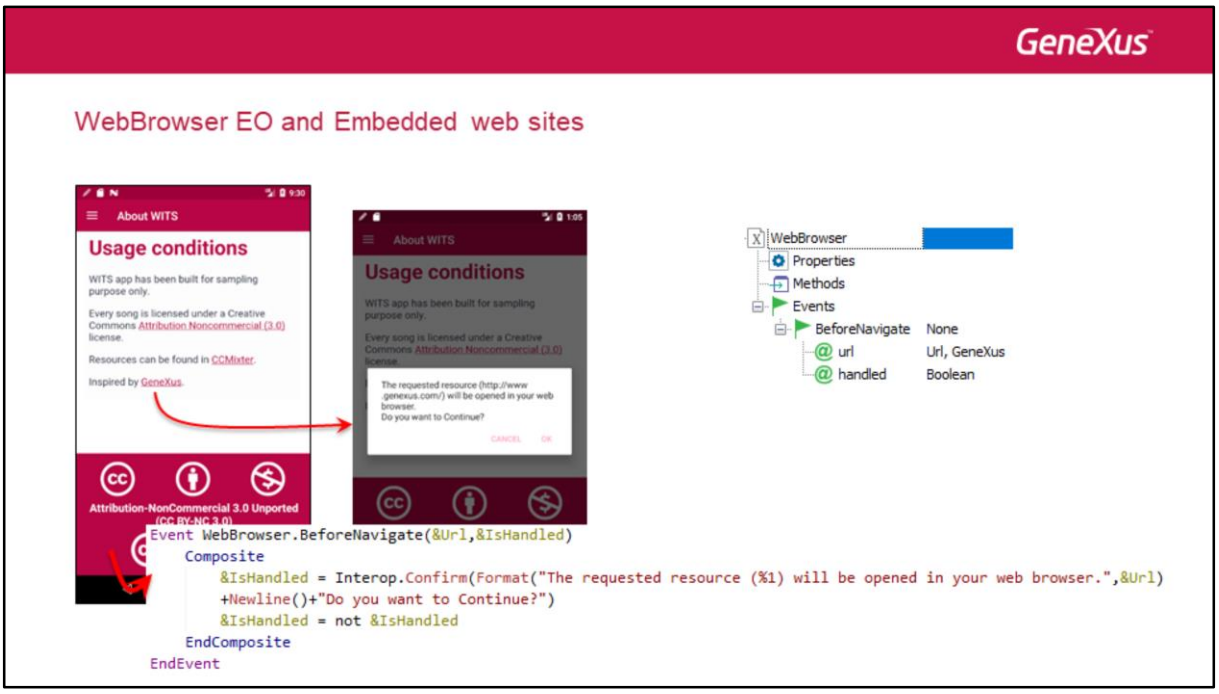
Por más información, puede visitar en nuestro community wiki el documento: **HowTo: In-app search in Smart Devices**.

In-app web content



En el ejemplo, cuando el usuario hace tap sobre el link GeneXus en el SD panel PanelAbout, antes de abrir el link en el web browser del dispositivo, se le pide confirmación al usuario.

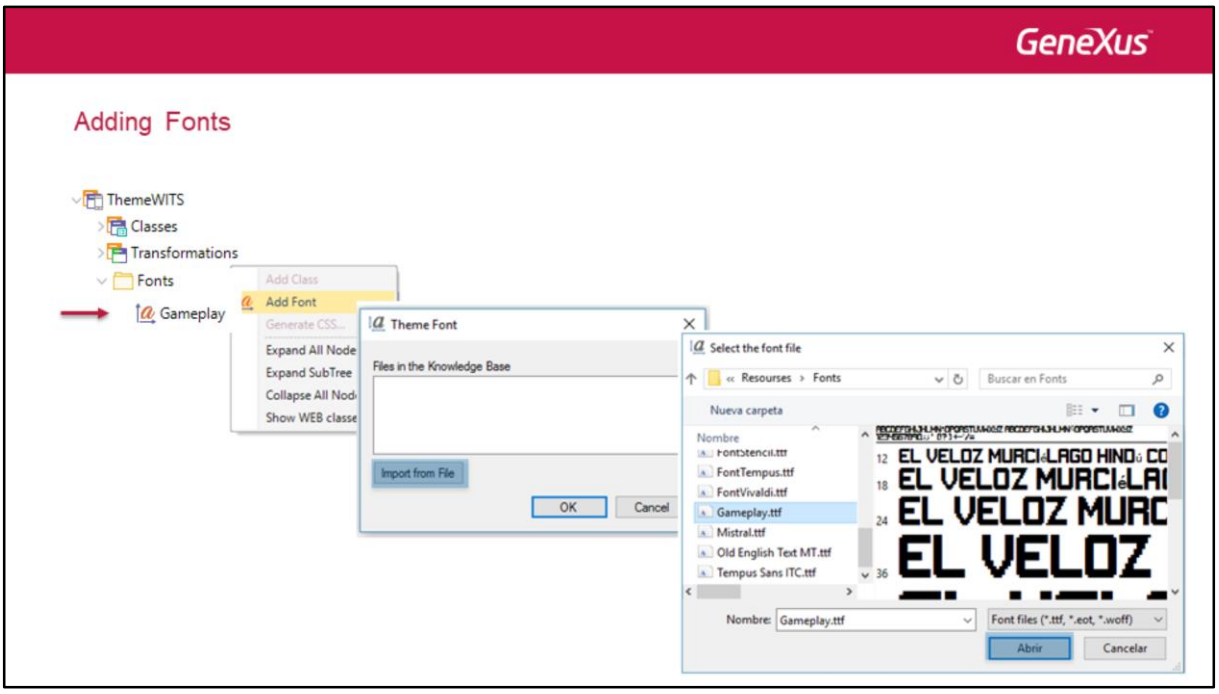
Obsérvese que ya por defecto en GeneXus 15 si se invoca desde un objeto SD a una página web, se abre el navegador (sólo si es Chrome o Safari –según se esté en Android o iOS respectivamente–, pero conservando la application bar de la app nativa, de modo que el usuario no sienta que salió de la aplicación.



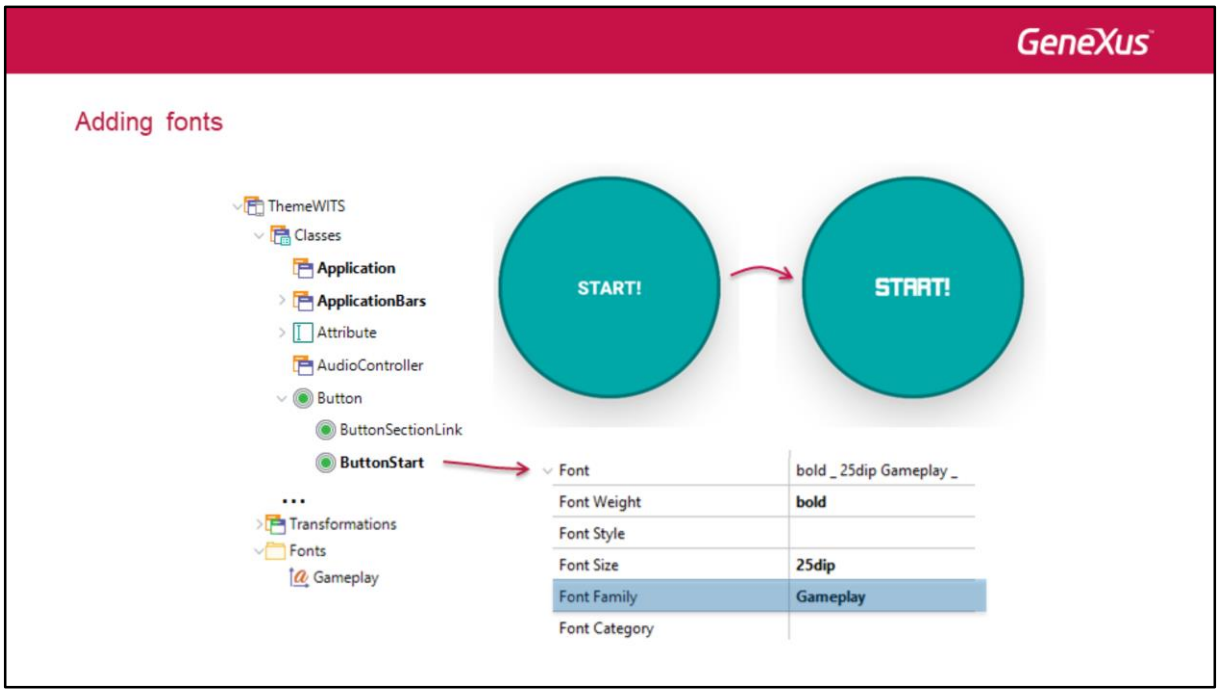
Contamos con un nuevo external object, WebBrowser, que nos permite ejecutar código cuando se ha invocado al navegador para abrir un link, un instante antes de hacerlo.

El evento BeforeNavigate del external object captura ese momento. Se indica si la app invoca al browser o no configurando el parámetro de salida handled, que por defecto está en False.

Fonts



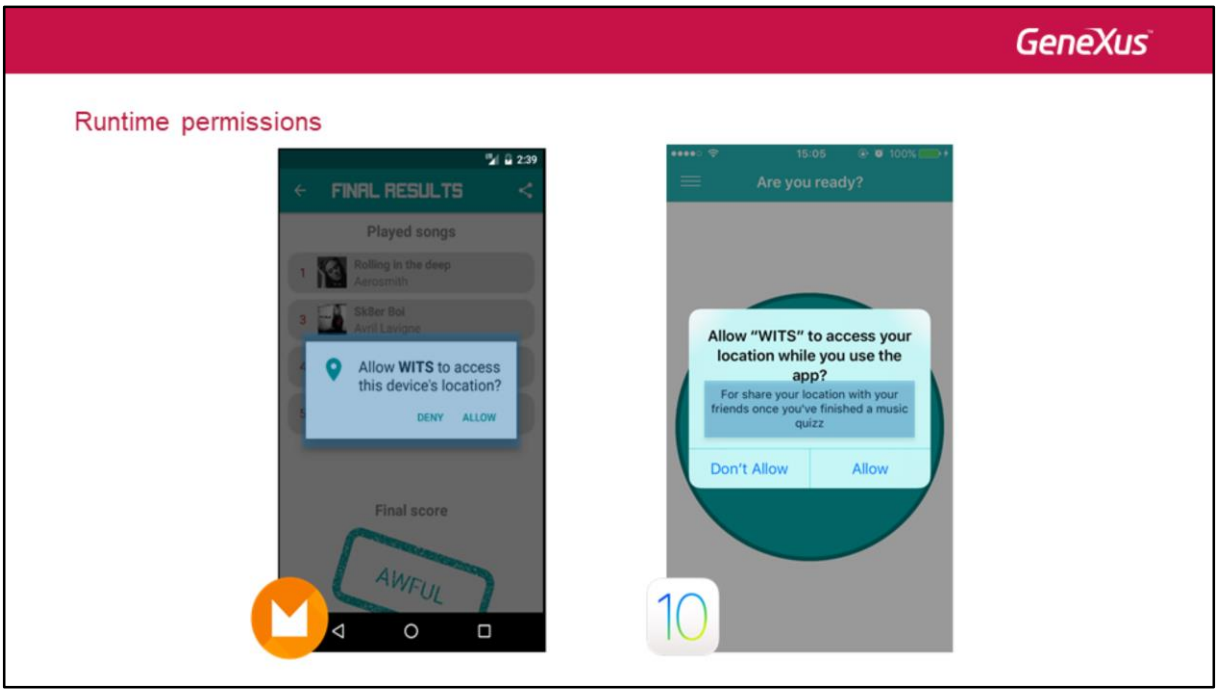
El objeto **Theme for Smart Devices** agrega un nuevo nodo **Fonts**, que ayuda al desarrollador a agregar fonts personalizados a la KB.



En el ejemplo agregamos una Font Gameplay al theme.

Luego en una clase button del theme podemos ver la propiedad **Font Family property** que muestra la nueva fuente en el combo-box.

Run-time permissions

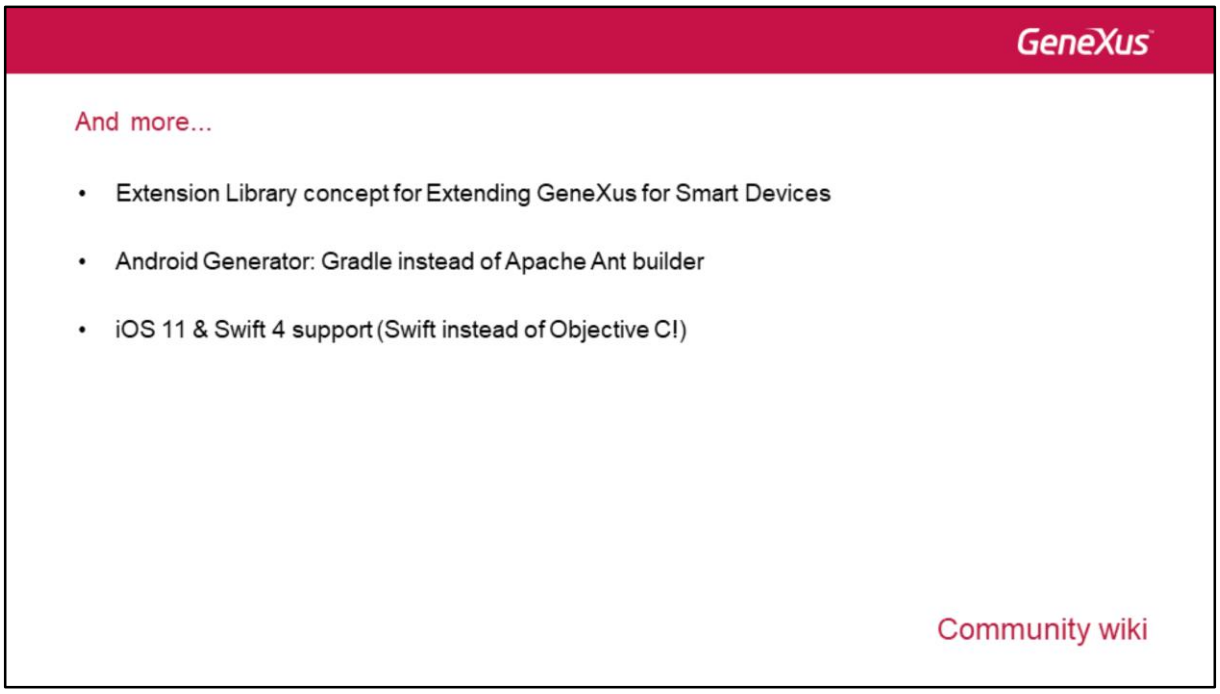


Las últimas versions de Android e IOS han mejorado sus esquemas de permisos.

Android, desde la version 6.0, incorpora runtime permissions. ¿Qué significa? Versiones previas pedían todos los permisos cuando la app era descargada. Las nuevas versions los piden a demanda. Este escenario es resuelto automáticamente por GeneXus y no requiere configuración extra.

iOS, por otro lado, aunque ya pedía permisos en runtime, a partir de la version 10 cada pedido debe incluir su descripción de uso. Apple fuerza a todos los desarrolladores a clarificar por qué están utilizando determinada feature del dispositivo (por ejemplo, el gps). Estas descripciones deben ser especificadas por el desarrollador GeneXus en el grupo **Purpose Strings properties** del objeto principal.

And more...



El concepto de Extension Library es introducido en GeneXus 15 para facilitar el desarrollo de extensiones para Smart Devices, como por ejemplo User Controls, External Objects y External Providers para SD.

Se cambia el anterior Apache Ant que se utilizaba para contruir la app, por Gradle, que produce una salida más clara, es más rápido debido a su building incremental, brinda más flexibilidad y se adapta a las nuevas tecnologías. Debido a esto ahora es posible configurar cuánta memoria será utilizada para el build de la app, si la KB es suficientemente grande, entonces se puede usar en modo multi-dex, etc.

En iOS ya no se genera en el lenguaje Objective-C, sino en Swift.

Más documentación en nuestro community wiki.

GeneXus™

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certificactions	training.genexus.com/certifications