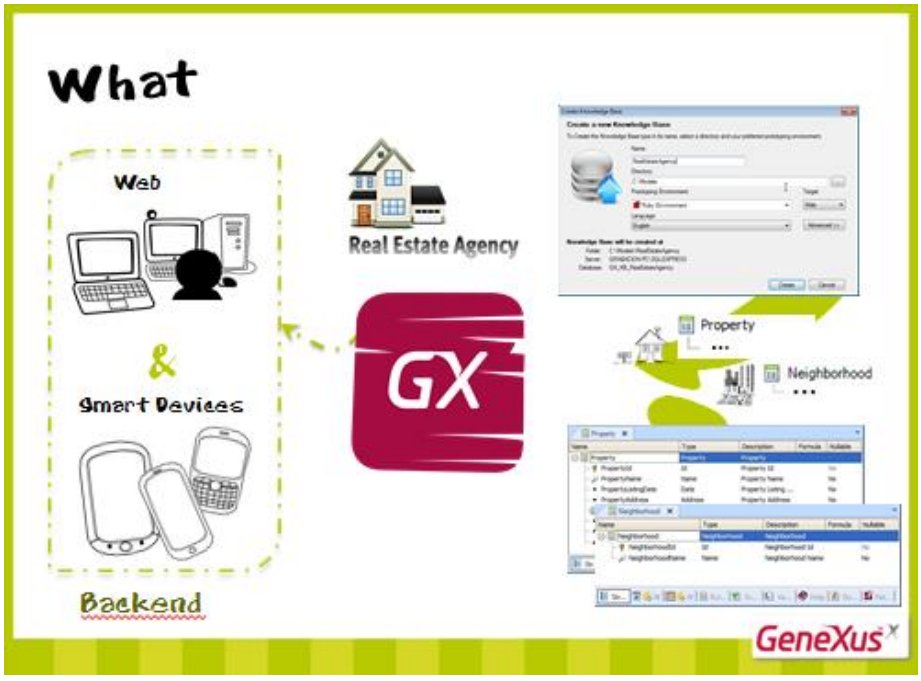
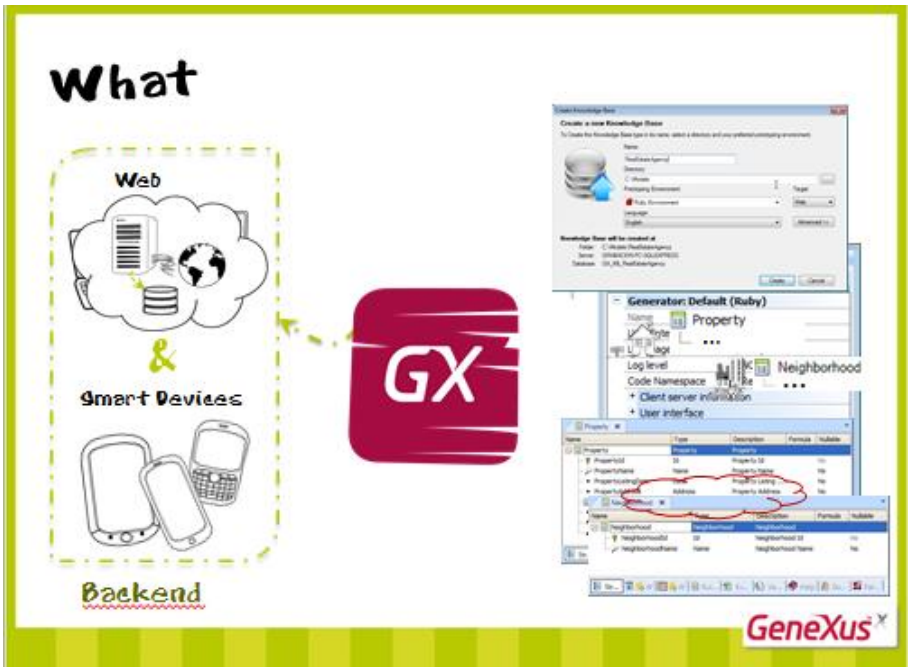
VIDEO 1: Work with for Smart Devices – Generalidades y Layout del List



Queremos construir un backend para una inmobiliaria: con una parte web y otra para Smart Devices, para ser utilizada por los agentes inmobiliarios en su trabajo móvil.

Para ello creamos una KB, la transacción Property para registrar las propiedades inmobiliarias en venta o alquiler, y la transacción Neighborhood, para registrar los vecindarios.



Además definimos que el backend web se generará en **ruby**, en la nube.

Para implementar la parte para Smart Devices, aplicamos el pattern Work With for Smart Devices a la transacción Property…

a la transacción Neighborhood…

y creamos un objeto Dashboard como punto de entrada.

Decidimos en principio generar sólo en **Android**, la plataforma por defecto.

Estudiaremos en lo que sigue la forma de personalizar los “Trabajar con” una entidad (representada por una transacción).

En otra parte veremos que también pueden crearse “Trabajar con” no asociados a transacciones.

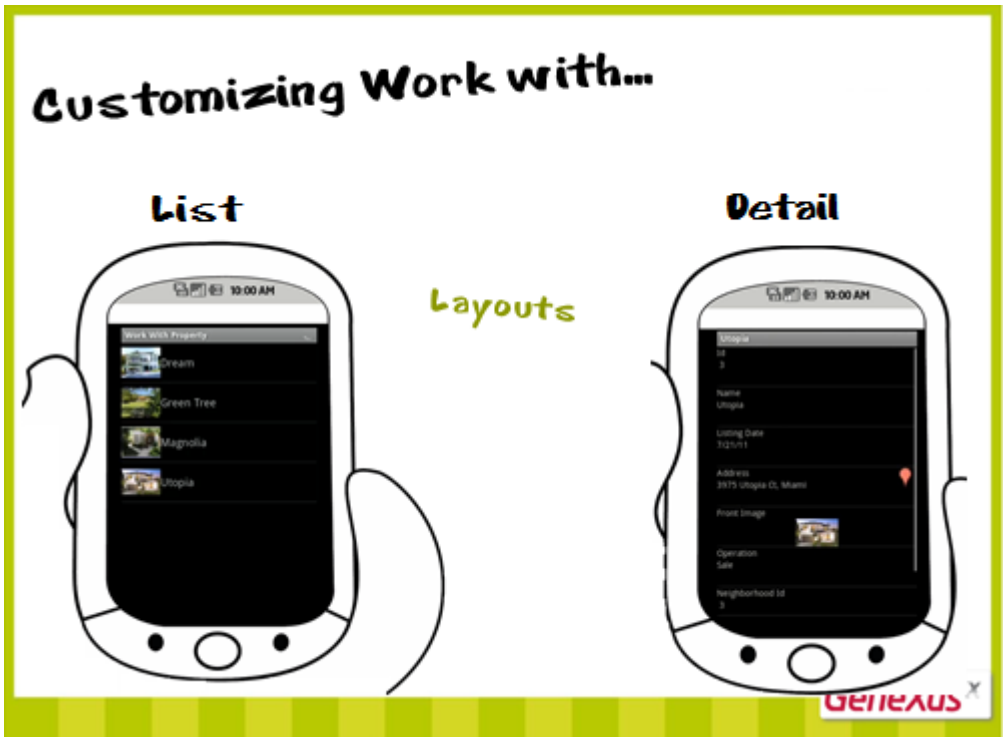
Como sabemos, la especificación de que se desea tener un “Trabajar con” cierta transacción, implementa automáticamente una pantalla para mostrar la lista de los elementos, y una pantalla para mostrar la información detallada de un elemento particular.

Por tanto lo primero que podemos personalizar es la información mostrada al usuario y la forma en que se la muestra. Es decir: todo lo que tiene que ver con la interfaz de usuario, esto es: los Layouts.

También podemos personalizar el orden por el que sale listada la información del List…

o las búsquedas que se ofrecen al usuario para seleccionar la información a recuperar, así como otros filtros internos para condicionar la información a desplegar…

Y por último, podemos personalizar el comportamiento, agregando o modificando eventos para realizar ciertas acciones.



Empecemos por las pantallas.

Dependiendo de cada una, las personalizaciones que podremos hacer…

{demo}

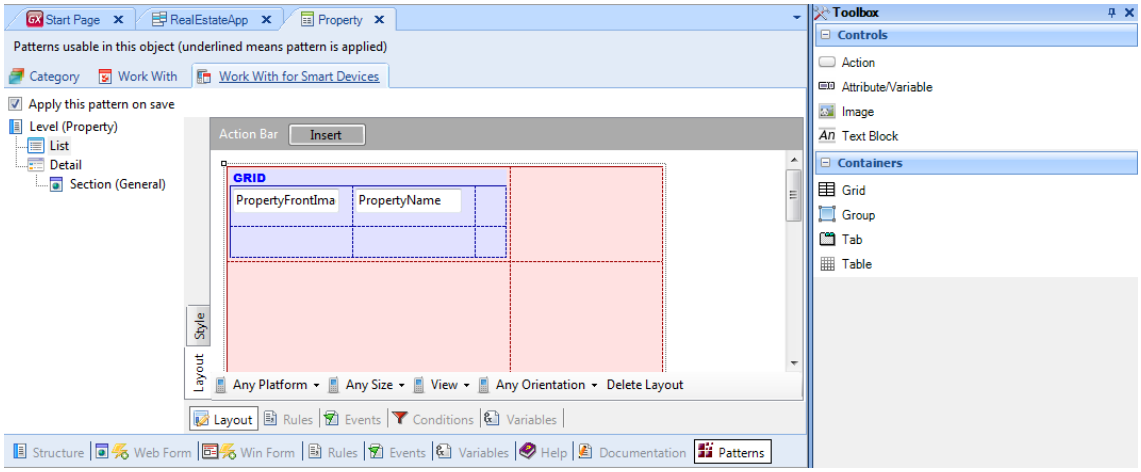
{demo. Ponerle al Theme Android, a la clase Attribute margin left = 5}

Viejo:
http://apps2.genexus.com/Id8704734959bf4b73ad79978b4893985e/DeveloperMenu.xml

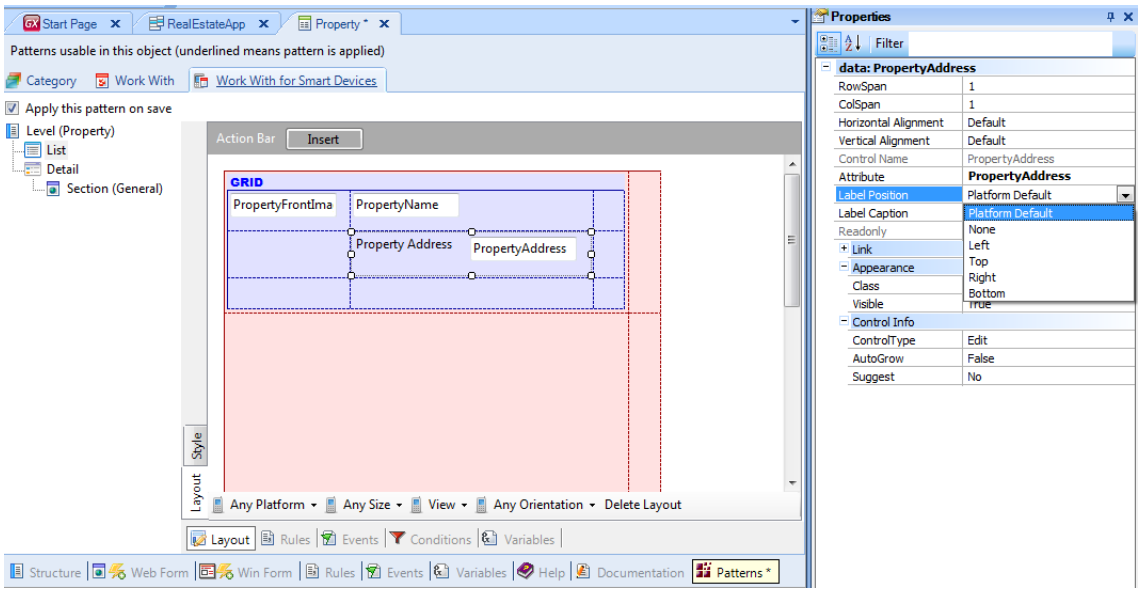
Actual:
http://apps2.genexus.com/Ide18ce2b1f85c4941a47debd1e95241cc/DeveloperMenu.xml

Posicionándonos en el nodo List del pattern Work With correspondiente a la transacción Property, observamos que nos muestra el Layout.

En forma predeterminada tenemos un control grid (que es el que implementa el listado) y dentro de él aparecen, de todos los atributos de la transacción, solamente algunos. Podemos agregar o quitar atributos a ese grid, así como otros controles, a través de la toolbox.

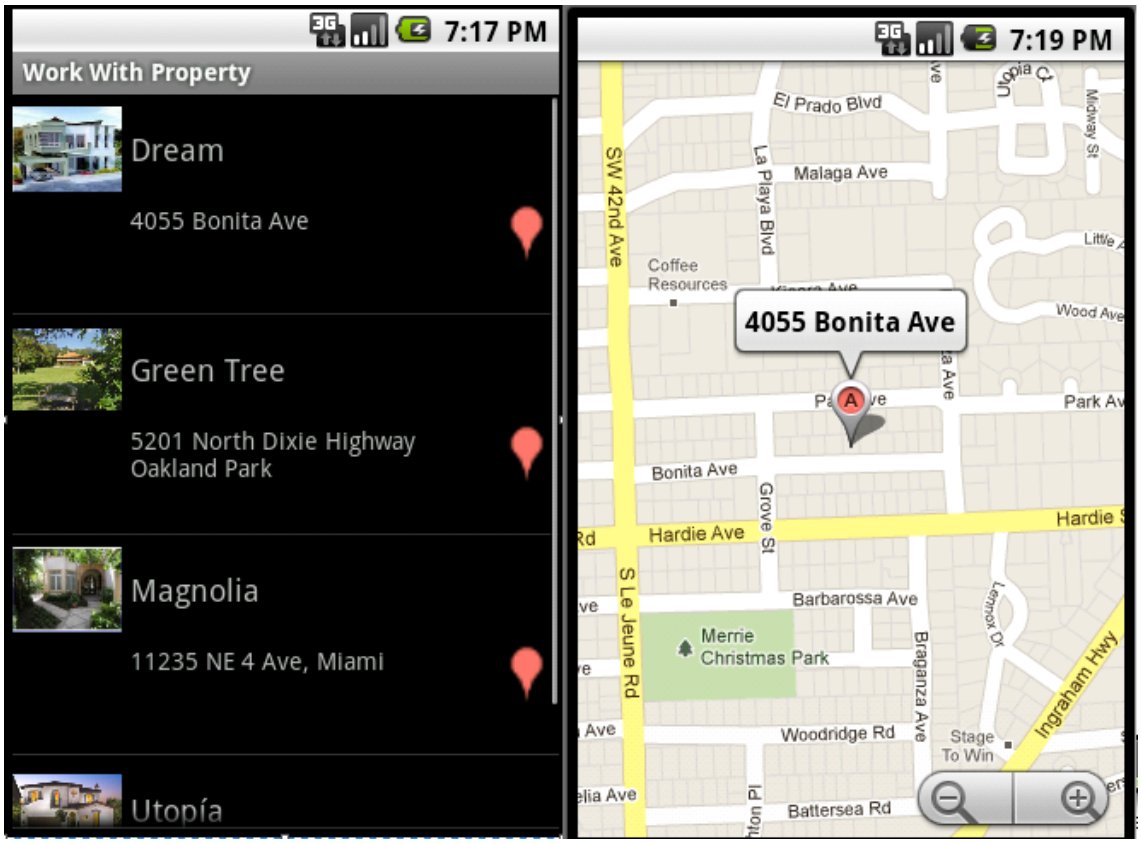


Por ejemplo, podemos agregar la dirección: PropertyAddress…



Y personalizar las propiedades de este control: por ejemplo, la posición de la etiqueta… aquí la vemos a la izquierda, pero supongamos que la queremos arriba (como suelen aparecen en Android)…(hacerlo) o no queremos que aparezca (hacerlo)… El valor "Platform Default" es como un comodín: establece que en ejecución se mostrará de acuerdo al estándar de la plataforma elegida. Si es Android se mostrará arriba, aunque en esta pantalla de diseño la veamos a la izquierda.

Veámoslo en ejecución.



Vemos que junto a la dirección, aparece el ícono para poder verla en un mapa (debido a que el dominio Address en el que está basado el atributo {dejar tiempo muerto para poder poner un callout imagen con la estructura de la trn} tiene asociada la semántica correspondiente, por lo que, en forma transparente para nosotros, el programa generado para el Smart Device incorpora la lógica para utilizar su mapa **nativo**). Vemos así que la aplicación generada se **integra** con funcionalidades nativas del dispositivo. {callout que diga "Device integration… with native functionalities"}

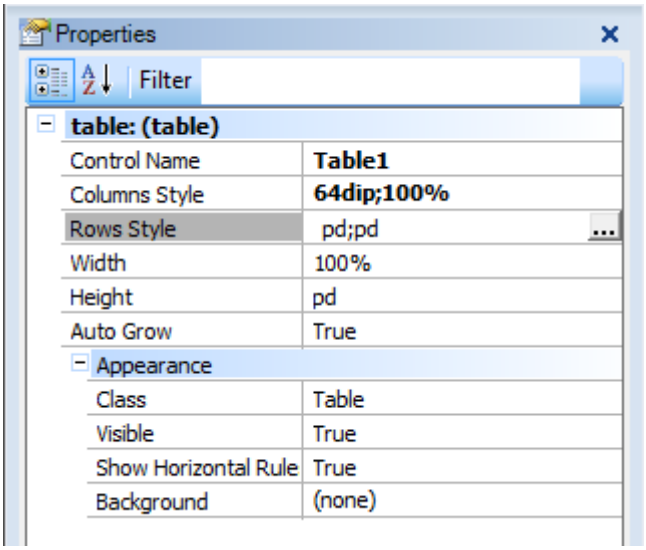
Ahora, queremos modificar la distribución de los controles y los espacios entre la información.

Observemos que la información de cada propiedad se presenta en dos filas {poner callout para señalarlar, por lo que hacer un **silencio** en el audio para que entren}

Y en dos columnas {poner callout para señalarlas, y hacer un silencio}. Es decir en una tabla (interna al grid). {Mostrar la tabla en GX }. La imagen de la propiedad y su nombre, aparecen en la primera fila y la dirección en la segunda.

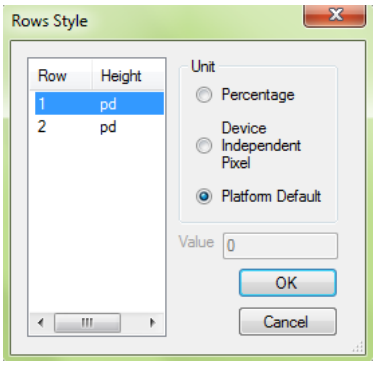
Si deseamos que la columna con la imagen sea un poco más ancha… en las propiedades de la tabla, modificamos el estilo de las columnas. Por defecto la primera ocupa 64dips (Device independent pixels) y la segunda se extiende hasta alcanzar el 100% del ancho.

{callout que diga "Dip?"} Dip? Es una unidad de medida que representa una abstracción sobre el pixel, de manera de poder escalar las pantallas a diferentes tamaños. Luego la aplicación generada los convierte a pixels físicos, de acuerdo a la plataforma del dispositivo.



Aquí se está dando un tamaño fijo de 64 dips para la primera columna. Supongamos que queremos, por el contrario, que ocupe el 30% del ancho. Y la segunda el 70% restante {hacerlo en GX y mostrar resultado en el emulador}

Por otro lado, si observamos las filas, vemos que tienen ambas el mismo tamaño, pero podemos variarlo. {ir a GX y posicionarse en la tabla, fila Rows Style}



Vemos que ambas filas tienen el valor “pd”. ¿Qué significa? “Platform Default”. Es decir, el tamaño (en cuanto al alto) de cada fila, será tomado del valor default de la plataforma. El valor default de Android/Phones, es de 64 dips.

Podemos modificar los valores de cada fila, tanto fijándolos en dips, por ejemplo la primera fila ocupará 65 dips y la segunda 35, como a través de un porcentaje relativo al alto de la tabla. Aquí le diremos que la primera fila ocupe el 65 porciento del alto, y la segunda el 35 porciento, donde el alto de la tabla está configurado a través de la propiedad Height, que por defecto asume el valor “Platform defaul” pero que puede ser variado a 100dips, por ejemplo.

{mostrar el resultado en el Emulador}

También podríamos querer que la imagen se extienda para ocupar las dos filas.

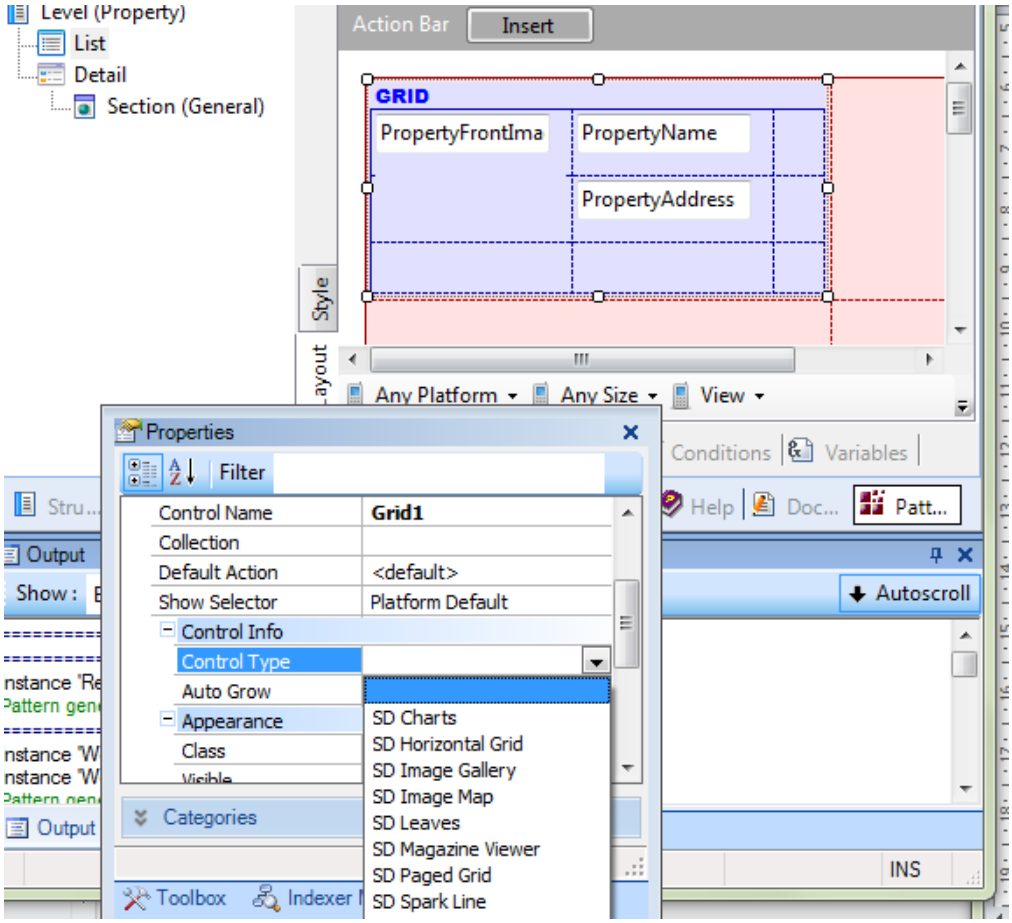
{hacerlo en GeneXus y mostrar el resultado en el emulador}

Ahora supongamos que en vez de visualizar la lista de propiedades inmobiliarias como una sucesión de elementos en fila, queremos verlas de otra manera, por ejemplo, como una galería de imágenes de fachada.

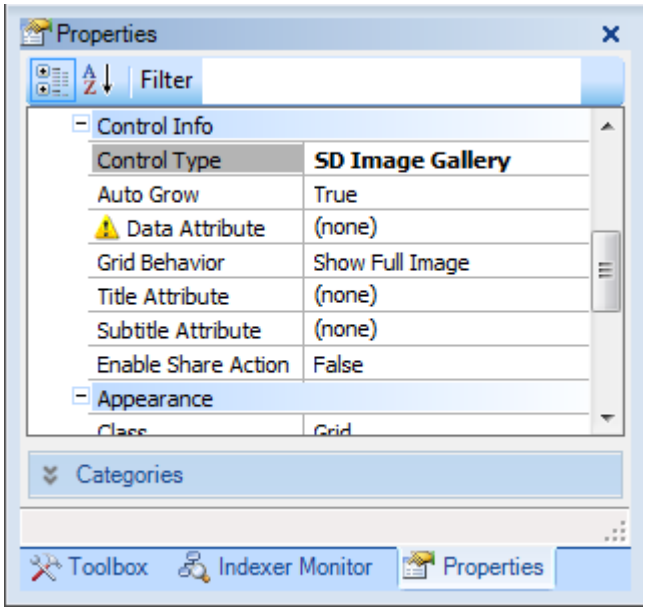
¿Quién implementa en GeneXus la lista?

{ir a GX y pararse sobre el grid}

El control grid. Busquemos entre sus propiedades una que represente la forma en que se presenta la información. Ésta: el tipo de control. Cuando no tiene valor asginado, representa el tipo de listado estándar.



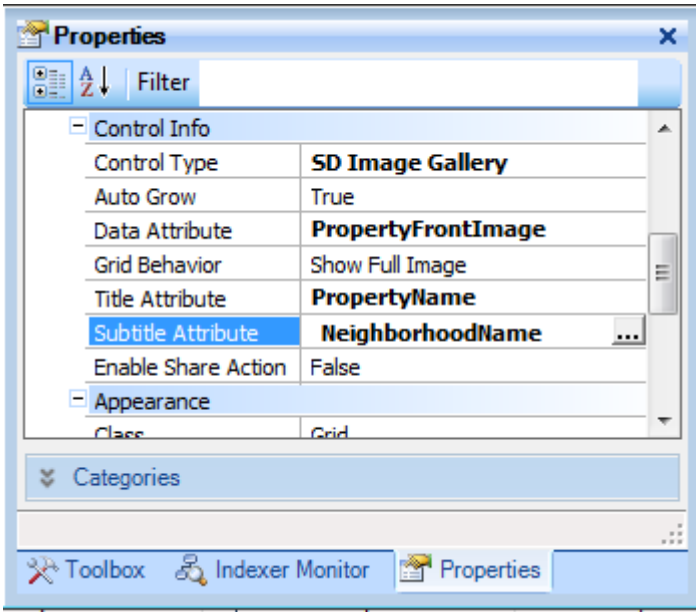
Pero podemos modificarlo, eligiendo entre los user controls específicos para Smart Devices que nos ofrece en esta lista. Entre ellos, elijamos el user control Image Gallery.



Tenemos que configurar cuál es el atributo que contiene la imagen utilizada para la galería {poner Data Attribute = PropertyFrontImage}

y si lo deseamos el atributo que utilizaremos como título de la imagen {hacerlo}

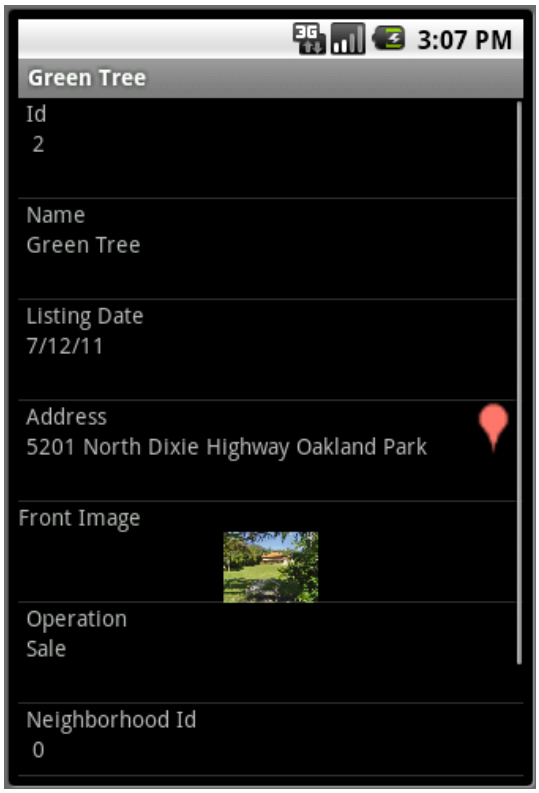
y el que usaremos como subtítulo {hacerlo}.



{mostrar el resultado en el emulador}

Vemos que podemos movernos por la galería tanto entre las imágenes en miniatura como a través de la imagen ampliada,

Y haciendo tap sobre la imagen de la propiedad, vemos su detalle:



De acuerdo a los datos que manejemos de la entidad, los user controls que tendrán sentido en ese contexto. Por ejemplo, para utilizar el user control SD Maps, que muestra el listado como puntos en un mapa, necesitaremos contar con un atributo para la propiedad inmobiliaria de tipo de datos el dominio **Geolocation**, que almacena latitud y logitud geográfica de esa propiedad.

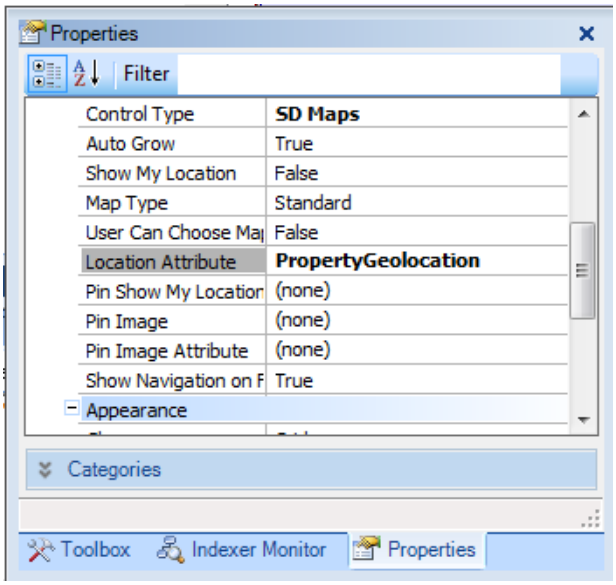
{en GX ir a la Structure de la trn y agregar el att PropertyGeolocation y grabar.

Lo definimos en la estructura de la transacción

Vemos que asume automáticamente el dominio Geolocation

Pasar al pattern y posicionarse en el grid para dale valor a la propiedad Location Attribute}

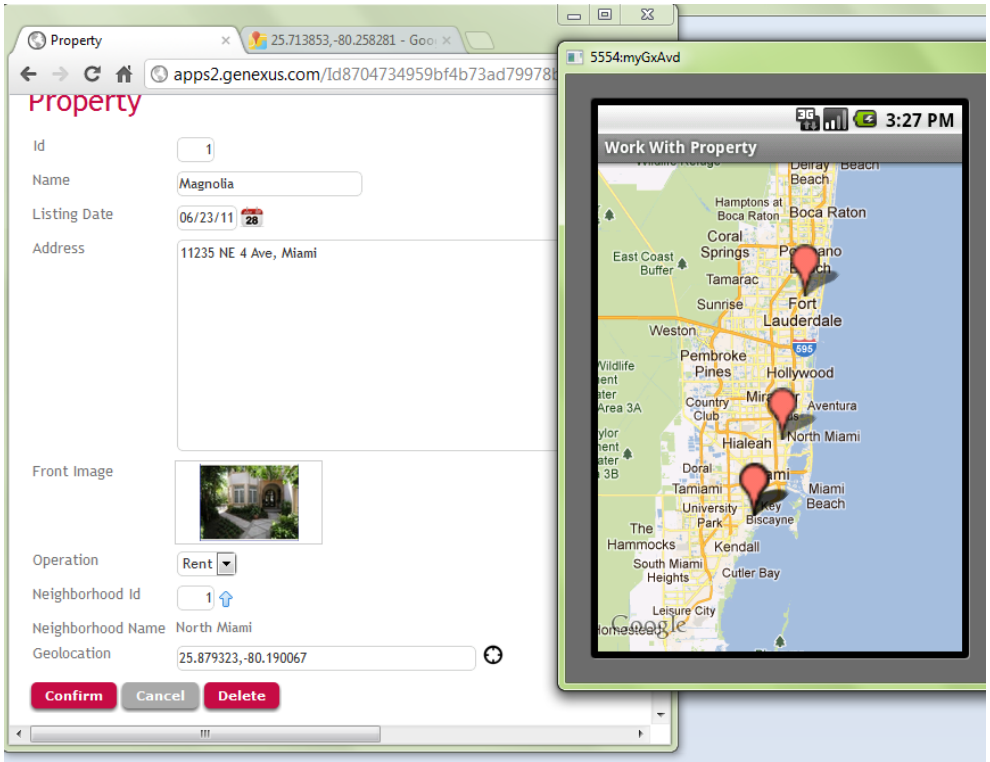
Y ahora sí, indicamos que el atributo que contiene la localización geográfica es el que acabamos de crear, PropertyGeolocation.



GeneXus reorganizará para agregar el nuevo atributo a la tabla y una vez regenerada la aplicación, tendremos que asignar la localización geográfica a las propiedades inmobiliarias que tenemos ingresadas …

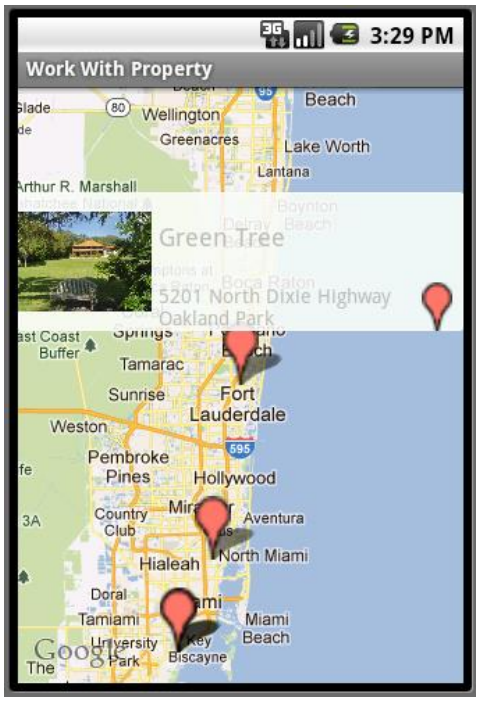
{transición para saltear todo esto }

Una vez hecho esto, vemos que al ejecutar el listado…

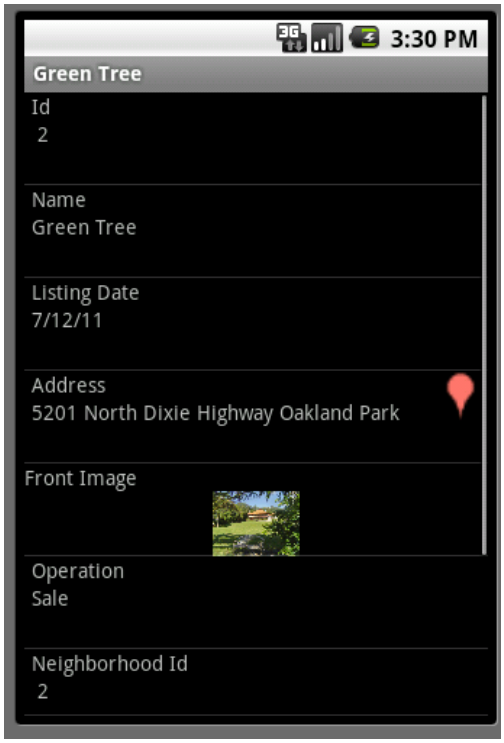


Y al hacer tap sobre un punto de los mostrados…

U otro…



Y si queremos ver el detalle completo de la propiedad, al hacer tap:



La pantalla de List sigue cumpliendo sus funciones, por más que hayamos modificado la forma de desplegar la información. Por ejemplo, podemos hacer un Search por dirección… {hacerlo}

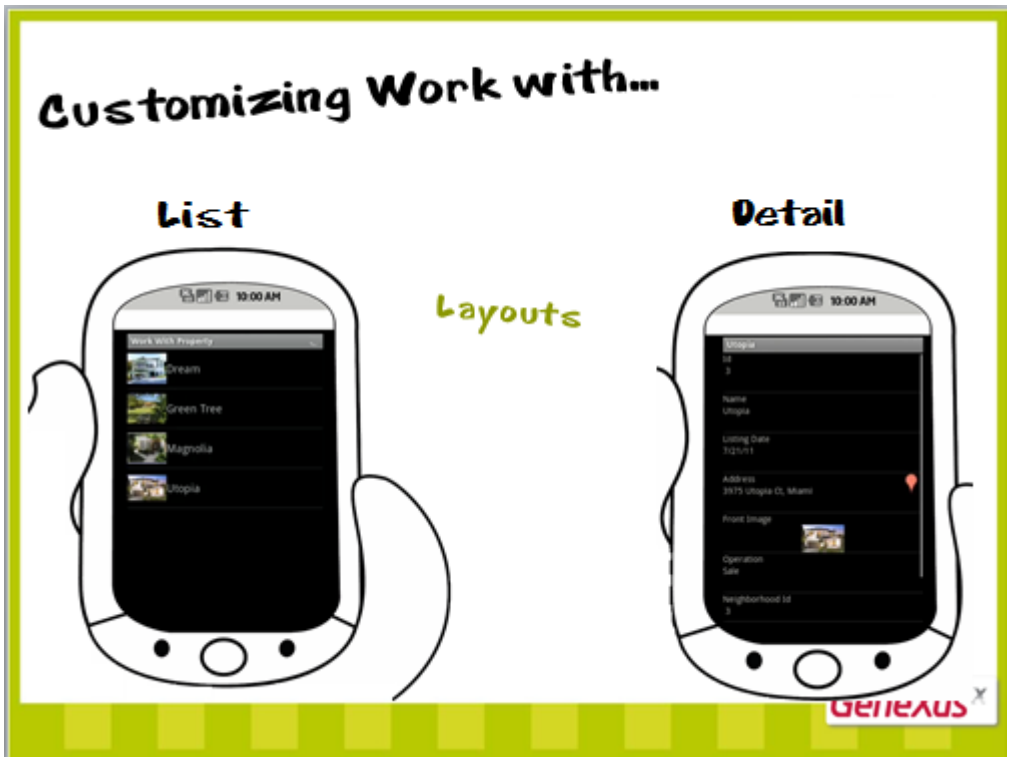
Filtrar por vecindario… {hacerlo por Coral Gables}

o insertar una nueva propiedad inmobiliaria {invocar al Insert sin insertar nada}

{transición hacia la ppt siguiente}

Aquí vemos algunos otros ejemplos de User Controls para mostrar de manera particular la información de las listas.

El desarrollador también podrá crear sus propios user controls. Para esto recomendamos buscar documentación en nuestro wiki.



Acabamos de personalizar el Layout del listado de propiedades inmobiliarias…

También queremos personalizar el de la información detallada de una propiedad en particular...

Para ello contamos con el nodo Detail, que tiene una sección anidada, la general. Investiguemos un poco cada cosa…

FIN VIDEO 1

VIDEO 2: Work wih for Smart Devices – Layout del Detail

Acabamos de personalizar el Layout del listado de propiedades inmobiliarias…

También queremos personalizar el de la información detallada de una propiedad en particular...

Para ello contamos con el nodo Detail, que tiene una sección anidada, la general. Investiguemos un poco cada cosa…

{demo → quitar el SD Maps del List de Properties}

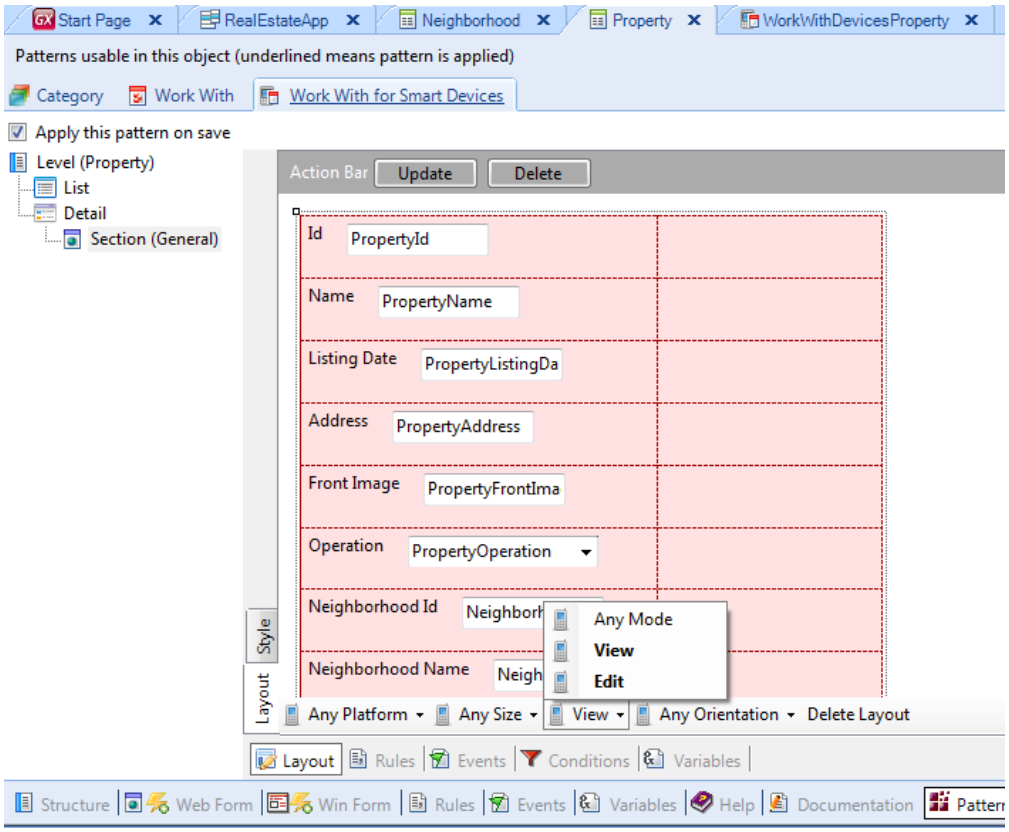
Vemos que el layout del nodo Detail tiene un atributo, el que está marcado en la transacción como Description attribute, y un contenedor de secciones, donde las mismas serán cargadas. En esta transacción {Property} tenemos solamente una sección, pero por ejemplo en esta otra {Neighborhood}, tenemos dos y allí tendrá más sentido este contenedor. Dejémoslo para más adelante.

Por ahora podemos observar que en ejecución, {mostrar el List Property} al elegir una propiedad de la lista aparece arriba del todo el nombre de la propiedad, y luego toda su información. Y al elegir un vecindario, aparece arriba del todo su nombre, y luego dos tabs para ver su información: uno por cada sección.

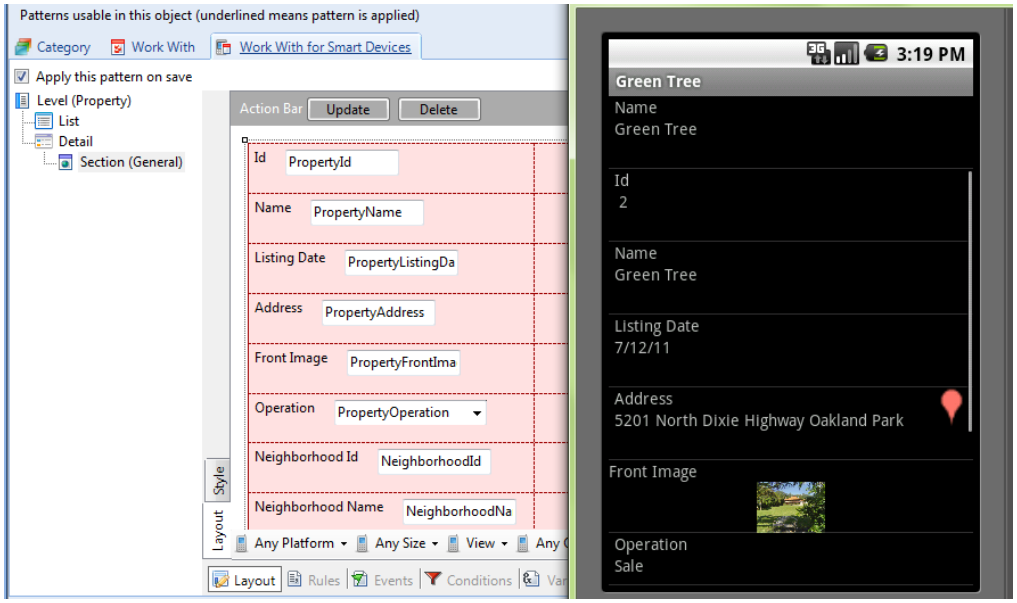
Si eliminamos este atributo dela parte fija…{hacerlo sólo de Property} F5…{transición} vemos que ya no está... y ahora sólo nos muestra la información de la sección general.

Ahora vayamos al Layout de la sección general. Aquí se encuentran por defecto todos los atributos presentes en la estructura de la transacción.

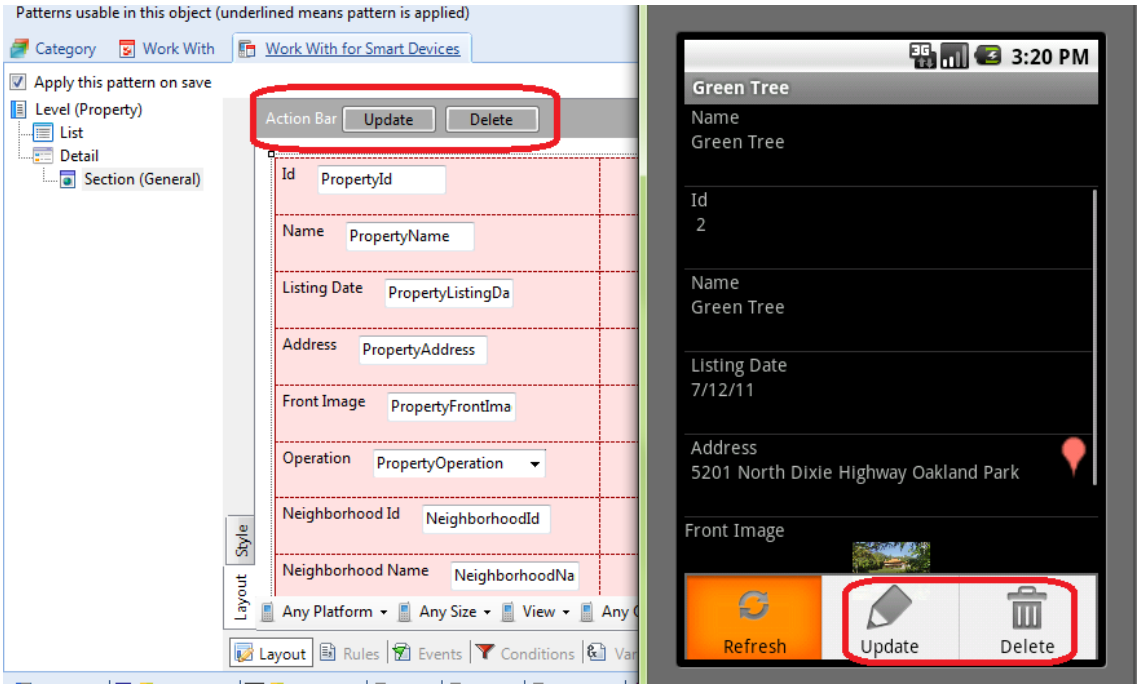
Para ser más exactos, estamos en el Layout de la sección general, en **modo View**.



Este es el layout que se utilizará toda vez que se quiera visualizar la información general de una propiedad. Por ejemplo, cuando hacemos tap sobre una de las propiedades de la lista de propiedades del vecindario… o cuando lo hacemos sobre el listado general de propiedades. {hacerlo en el emulador}



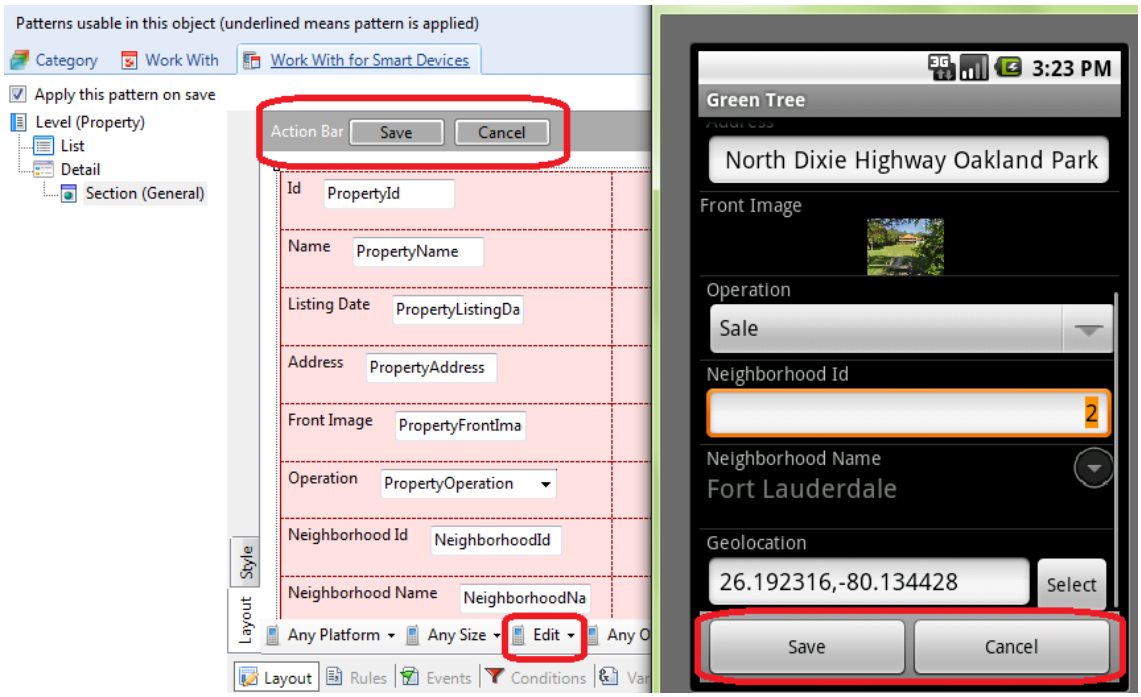
Desde este layout podemos modificar o eliminar la propiedad. {menú: mostrar Update y Delete} acciones que no casualmente se pueden ver aquí:



Si pensamos en la arquitectura, al abrir esta pantalla, desde el dispositivo se hará un request para ejecutar el servicio Rest (un data provider) que consultará la base de datos y devolverá un Json con la información. En el dispositivo se armará la pantalla a partir de esa información devuelta.

Si elegimos hacer una actualización de esta propiedad, ¿qué layout es este que se nos está ofreciendo, en el que podemos cambiar los valores? {mostrar date picker, el combo de operation, y los botones etc.?

Es el de la Section(General) sí, pero en su **modo Edit**. {cambiar de modo} Veamos que en este caso las acciones ofrecidas son para grabar o cancelar.



Por tanto estamos trabajando con dos layouts distintos (aunque estén inicializados con los mismos atributos).

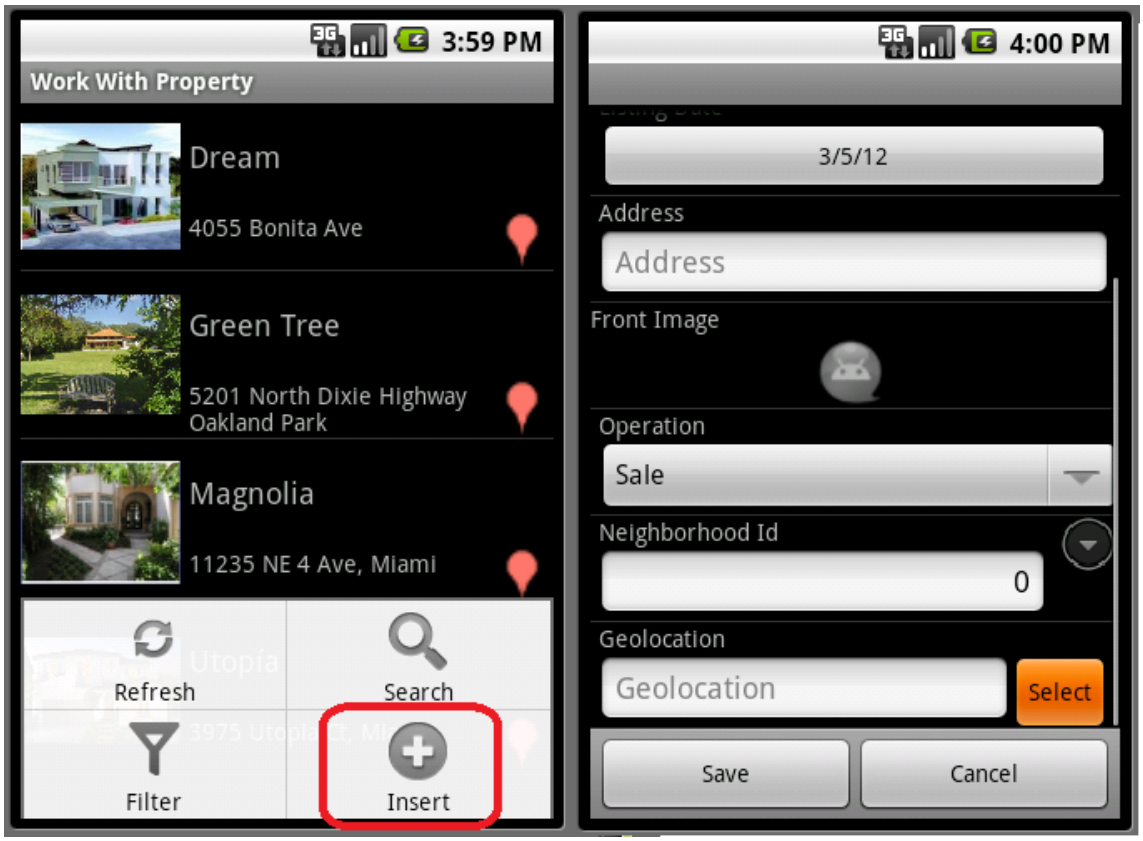
Podríamos querer que en modo Edit no aparezca el atributo inferido NeighborhoodName { quitarlo}… y que en modo View, en cambio, no aparezcan los identificadores, pues son internos al sistema {pasar a View y quitar los ids}. Y tampoco nos interesa mostrar la geolocation. {quitarla y elminar las filas sobrantes}

A través de la toolbox podemos insertar otros controles al layout.

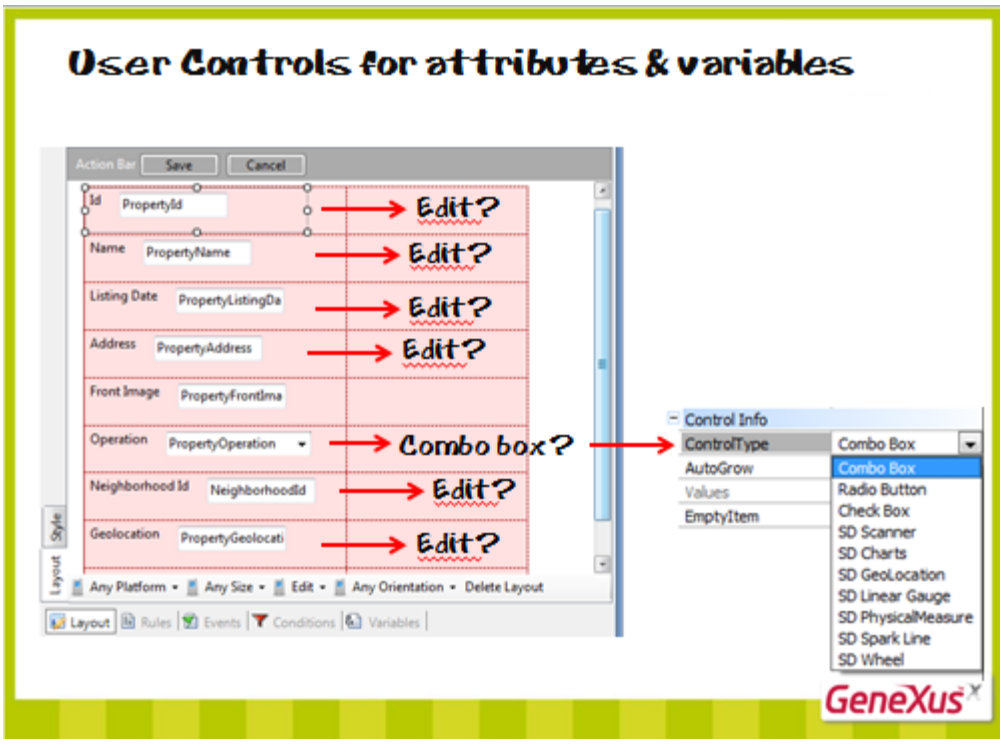
Veamos cómo nos quedó la información detallada de una propiedad en ejecución. F5…

Cuando se visualiza no están los ids ni geolocation… y cuando se edita, en cambio, éstos sí aparecen, pero el que no aparece es el nombre del vecindario.

La pantalla de detalle en modo edit también es utilizada cuando se inserta una nueva propiedad inmobiliaria:



Podemos pensar ese Layout como haciendo las veces del Form de la transacción. La diferencia será que este layout es implementado como una pantalla más del Work With y la manipulación sobre la base de datos se hará de forma transparente para el desarrollador, a través del Business Component asociado (que será un servicio Rest ejecutado en el Server).



También podemos configurar aspectos de la tabla, como lo hicimos con el grid…

…e incluso mostrar los atributos utilizando algún otro tipo de control para desplegar la información. Es decir, dependiendo del tipo de control del atributo o variable en el layout y de su tipo de datos, se ofrecerán user controls tanto para ingresar la información (en el layout Edit) como para mostrarla (en el layout View).

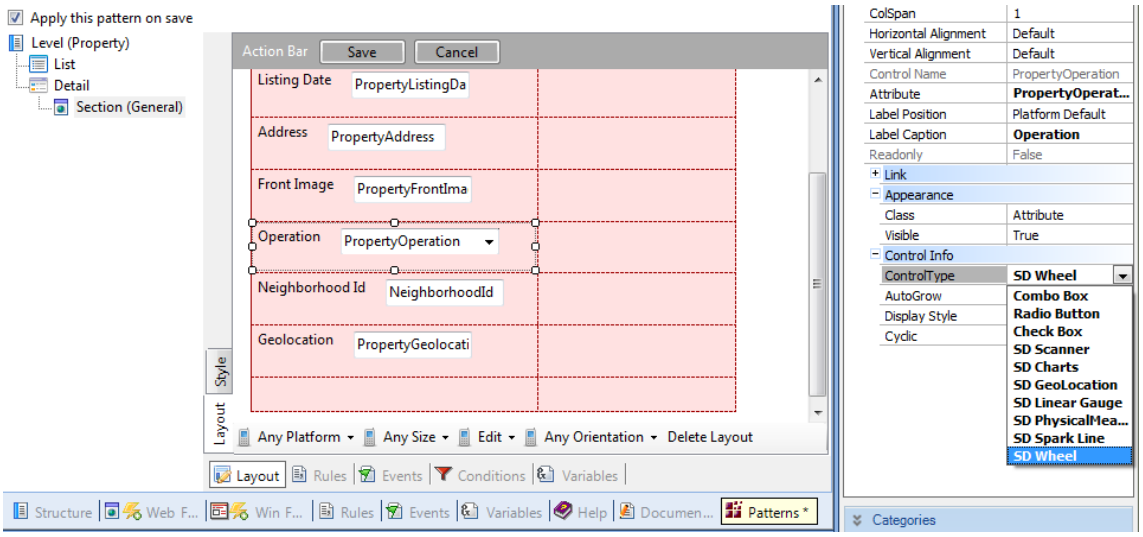
Por ejemplo, podemos dar a elegir al usuario la operación (venta o alquiler) utilizando, en lugar de un combo box, una ruedita, a través del user control Wheel…

{demo}

{hacerlo} {pasar a Edit}

Lo haremos para el ingreso de la operación, por tanto para el Layout en modo Edit.

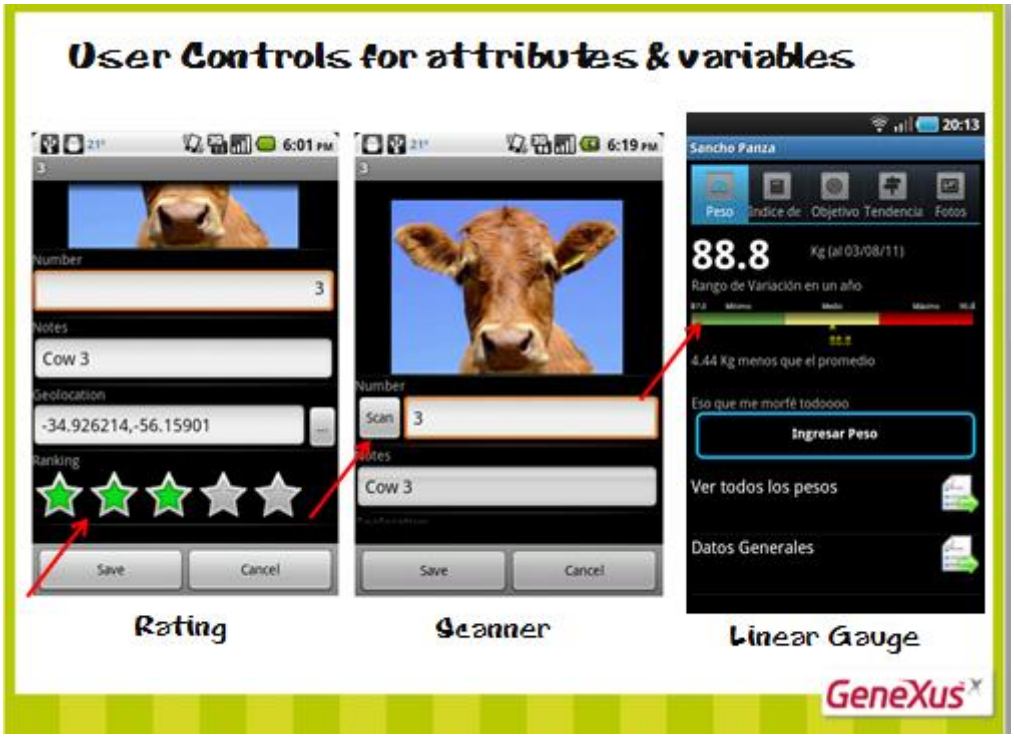
Aparecen ciertas propiedades a configurar…



Veámoslo en ejecución, F5… {transición}

Este caso tiene más sentido cuando se trata de varios valores y no sólo dos. También aplica a atributos numéricos, ofreciendo una ruedita con todos los dígitos. Es muy útil en los dispositivos touch screen, de manera de evitar la digitación vía teclado.

{transición a ppts}



Entre otras cosas, podemos utilizar un atributo numérico para que el usuario ranquee cierta información... para ello se creó el user control Rating.

O podemos hacer que el ingreso de un atributo se consiga escaneando un código de barras o un QR Code. Al configurar como tipo de control **Scanner**, se colocará un botón

Scan, al lado del atributo y al presionarlo se abrirá el lector que tenga instalado el dispositivo.

También puede necesitar mostrar la información de un atributo dentro de un rango determinado. Para ello utilizará el control **Linear Gauge**.

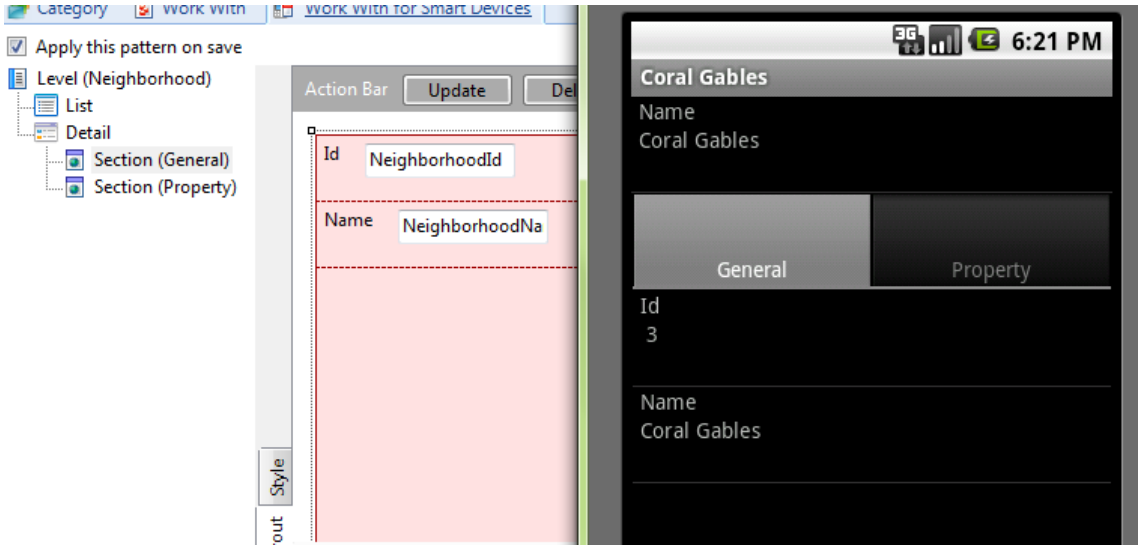
Estas y otras opciones se encuentran documentadas en nuestro wiki.

Cuando además de la sección general, se presentan secciones que corresponden a información relacionada, ¿cómo presentar toda esa información?

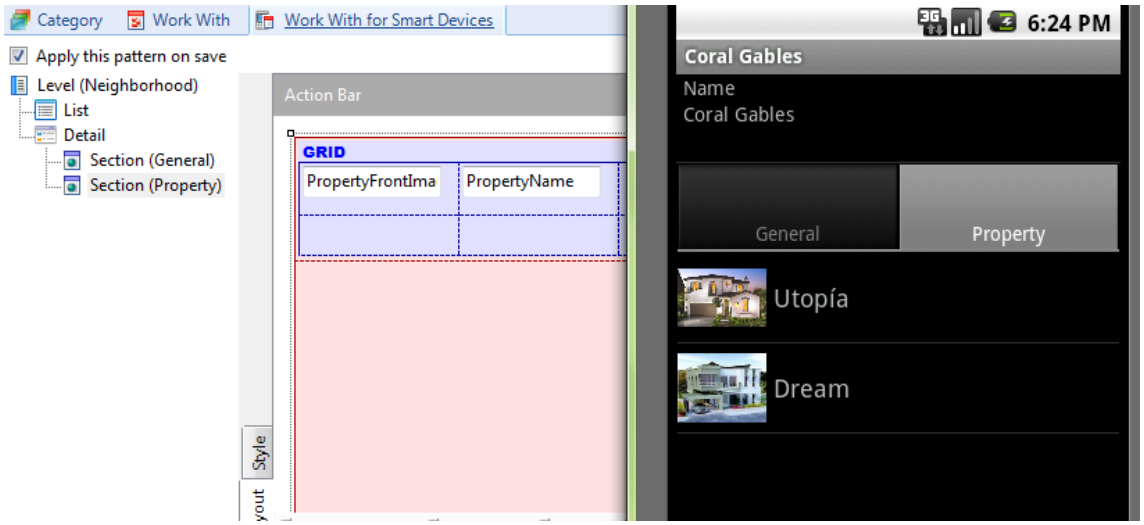
Como vimos en el caso del vecindario, por defecto para Android se muestran dos pestañas (tabs), cada una con la información de cada sección. Esta forma de presentación dependerá de cada plataforma.

El layout de cada sección, así como su comportamiento, se personalizan por separado… Así por ejemplo… {demo}

Aquí …… personalizamos el layout de esta pantalla {mostrar layout de section(General) viendo a la vez el emulador)



Y {moverse en el emulador al tab Property} aquí, personalizamos la información de esta otra pantalla, que muestra las propiedades inmobiliarias de **este** {señarlar el nombre arriba en el emulador} vecindario.



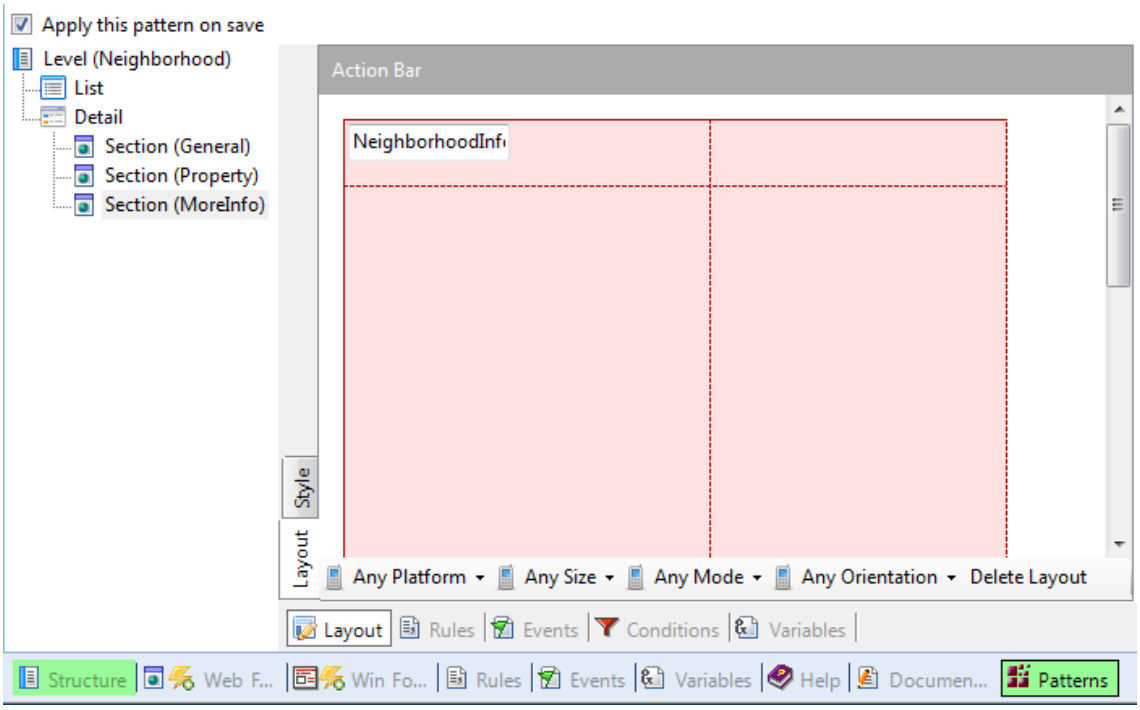
¿Cómo sabe el programa ejecutado cuál es el vecindario en cuestión? Porque es recibido como parámetro en cada sección. {mostrar regla parm en c/section}, y lo es en el detail…

Podemos personalizar el layout de esta sección al igual que lo hicimos antes con el List de propiedades inmobiliarias, por ejemplo pidiéndole que muestre las propiedades en un mapa, en lugar de como lista.

Si ahora agregamos el atributo *NeighborhoodInfo* a la transacción Neighborhood, de tipo Varchar(1024), de manera de guardar información general del vecindario…

…y queremos mostrar esa información en una sección independiente, eliminamos el atributo automáticamente colocado en la Section(General) y creamos una nueva sección, a la que llamamos por ejemplo 'MoreInfo', que por defecto aparece con el Layout vacío y en "Any Mode" (es decir, esta pantalla será la utilizada tanto en Edit como en View).

… e insertamos allí el control atributo *NeighborhoodInfo*. Le quitamos la etiqueta y especificamos que el campo pueda crecer de acuerdo a su contenido.



En la sección de reglas le asignamos regla **parm**, para que automáticamente filtre por el vecindario.

Especificamos el Caption que se verá en ejecución … {Info} y también podemos asociarle una imagen, así como a las otras secciones. {hacerlo}

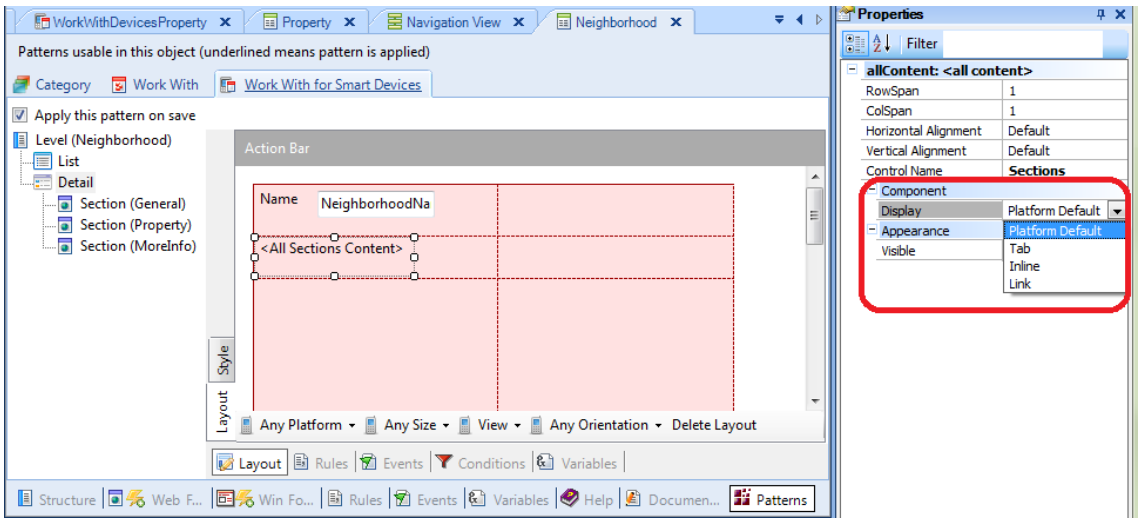
Veámoslo en ejecución. {(Acá hay que compilar para que se vean las imágenes)}…

Una vez asignada la información a los vecindarios…

Pero podríamos querer modificar esta forma de presentación de las secciones en pestañas, para que se presente de manera distinta. ¿Dónde hacerlo, qué posibilidad me brinda cada plataforma?

Vayamos al nodo Detail…

Este layout tiene un control contenedor <All Sections Content> que será donde se cargarán las distintas secciones, con el tipo que le especifiquemos en la propiedad Display, que por defecto está con el valor “Platform Default”. Esto se debe a que dependiendo de la plataforma, será la forma más adecuada de mostrar la información (está relacionado al tamaño de la pantalla).



Vemos que aparecen 3 opciones: **Tab** corresponde al default de Android y de BlackBerry, y es el que establece que cada sección se muestre en una pestaña. **Inline**, significa que cada sección se presente en la misma pantalla, una a continuación de la

otra y **Link** significará que cada sección en lugar de presentarse en la pantalla del detail, se presente en pantallas independientes a través de links.

El default de iOS será para secciones con grid mostrarlas como links y las demás inline.

Con esta propiedad {señalar la detail} y dentro de las posibilidades de cada plataforma, podría variar la opción por defecto para que todas las secciones se presenten uniformemente de otra manera. Por ejemplo, en Android podría lograr que todas se mostraran como Links.

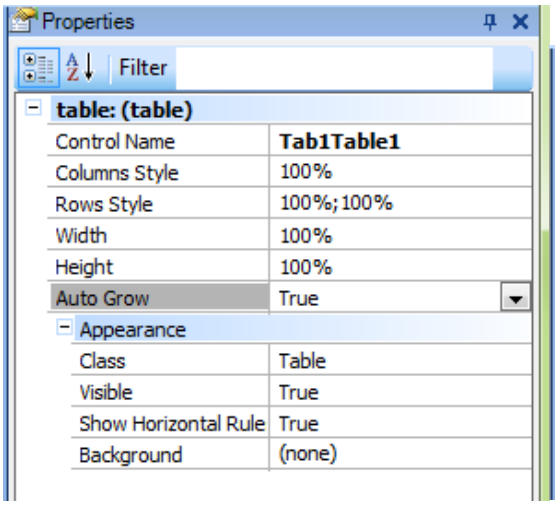
Pero si queremos por ejemplo sólo mostrar la información General y la More Info dentro de tabs, y las propiedades inmobiliarias del vecindario mostrarlas dentro de la información general como un Link, entonces no nos sirve utilizar esta propiedad que aplica a todas las secciones uniformemente. {callout con imágenes que representen en ejecución lo que digo}

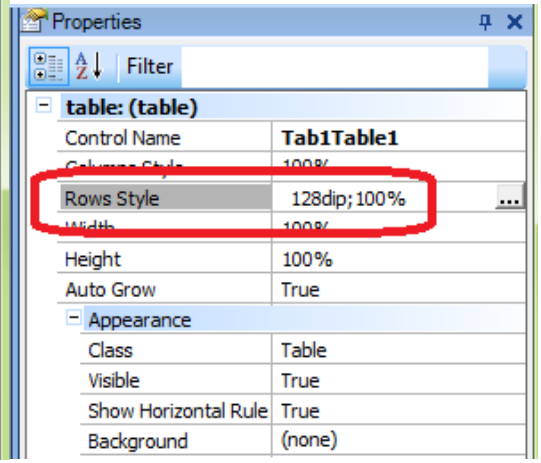
Borremos este control. Aparece en la toolbox para poder reinsertarlo, pero además, aparece un control por cada sección del Detail, para poder insertarlas independientemente de las otras e independizar así su comportamiento.

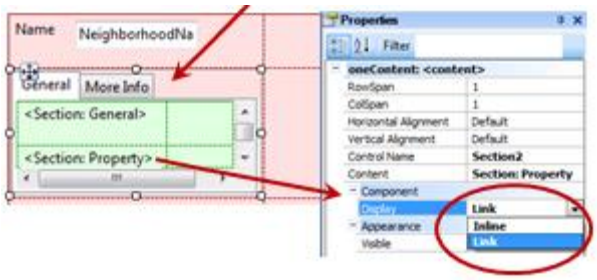
Por ejemplo, podemos insertar un tab control de la toolbox y en la primera página colocar la <Section General> … {hacerlo} y la <Section Property> {hacerlo}. Entre las propiedades de este ítem configuramos el Caption {General} y le podemos asociar una imagen. {hacerlo} En la segunda página del tab control podemos colocar la <Section MoreInfo> {hacerlo, Caption= More Info y también agregarle imagen}. El tercer tab lo eliminamos. {hacerlo}

{ir a la primera tab page}

En la tabla modifiquemos el tamaño delas filas para que entre toda la información:





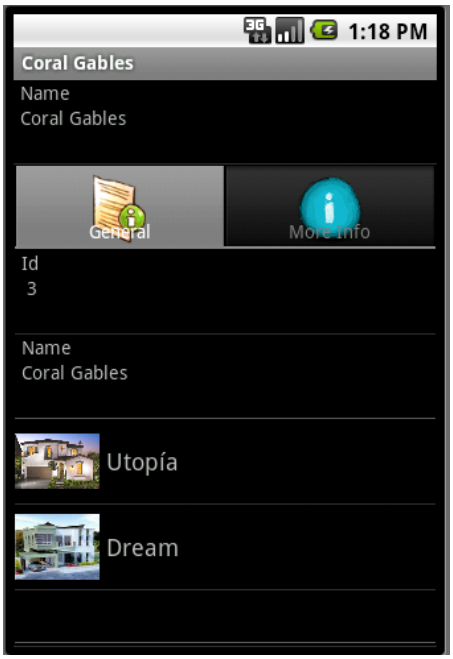


Observemos que la propiedad **Display** de las secciones individuales asume 2 valores en lugar de 3:

- Inline (default)
- Link

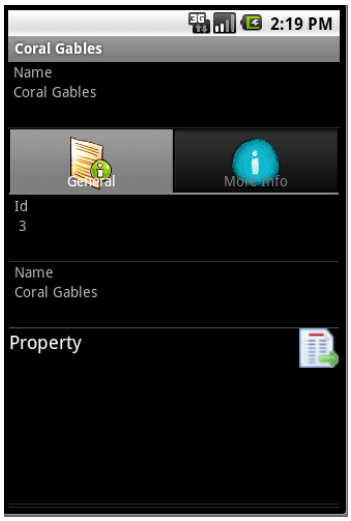
Por ejemplo, en el caso de <Section Property>, refiere a cómo queremos que aparezca la lista de propiedades asociadas al vecindario que se está visualizando:

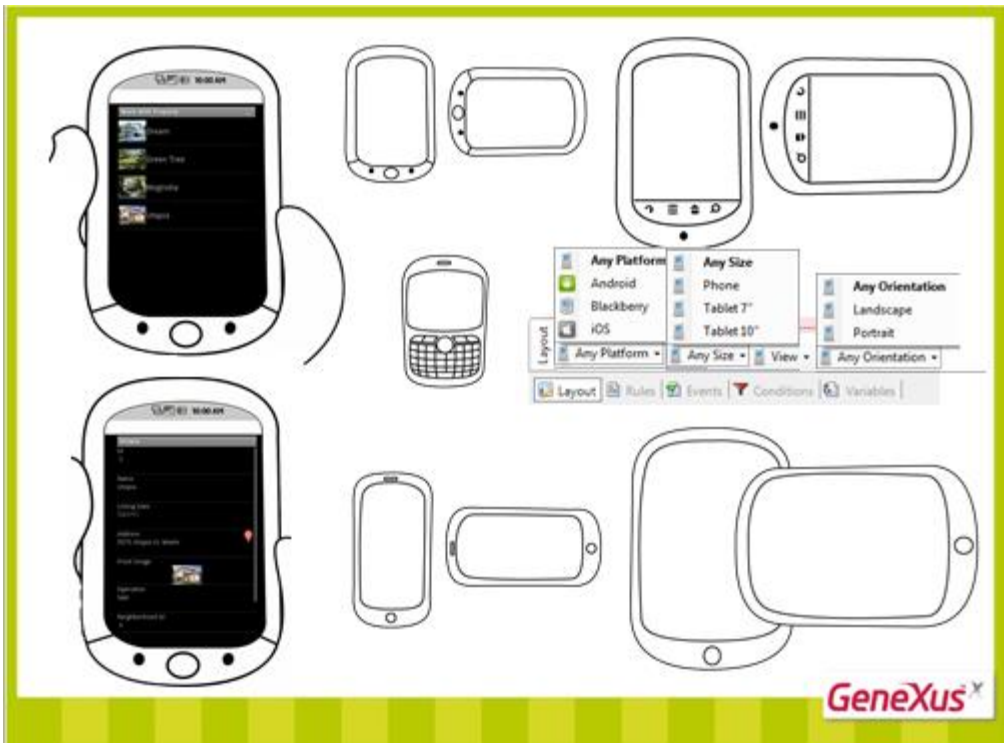
1. en el mismo form, como líneas bajo la información plana general (veámoslo en ejecución)… {mostrar en ejecución, con callout } ó



(callout TabcontrolCon2SectionsInline.jpg)

2. como un link, que es lo que queremos. {elegir esta opción}





Hemos diseñado las pantallas del List y el Detail teniendo en mente un Android Phone, que tiene un tamaño de pantalla determinado. Pero podemos querer variar este diseño si la pantalla es la de una tableta (porque allí entra más información). Y en este caso también de acuerdo a si es de 7 pulgadas o 10.

O podemos querer variar la pantalla de acuerdo a la plataforma, para seguir las guidelines de cada una,

…o incluso diseñar diferente las pantallas cuando se van a visualizar verticalmente (portrait) u horizontalmente (landscape).

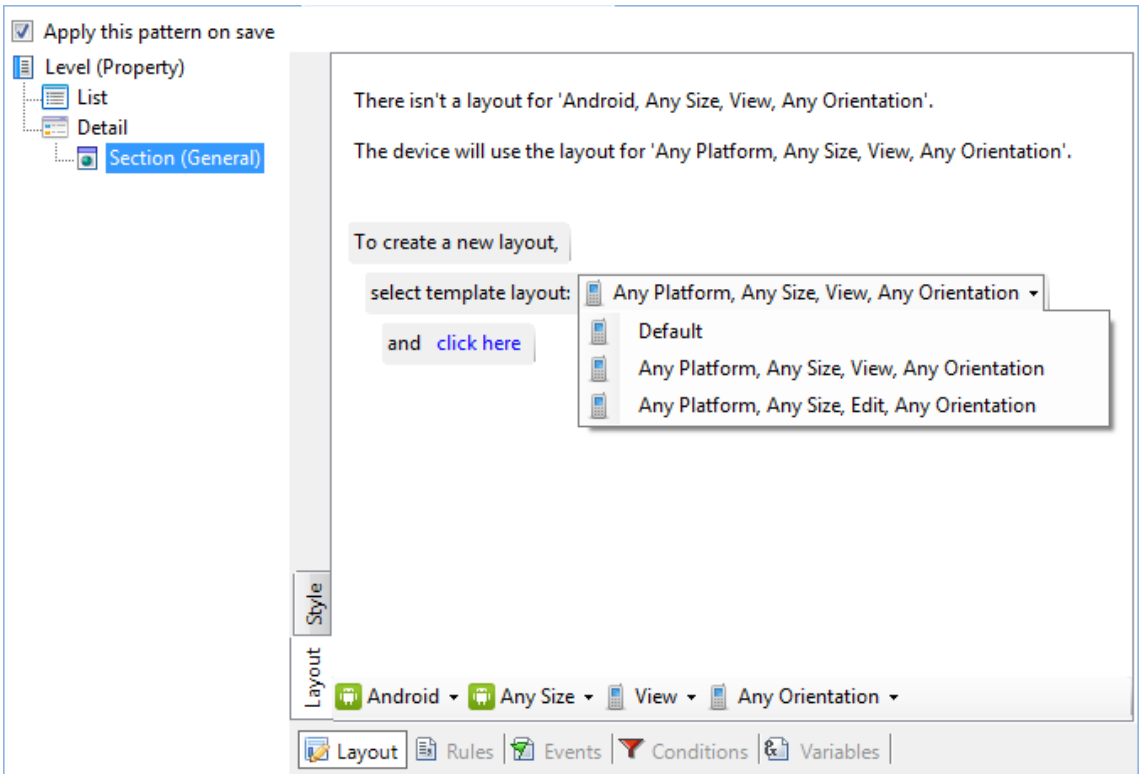
¿Cómo hacemos para considerar estas variaciones?

Para ello contamos con combos en el borde inferior de cada Layout.

Uno para elegir la plataforma, otro para elegir el tamaño, y otro para la orientación.

La opción por defecto es Any Platform, Any Size, Any Orientation, lo que significa que tendremos un sólo Layout para View y uno para Edit, que valdará para todas las plataformas, todos los tamaños y todas las orientaciones.

{demo → mostrar abierta Section(General) de WWProperties}



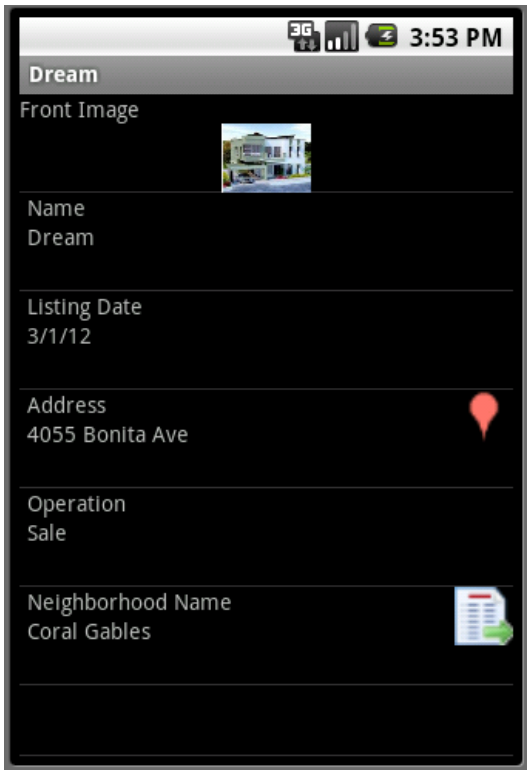
Pero si creamos uno específico para una plataforma determinada, por ejemplo, para Android, nos aparecerá un mensaje como el que vemos arriba, que nos indicará que no existe ese Layout, y nos da para elegir si inicializarlo vacío, (Default) {hacerlo} o (borro este layout) {hacerlo} de acuerdo a la pantalla de Edit {hacerlo} (que no tenía NeighborhoodName), o {borrarlo} a la de View (que no tenía los identificadores ni la geolocation). Observemos que ahora nos muestra las etiquetas arriba, tal como es el estándar de Android.

Cambiemos, por ejemplo, para el layout del view en Android, el lugar donde se muestra la imagen. {llevarla para arriba}

Tenemos en este momento dos Layouts diferentes definidos para el modo View. {elegir Any Platform y luego vovler a Android}

¿Cuál se utilizará en la aplicación generada? Cuando se genere para Android, este que vemos, y cuando se genere para cualquier otra plataforma, éste. {pasar de uno a otro}

Veamos lo que mostraba el emulador de Android antes del cambio {mostrarlo}, y en la próxima generación… F5 {mostrarlo}



Una misma aplicación probablemente requerirá un look & feel diferente al variar la plataforma.

Así, esta agenda de un Encuentro de Usuario de GeneXus, presenta diferencias de diseño de acuerdo a la plataforma escogida.

Incluso sin cambiar la plataforma podemos querer asociar cierto diseño a una aplicación desarrollada para un cliente determinado, y otro para la misma aplicación pero para otro cliente.

Necesitamos separar los aspectos de diseño de los de funcionalidad, de manera tal de poder paralelizar las tareas, y dejar las de diseño a un especialista y concentrarnos como desarrolladores en construir la aplicación.

¿Cómo?

Continuará.

FIN VIDEO 2

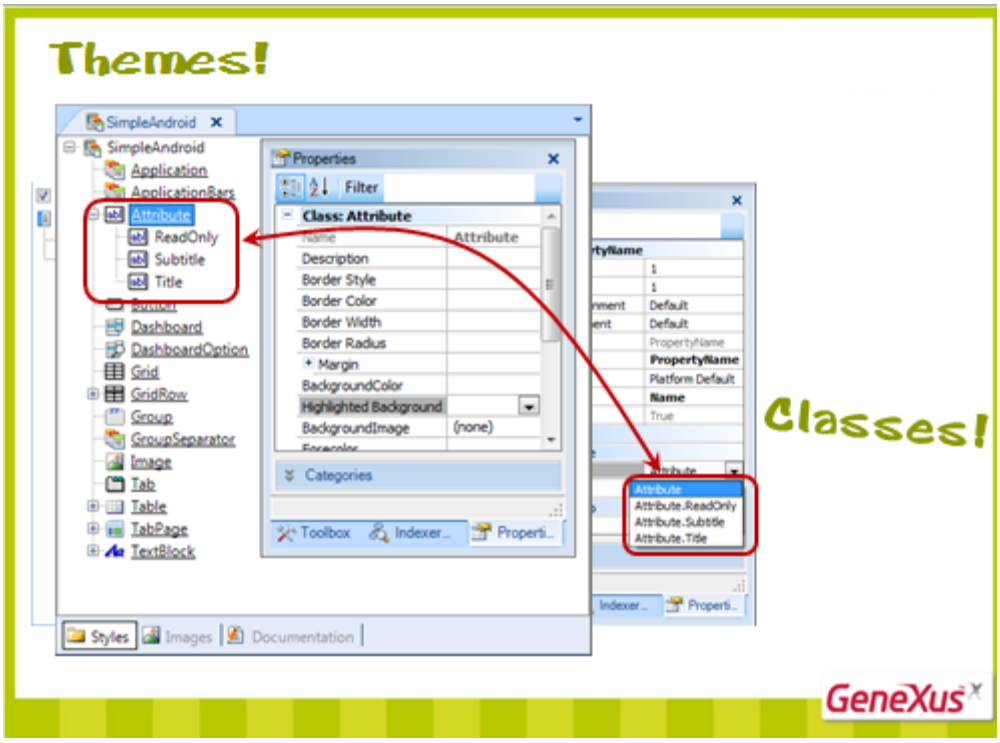
Video 3: Diseño a través de clases y Themes

Una misma aplicación probablemente requerirá un look&feel diferente al variar la plataforma.

Así, esta agenda de un Encuentro de Usuarios de GeneXus, presenta diferencias de diseño.

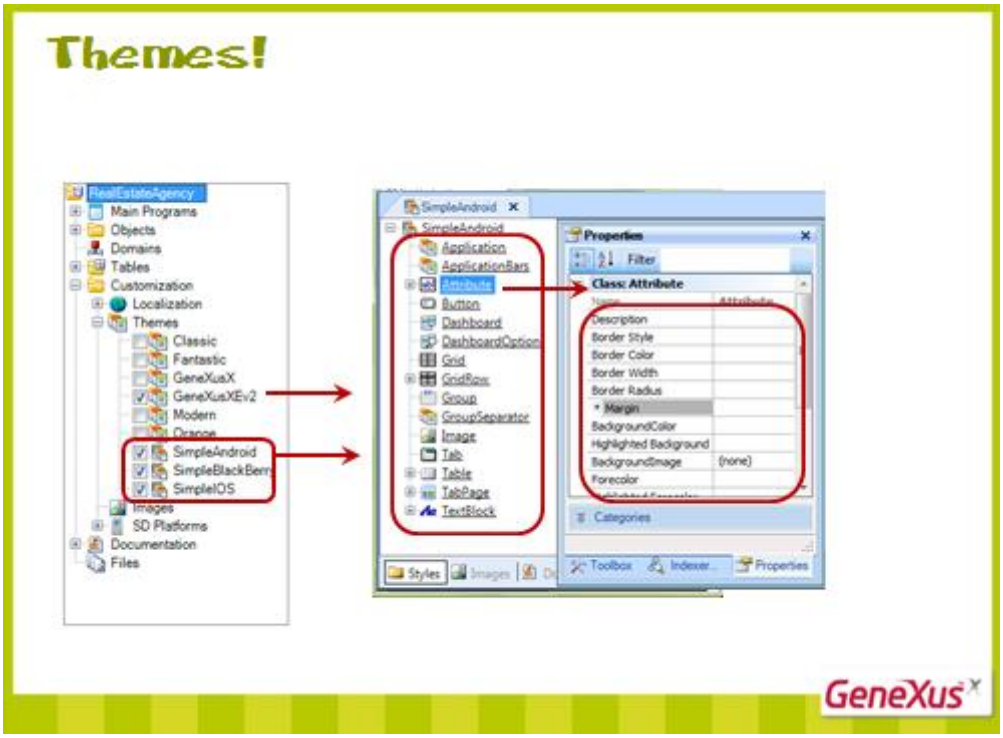
Incluso sin cambiar la plataforma podemos querer asociar cierto diseño a una aplicación desarrollada para un cliente determinado, y otro para la misma aplicación para otro cliente.

Para separar los aspectos de diseño de los de funcionalidad, de manera tal de poder paralelizar las tareas, y dejar las de diseño a un especialista y concentrarnos como desarrolladores en construir la aplicación, a los controles de los Layouts se le asocian “clases” y luego existen los objetos themes (temas) que concentran las especificaciones de diseño de cada una de esas “clases”.



Así, a un control atributo se le puede asignar una de estas cuatro clases, que están configuradas en un lugar centralizado, en este caso para esta plataforma.

El lugar centralizado es el objeto Theme.

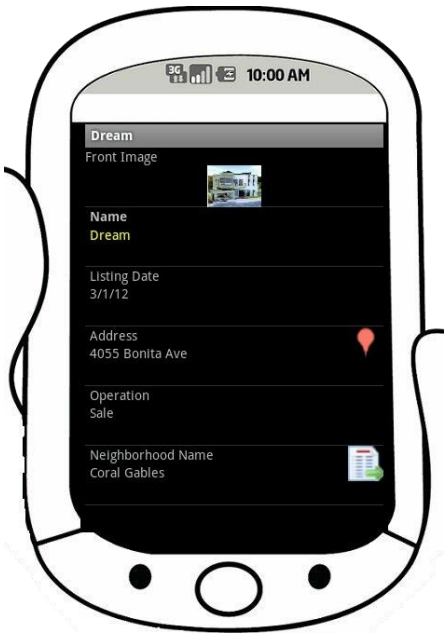


Por defecto, además del Theme default para la aplicación web…

existen tres themes para Smart Devices, uno por cada plataforma.

En cada Theme se configuran las propiedades de cada clase (de control, de usuario, etc.)

Por ejemplo, supongamos que deseamos que el atributo PropertyName salga en negrita y amarillo en Android.



Para ello…

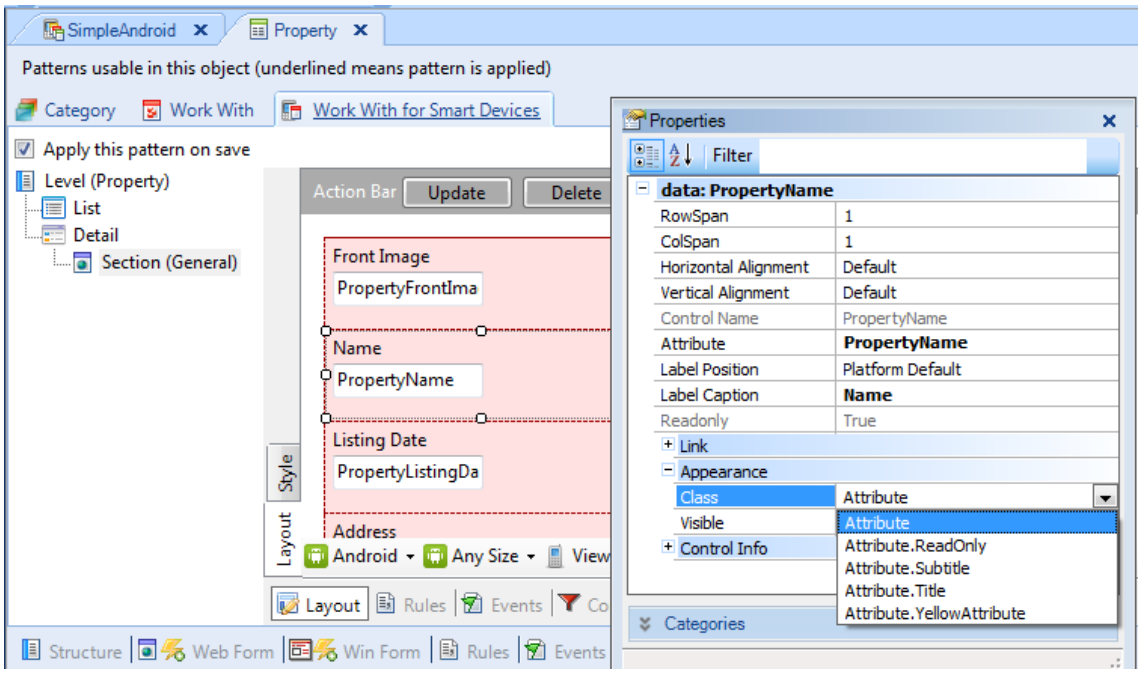
{demo}

Abrimos el theme por defecto para Android. Creemos una nueva clase de atributo {hacer AddClass} de nombre YellowAttribute.

Configuramos la propiedad Forecolor en yellow y la Label class Textblock.Title que corresponde a esta otra clase {mostrar abajo la class TextBlock} a la que le ponemos negrita.

{grabar}

Ahora vamos al layout del view para Android y cambiémosle la clase al atributo PropertyName, asociándole la nueva.



Ejecutamos… {mostrar}

¿Dónde se configura qué theme utilizar?

Es en las Preferences {ir} Aquí están centralizados los valores generales default de todas las instancias del pattern Work with para Smart Devices.

Entre las cosas que se pueden configurar, se encuentran los themes.

La plataform AnyAndroid tiene configurado el theme que acabamos de modificar.

En cambio Blackberry tiene… e iOs tiene el tercer theme que habíamos nombrado.

Podemos modificar el theme de cada uno.

Las precedencias son las usuales: Si estamos generando para iPad, como no tiene un theme definido, utilizará el de AnyIOS.

FIN VIDEO 3

Video 4: Work With for Smart Devices – Órdenes, búsquedas y filtros

Habiendo visto en otro lado cómo personalizar los Layouts, estudiemos cómo personalizar los órdenes por los que la información es desplegada, y los searchs y filtros que se ofrecen al usuario…

{demo, mostrar ejecución}

Si observamos el List de propiedades inmobiliarias, se está ordenando la información desplegada por nombre de la propiedad {pasar el mouse por la letra inicial del nombre de c/propiedad}. Además se está permitiendo filtrar la información por fecha {cliquear} dentro de un rango, por operación {cliquear} y por vecindario {cliquear} (como es clave foránea (foreign key), abrirá una lista de selección).

El Search luego veremos por qué atributos se puede realizar.

¿Dónde se configura todo esto?

En la sección conditions de la instancia del pattern.

Existe un sólo orden definido, de nombre Name {señalar la propiedad Name}, compuesto por el atributo PropertyName (elegido por ser el description attribute de la transacción).

Podemos ver que el Search está definido por PropertyName y por PropertyAddress {ir a ejecución} Así, puedo filtrar por dirección, {poner Bonita} o por nombre de la propiedad {poner Uto}

Y bajo **Advanced Search** podemos ver los atributos por los que podíamos filtrar.

Todo esto es personalizable. Por ejemplo, si deseamos que el usuario pueda elegir entre ordenar las propiedades inmobiliarias por Nombre o por Vecindario, entonces debemos agregar un nuevo orden {hacer Add Order y nombrarlo Neighborhood} compuesto por {hacer Add Attribute} NeighborhoodName.

Apliquemos el cambio. F5…

{transición para mostrar en ejecución}

Se ha agregado Order a esta acción. Está seleccionado ordenar por nombre, pero el usuario puede elegir ordenar por {cliquear} vecindario…

Si queremos que al elegir esta opción, nos aparezcan agrupadas las propiedades por vecindario, entonces… {volver a GX}

…especificamos que realice un corte de este orden…{marcar Break by true} por el atributo del order.

{ejecutar}

Y si queremos que además de agrupar por vecindario, ordene dentro del vecindario por nombre de la propiedad…

{hacer Add Attribute y agregar PropertyName} acá es importante el orden en que aparecen listados estos atributos (no es lo mismo ordenar por vecindario y dentro de vecindario por nombre, que al revés). Nos falta indicar que el agrupamiento sólo queremos que lo realice desde el primer atributo del order hasta el atributo… Vecindario (en este caso es el mismo). {hacerlo}

{ejecutar} Vemos que ahora está ordenando dentro de vecindarios.

{volver a GX}

Podemos agregar o eliminar atributos de búsqueda {hacer botón derecho sobre Search y Add], y lo mismo para las búsquedas avanzadas, que son las que aparecen como

Filter en ejecución y que tienen como agregado que se especifica la expresión que determina el tipo de filtro a realizar {pasar por los atts y mostrar la propiedad Expression}. Estas variables internas {mostrarlas en el Expression} corresponden a los controles que aparecen para que el usuario ingrese valores {mostrar en ejecución}…

Pero además, tenemos la sección de Conditions generales, que establecen condiciones sobre la información que no se presentarán al usuario para que éste elija. Si queremos que siempre salgan listadas las propiedades que están en venta, establecemos entonces que las propiedades que liste, cumplan siempre esta condición (además de las otras que elija el usuario a través del Search y Advanced Search)…

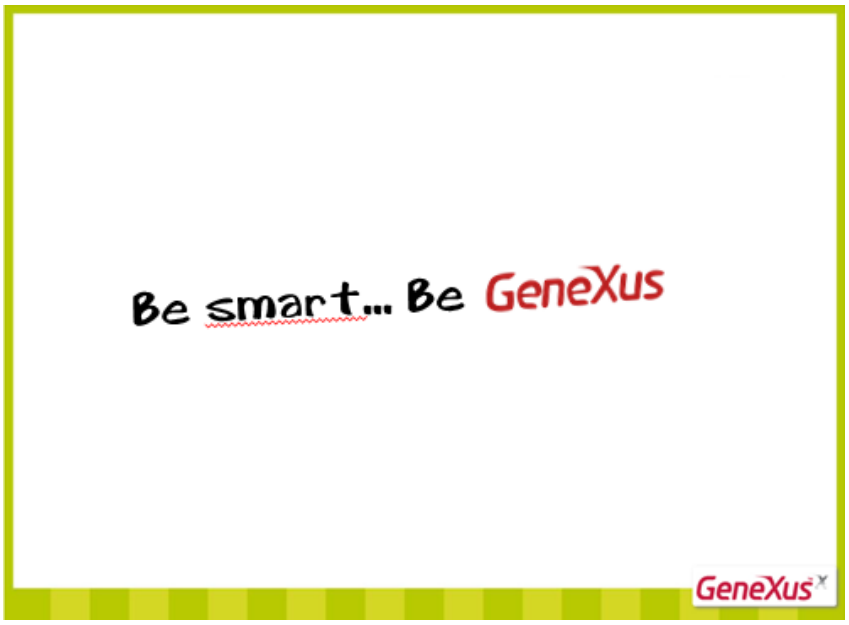
{agregar PropertyOperation = Operation.Sale;)

El Detail está más limitado, puesto que la información que se muestra por lo general no es repetitiva. {pasar por el detail}

¿Desea programar comportamiento?

Continuará…

FIN VIDEO 4



Video 5: Work With for Smart Devices – Comportamiento a través de eventos

Habiendo visto en otro lado cómo personalizar los Layouts, los órdenes por los que se lista la información, las búsquedas y los filtros, ahora veremos cómo personalizar el comportamiento, a través de Eventos.

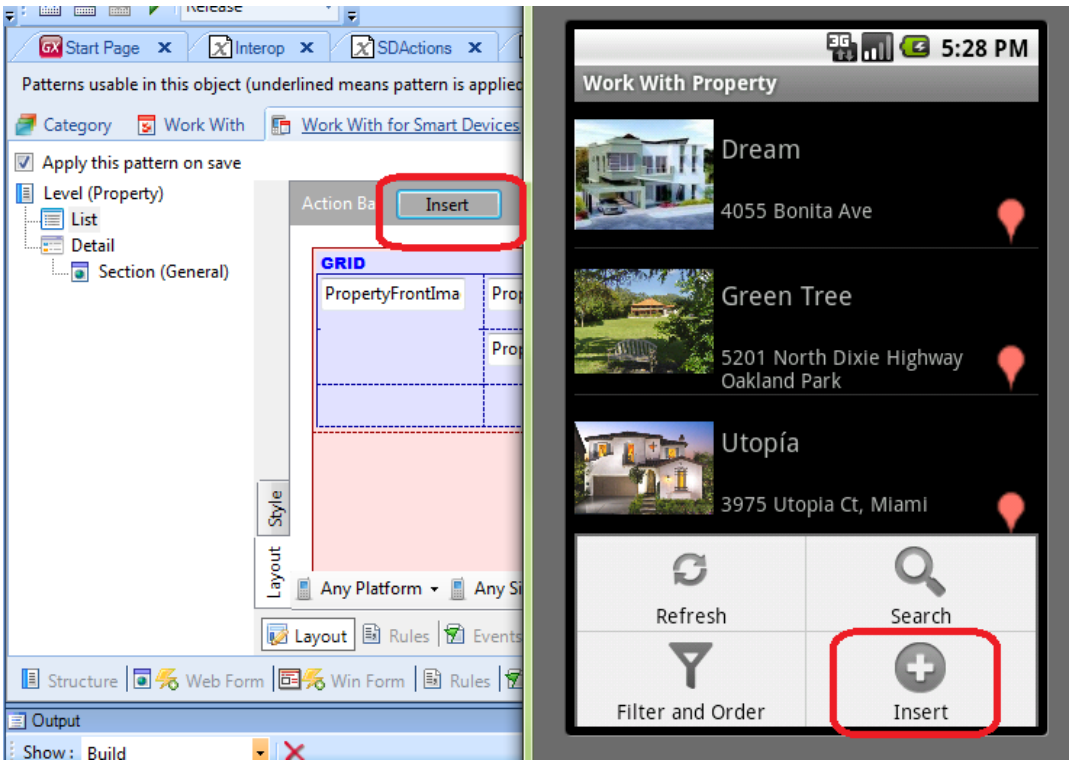
Existen ciertas acciones predefinidas para cada pantalla. Por ejemplo, cuando estamos en el List, tenemos en el **menú** un botón para insertar un nuevo elemento en la lista.

En la pantalla de Detail, dependiendo de si estamos en modo Edit o en View, tendremos ciertas acciones predeterminadas. Por ejemplo, si estamos en View, podemos ver en el menú las acciones Update y Delete, que nos llevarán a la misma pantalla, pero en modo Edit.

A las acciones en GeneXus las conocemos como eventos.

{demo}

Empecemos por el List. En la Action Bar se encuentran las acciones que se agregarán al menú del dispositivo. {mostrar el emulador sobre el IDE}



{cliquear sobre Insert en la action bar, ir a las propiedades, y en silencio mostrar arriba del todo, donde dice **action:Insert**, detenerse allí un segundo, luego el name Insert y luego Class y decir ahí lo que sigue: }

Se trata de un botón.

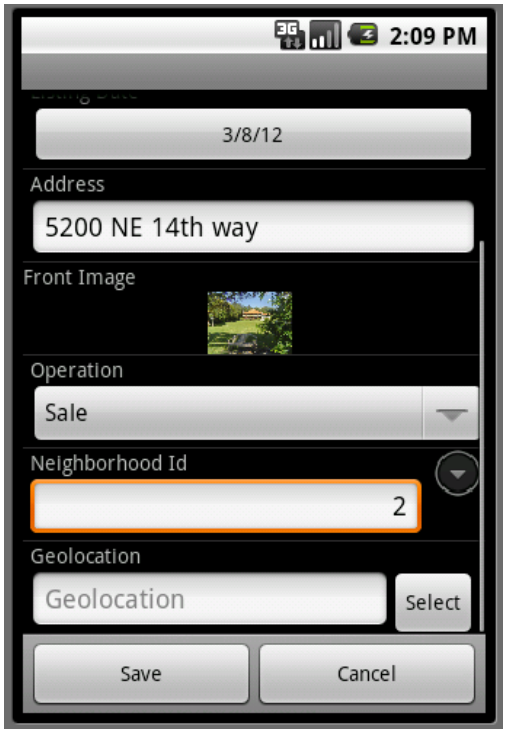
Al presionarlo, {presionar en el emulador} abre la pantalla del Detail en modo Edit, para que el usuario ingrese una nueva propiedad inmobiliaria. ¿Dónde está esto programado? {volver a GX}

En la sección de Eventos. {abrirla}

Aquí vemos el Evento asociado y el código que se ejecutará toda vez que este evento se produzca. Se invocará al WorkWith for Smart Devices de la transacción Property, level Property {mostrar con el mouse} y Detail {mostrar con mouse} , en modo Insert {mostrar sobre código de evento}, sin pasar parámetros, lo que significa que se abrirá el Layout de Edit… {mostrarla, para eso primero pasarse al Layout, luego cambiar al nodo Section(General) y ahí elegir Edit. Mostrar enseguida la pantalla abierta del emulador que quedó en esa misma pantalla}

{volver al Layout}

En esta pantalla de Edit, que corresponde a la Section(General), también podemos ver dos botones en el menú. Una vez cargados los datos de la nueva propiedad {cargar nombre MyProperty…} querremos grabarla, Save {señalar} o por el contrario, cancelar {señalar} y volver a la pantalla anterior. {presionar esto}



{pasar a GX} En la Action Bar vemos esos dos botones. {presionar sobre Save y mostrar en las propiedades arriba del todo action:Save y hacer lo mismo sobre Cancel}.

¿Qué esperaríamos encontrar como código del evento Save? Estamos en el dispositivo, {volver al dispositivo con todo cargado} tenemos toda la información ingresada por el usuario para esta nueva propiedad inmobiliaria, al hacer Save {mostrar botón} tendríamos que enviársela al servidor (en nuestro caso en la nube) {callout con server en la nube} para pedirle que ejecute la lógica de la transacción Property con esos datos {callout con trn}, creando un nuevo registro en la base de datos {callout con DB} para esa propiedad inmobiliaria. Es decir, tenemos que pedirle que ejecute el BC Rest {hacer Save}y que nos devuelva si la operación fue o no exitosa, de modo que le podamos informar al usuario del dispositivo. {mostrar el texto que sale} Pero toda esta lógica, que depende de la plataforma del dispositivo {ir a GX, y editar los Events}, está encapsulada en una API provista por GeneXus, SDActions {señalarla}. Al invocar a su método Save() {señarlarlo} se hará lo que esperamos.

Cuando el usuario presiona Cancel, queremos volver a la pantalla de la que partimos. Por ejemplo, {ir a emulador} si estamos queriendo hacer un Insert… {desde el List hacer Insert y allí cancelar} volver al List. Y si estamos queriendo hacer un update {ingresar a una propiedad con tap y luego presionar Update} desde el View de la propiedad, {presionar Cancel} volver a ese View. {volver a GX} Esto es lo que implementa el método Cancel() de la Api.

Por otro lado, en el View tenemos los botones Update y Delete {mostrarlo en el emulador en la ventana que quedó abierta}. Esos botones podemos verlos en la Action Bar del layout correspondiente {movernos en GX al View}

Al elegir Update, el comportamiento esperado es que abra el Layout Edit, para que así el usuario pueda modificar la información de esa propiedad. Veamos el código del evento {volver a GX, hacer botón derecho, Go to Event} Se está invocando al Detail, del WorkWith de Property {señalar}, level property {señalar}, en modo Update, pasándole como parámetro el PropertyId recibido como parámetro. De esta manera entiende que tiene que abrir el Layout Edit, instanciando los valores de la propiedad inmobiliaria con ese identificador. {mostrar el emulador. Luego hacer Cancel }

Por último, si presionamos Delete, {hacerlo} también queremos que se invoque al Detail, pero esta vez en modo Delete, de manera que el usuario pueda elegir si eliminar la propiedad {señalar botón}, o cancelar la operación {señalar botón}.

{ver en GX, señalar invocación al WW dentro de evento}.

¿Por qué aparece este return? Porque estando en el View {vovler al view} al eliminar el registro que estábamos viendo, desde este botón {mostrar Menú/Delete} querremos eliminar este registro que estamos visualizando, por lo que enseguida de eliminarlo, querremos volver al que llamó a este View en primera instancia. {hacerlo}

{ir a GX, código de evento Delete} Aquí podemos ver que tenemos dos comandos consecutivos dentro del evento. Cuando dentro de un evento ejecutado en el dispositivo necesitamos hacer dos invocaciones, tendremos que utilizar el comando Composite. Lo estudiaremos luego.

{ppts}

Acabamos de ver eventos predeterminados tanto para el List como para el Detail, programados automáticamente por el pattern.

Pero podemos definir otros eventos, colocarlos en diferentes lugares del Layout (no sólo en la Action Bar para que aparezcan en el menú) y utilizar diversos comandos para programar el comportamiento deseado.

Tenemos dos tipos de eventos: los del sistema, cuyo código se ejecutará en el servidor,

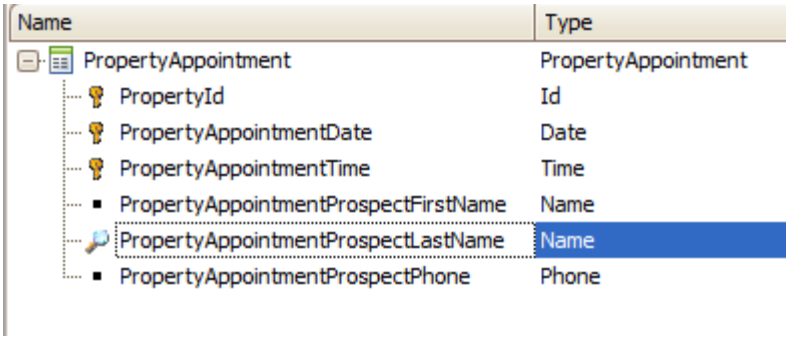
y los de usuario, cuyo código se ejecutará en el dispositivo (independientemente de que pueda establecer comunicaciones con el servidor a través de invocaciones a servicios rest).

Los del servidor los dejaremos para más adelante. Hemos visto algunos del cliente, predefinidos,

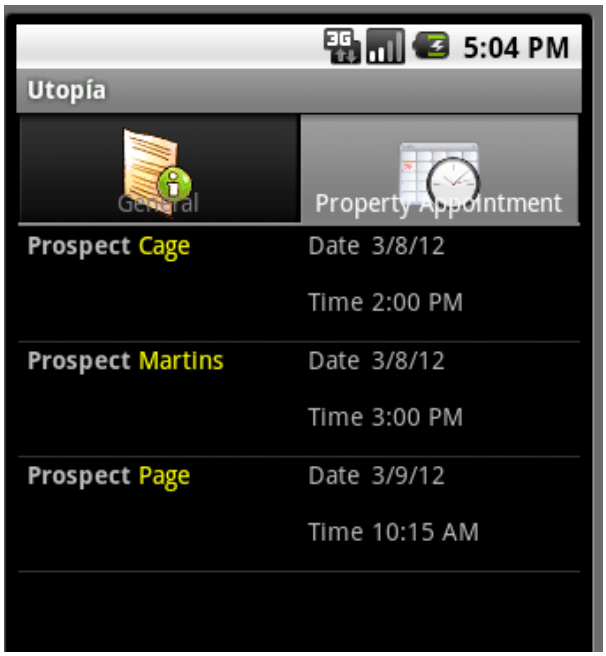
y ahora definiremos alguno nuestro…

{demo – agregar trn PropertyAppointment, marcar el pattern, personalizar el grid, hacer F5 y reorg y cargar algunas visitas para alguna propiedad }

Hemos agregado una transacción {señalar el nombre} para agendar visitas a las propiedades inmobiliarias {señalar PropertyId}, en una fecha {señalar } y una hora {señalar }determinadas, de posibles clientes, de los que almacenamos su nombre {señalar }, apellido {señalar }y teléfono {señalar }.



Le hemos aplicado el pattern work with {mostrar que está marcado}, con lo cuál se ha agregado automáticamente al Work With de Property {pasarse a ese} una Section que mostrará las visitas: apellido del prospect, fecha y hora {señalar} asociadas a la propiedad que se esté visualizando {mostrar regla parm}. Le asociamos imágenes a las secciones y mejoramos un poco el aspecto del grid, para que en ejecución se vea… así…{mostrar en ejecución el dashboard, de ahí el list de propiedades y de ahí elegir una propiedad y ver las appointments}:



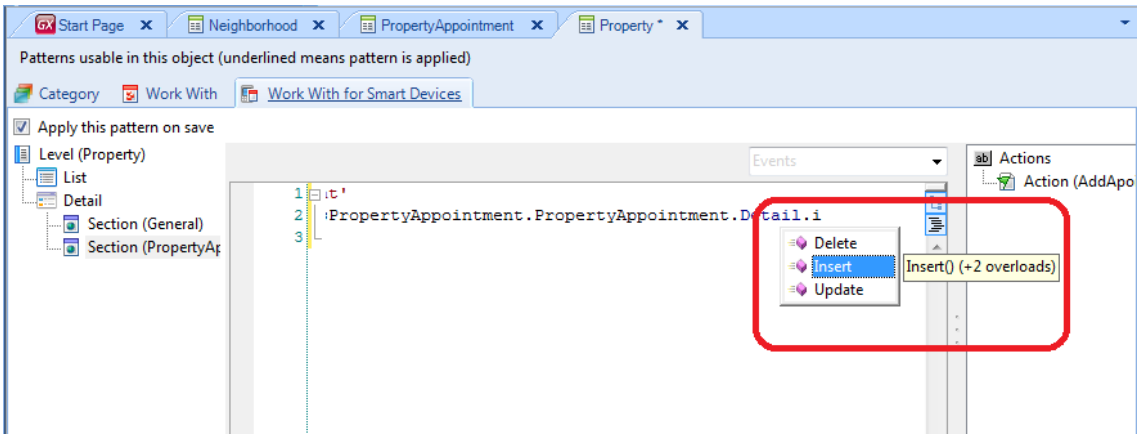
Si hacemos tap sobre una de las visitas {hacerlo}, vemos que nos muestra su información general: esto es {ir a GX}: el nodo Detail del Work with de PropertyAppointment, en modo View. En todo grid se configura la acción por defecto que ocurrirá al hacer tap sobre una línea {ir a GX a mostrar}. El Default siempre es abrir el view.

{volver a emulador} En el menú {abrirlo} sólo vemos la posibilidad de refrescar la pantalla, pero no hay otras acciones definidas. Queremos dar la posibilidad de insertar una nueva visita para esta propiedad {señalar el nombre arriba del caption} desde esta pantalla. Es decir, debemos llamar a este mismo detail {volver a hacer tap} pero al layout de Edit para hacer un Insert.

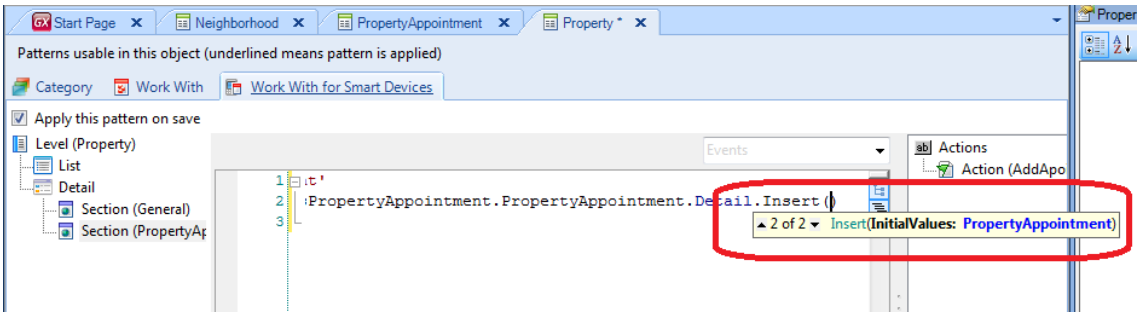
Para ello, {volver a Gx} con botón derecho sobre la Action Bar {hacerlo}, elegimos insertar una acción {hacerlo}, a la que llamaremos AddAppointment {hacerlo}.

Editamos las propiedades del botón, y acortamos el Caption (lo que se verá en ejecución). Ahora programemos el código del evento asociado {botón derecho Go to Event}.

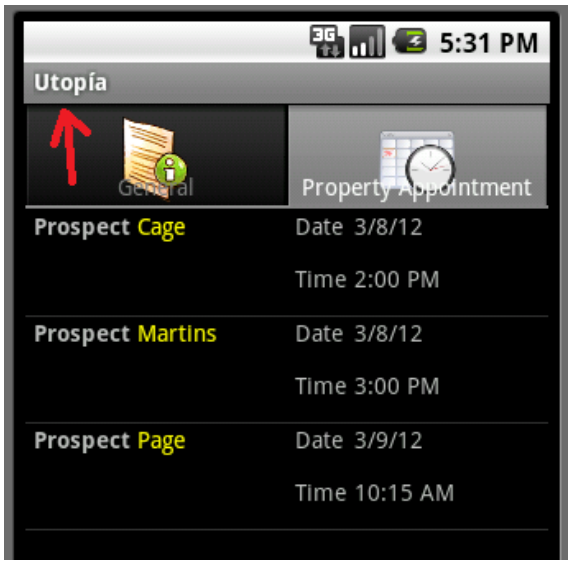
Arrastramos el nombre del Work with {hacerlo}, presionamos punto {hacerlo} lo que sigue es el nivel. Lo pone automático porque sólo hay uno {mostrar en el ide, cambiando de tab al de PropertyAppointment}, y luego a quién queremos llamar de ese work with? Al Detail {elegir propiedad} en modo Insert {elegir}



Nos indica que tenemos 2 posibilidades, o no enviar parámetros, o enviar como parámetro una variable de tipo business component de la transacción PropertyAppointment.

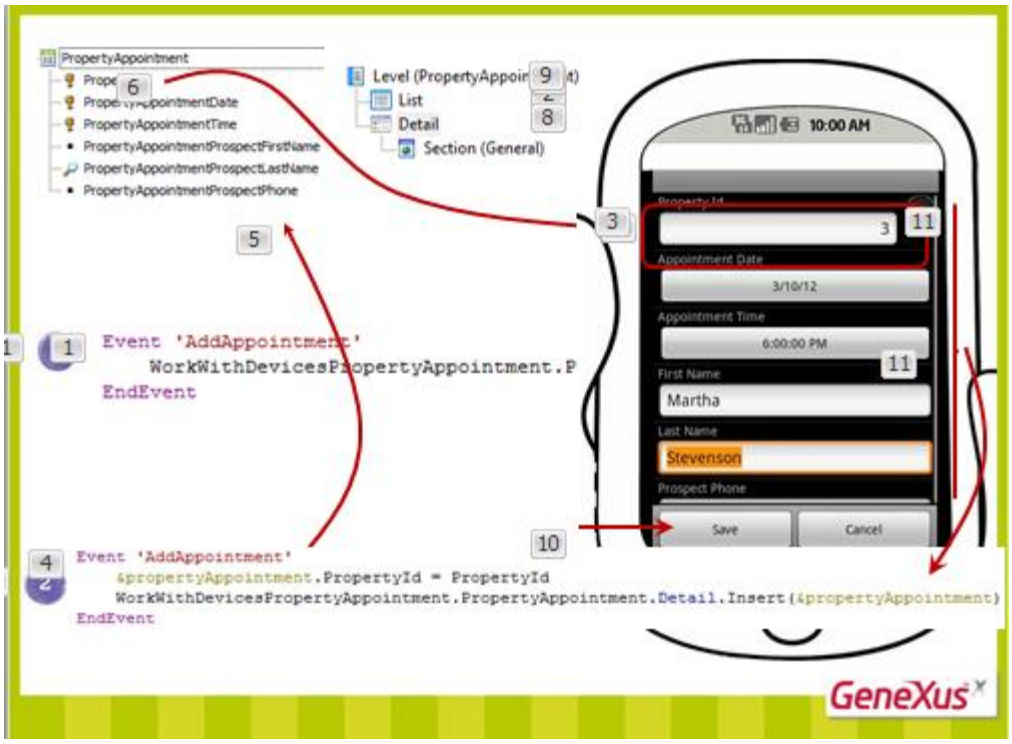


En nuestro caso, necesitaremos enviar un parámetro, el identificador de propiedad, porque querremos insertar una nueva visita,



pero para esta propiedad {señalarla}

{ppts}



Entonces, esta primera alternativa, no nos sirve, pues queremos que la propiedad quede instanciada.

Tendremos que usar la segunda alternativa. En ella, definimos una variable de tipo BC de PropertyAppointment, y en esa variable solamente inicializamos el campo que nos interesa enviar con valor, que es PropertyId, tras lo cual…

Lo interesante será que al hacer esto, una vez completados por el usuario los campos de esta pantalla, al hacerse el Save, se devolverá la misma variable cargada con todos los valores, para poder utilizarlos luego, como veremos…

{demo}

Una vez hecho esto en GeneXus… (ya definida la variable {señalarla} de tipo el BC de PropertyAppointment. Y ya asignada su propiedad PropertyId con el valor del atributo PropertyId recibido por parámetro {mostrar} y enviada por parámetro, veamos cómo nos quedó en ejecución…

En la lista de propiedades, elegimos una, y vemos sus visitas agendadas… Ahora en el Menú, nos aparece el botón Add (podríamos haberle asociado una imagen) y al presionarlo… vemos que inicializó con el id de esa propiedad.

Ingresemos valores {hacerlo} y al hacer Save, vuelve y refresca la pantalla, donde podemos ver la visita recién ingreada.

Si hacemos tap sobre la misma, nos muestra todos sus datos.

Aparece esta imagen {la del telefonito}, y al hacer tap sobre la misma, nos lleva a la libreta de contactos del dispositivo, para que hagamos la llamada telefónica a ese número. ¿Cómo lo supo? {volver a GX}

Porque el dominio del atributo es Phone. Vemos una vez más, que con los dominios semánticos la aplicación se integra con funcionalidades del dispositivo de manera transparente para nosotros.

Pero además de esta forma transparente de integración, tenemos otras explícitas, que son a través de las distintas APIS que nos provee GeneXus.

Por ejemplo, si quisiéramos que una vez ingresada la visita a la base de datos {hacer back y mostrar el view de la visita}, los datos del prospect {señalarlos} queden registrados en la libreta de contactos del dispositivo, {ir al Evento} tendremos que agregar aquí una llamada a la libreta de contactos pasándole los tres datos que tenemos: nombre, apellido y teléfono del contacto. Pero ¿cómo interactuamos con esa funcionalidad del dispositivo, que va a depender de cada plataforma?

A través de una api provista por GeneXus, AddressBook… {ir al folder SmartDevicesApi}

Al abrir este objeto externo, vemos que nos provee de métodos para agregar un contacto, elminarlo o visualizarlo.

Lo que haremos entonces, es…

llamar a la libreta de direcciones del dispositivo {arrastrar al Evento}, a través de esta api, utlizando el método AddContact… al que debemos pasarle todos los parámetros que requiere (todos y en el mismo orden). Primero va el nombre. ¿De dónde lo sacamos? Vino cargado en el BC… así que {&propertyAppointment.ProspectName}, después nos pide el apellido {elegirlo}, luego el email (como no lo tenemos dejamos pasamos un valor vacío) {'' hacerlo}, luego el teléfono {elegirlo}, después vendría el nombre de la compañía, no le tenemos así que pasamos vacío, después la foto y por último un mensaje.

Vemos que al grabar nos está pidiendo el comando Composite, ya que estamos realizando más de una invocación. {agregarlo} Lo estudiaremos en breve.

Ejecutemos…

Tras grabar la visita, nos está llevando a la libreta de direcciones, para permitirnos agregar el contacto… con los datos que le hemos enviado.

{volver a Gx, AL FOLDER VIEW A MOSTRAR LAS APIS}

Tenemos todas estas APIS, que se agregan a la kb en el momento de utilizar el generador para Smart Devices. Algunas requieren el uso de datos estructurados, por lo que también se crean automáticamente los SDTs correspondientes para su manipulación.

Por ejemplo, vemos esta api GeoLocation, {abrirla} que necesita para algunos de los métodos ofrecidos, de estos dos SDTs {señalarlos}. Esta api permite interactuar con el gps del dispositivo y obtener por ejemplo su localización en un momento dado, a través del método GetMyLocation. Entre otras cosas, también permitirá registrar el camino efectuado con el dispositivo en un intervalo de tiempo dado, con el método StarTracking {señalarlo} , detener el registro con el EndTracking {señalarlo}, disparar alertas de proximidad, etc.

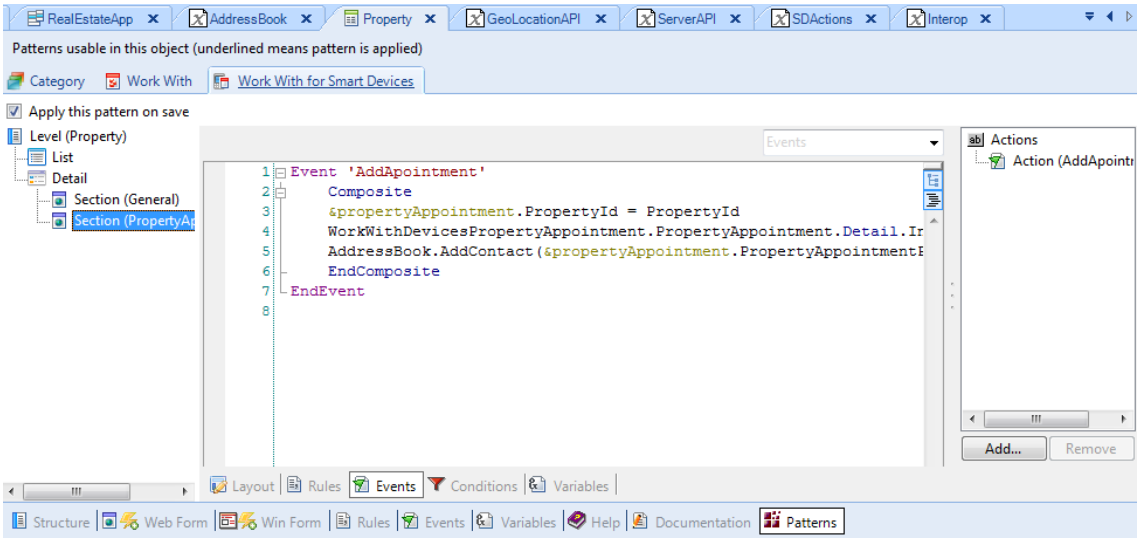
{abrir SDActions} Por su uso extendido, los métodos de algunas apis podrán utilizarse sin tener que referirlos al nombre de la api. Ya vimos un caso, el del método return {señalarlo} de SDActions. {abrir WW property → Section(Generla)→ evento Delete}

{volver a SDActions} La misma suerte corre el método Refresh {señalar}, que se utilizará para obligar a refrescar la pantalla. Y algunos de la api Interop {abrirla}.

{agregar callout con flecha sobre Msg } para desplegar un mensaje en la pantalla, {agregar callout con flecha sobre Confirm } para pedir confirmación al usuario para continuar, {agregar callout con flecha sobre OpenInBrowser} para por ejemplo armar un link dinámico.

Esta api implementa la posibilidad de enviar mensajes, escanear códigos de barras, reproducir un video, o audio, enviar un e mail, un mensaje, etc.

{Volver en GX al evento AddAppointment}



Hasta aquí hemos programado código al evento de usuario que agregamos. ¿Qué otras cosas podemos hacer dentro de un evento ejecutado en el cliente?

FIN VIDEO 5