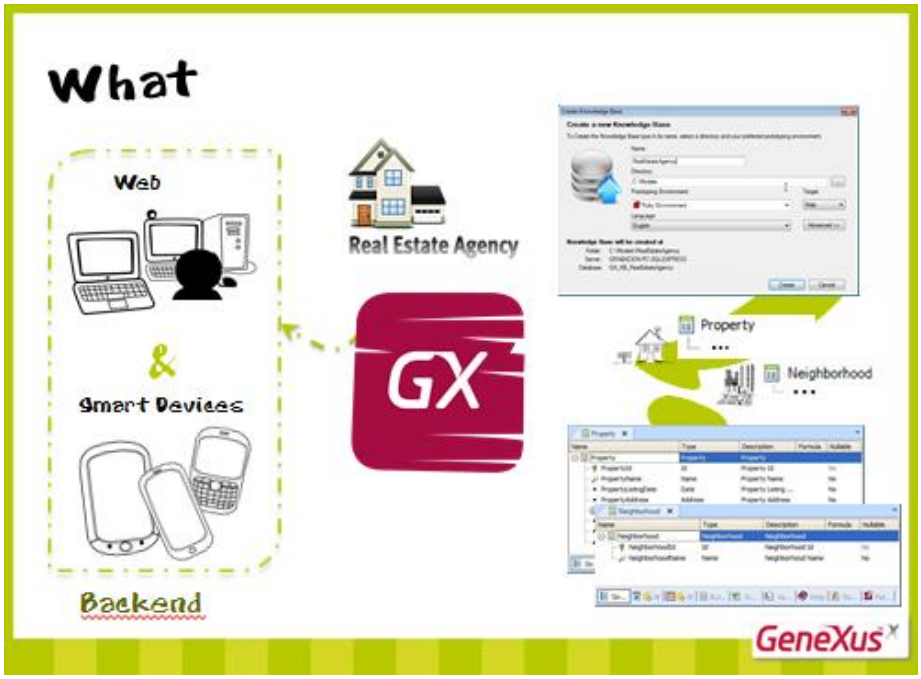
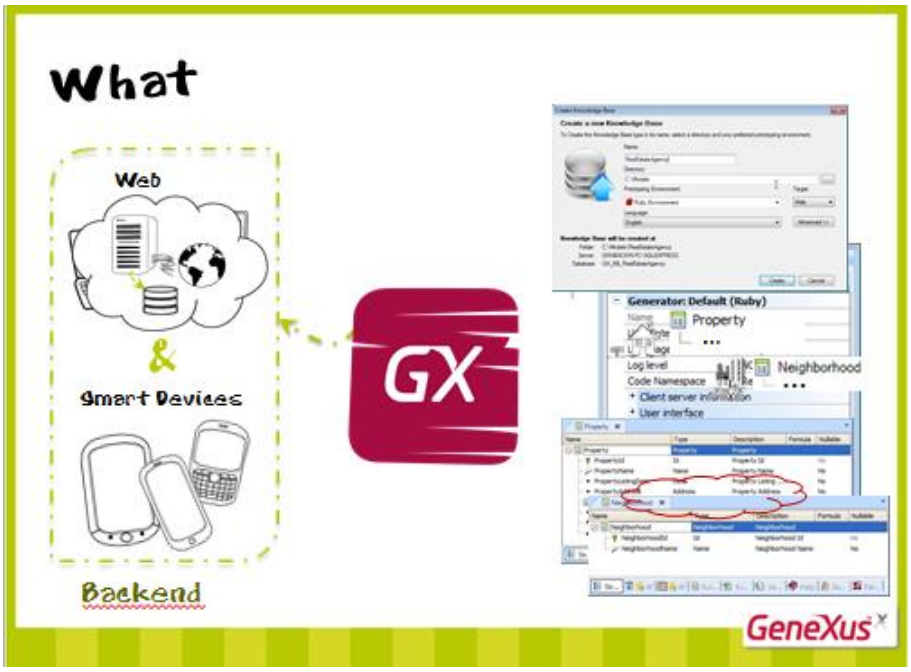
VIDEO 1: Work with for Smart Devices – Generalidades e Layout do List



Vamos construir um backend para uma imobiliária com uma parte web e outra parte para Smart Devices que será utilizada pelos corretores de imóveis desde seu celular.

Para isso criamos uma KB, a transação Property que registra os imóveis à venda ou para alugar; e a transação Neighborhood, para registrar os bairros.



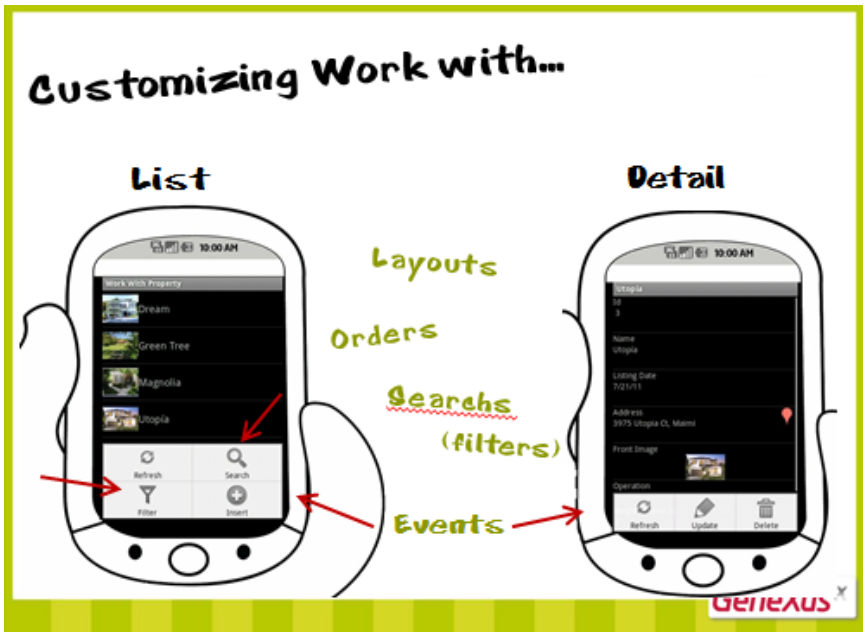
Além disso, definimos que o backend web será gerado em **ruby**, na nuvem.

A fim de fazer a implementação da parte para Smart Devices, aplicamos o pattern Work With for Smart Devices à transação Property…

à transação Neighborhood…

e criamos um objeto Dashboard como ponto de entrada.

A princípio decidimos gerar somente em **Android**, a plataforma padrão.



A seguir, veremos como personalizar o “Trabalhar com” uma entidade (representada por uma transação).

Em outra parte, veremos que também se pode criar “Trabalhar com” não associados a transações.

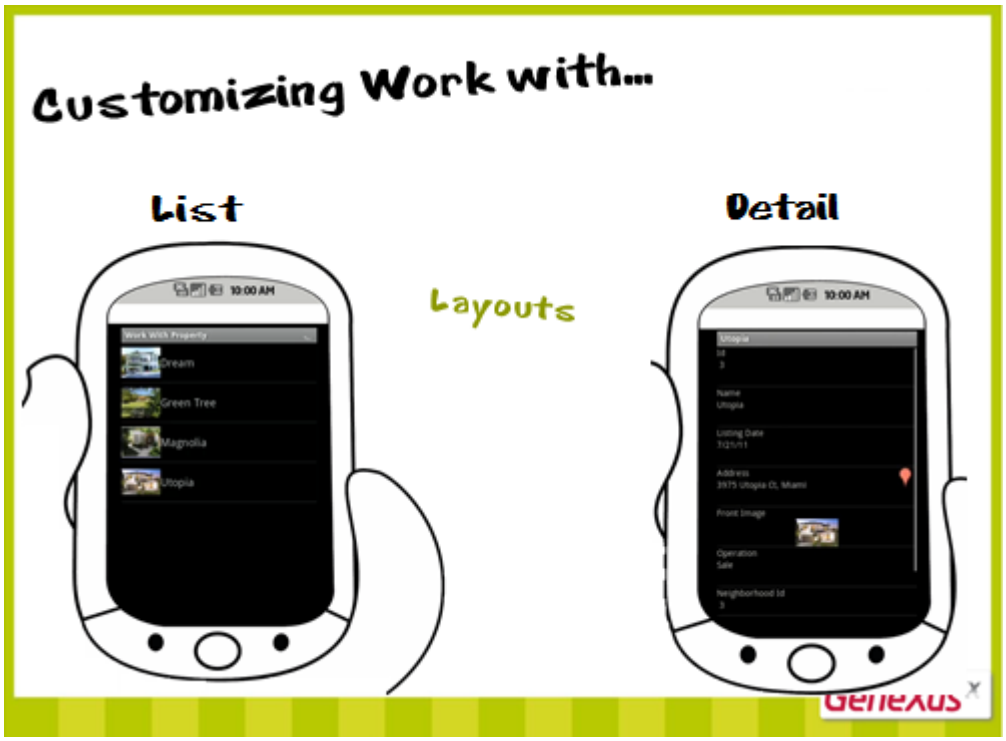
Como sabemos, a especificação da qual se deseja obter um “Trabalhar com” certa transação implementa, automaticamente, uma tela que mostra a lista dos elementos e outra que mostra a informação detalhada de um elemento particular.

Portanto, podemos personalizar a informação mostrada ao usuário e seu modo de exibição. Em resumo: tudo o que tem a ver com a interface de usuário, isto é, com os Layouts.

Também podemos personalizar a ordem como a informação sai do List…

Ou os tipos de buscas oferecidas ao usuário para selecionar a informação a ser recuperada, assim como outros filtros internos que selecionam as informações a serem implantadas…

E, por último, podemos personalizar o comportamento, incluindo ou modificando eventos com a finalidade de realizar determinadas ações.



Começaremos pelas telas.

Dependendo de cada tela, podemos fazer as seguintes personalizações…

{demo}

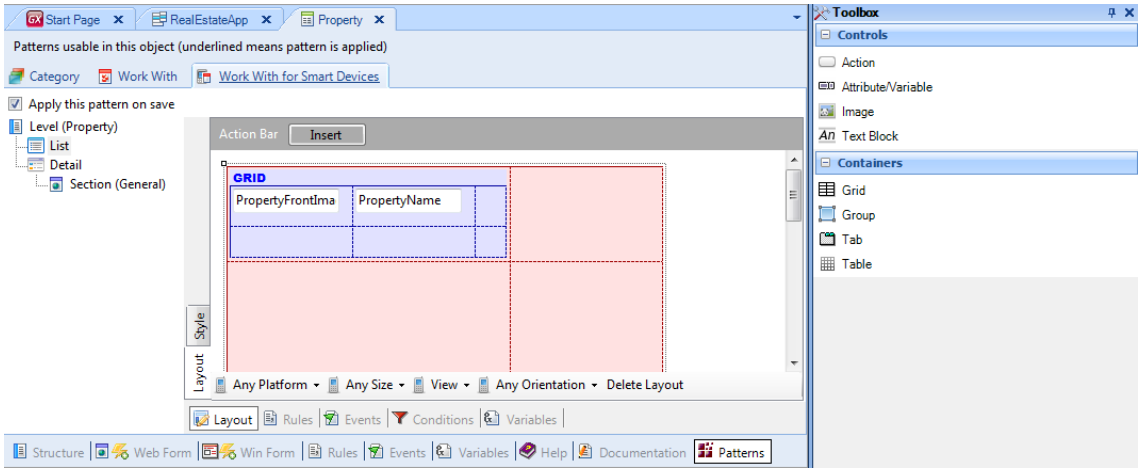
{demo. Ponerle al Theme Android, a la clase Attribute margin left = 5}

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

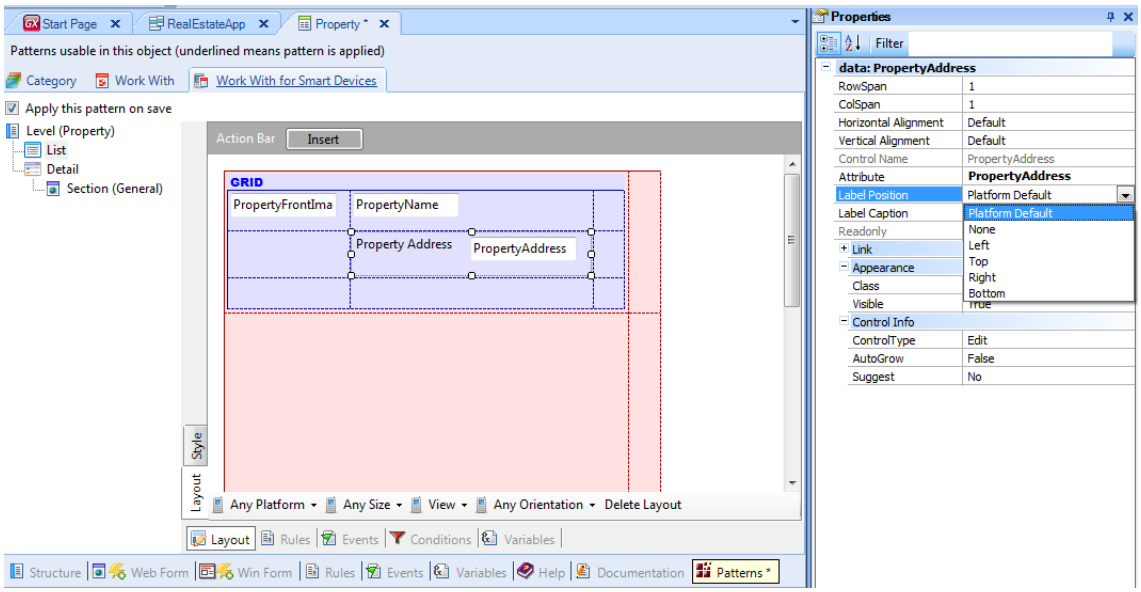
Actual:
http://apps2.genexus.com/Ide18ce2b1f85c4941a47debd1e95241cc/DeveloperMenu.xml

Posicionando na pasta List do pattern Work With correspondente à transação Property, observamos que nos mostra o Layout.

Na forma padrão, temos um controle em grid (no qual se implementa a lista) e dentro dele aparecem somente alguns atributos da transação. Podemos incluir ou retirar atributos desse grid, assim como outros controles, através da toolbox.



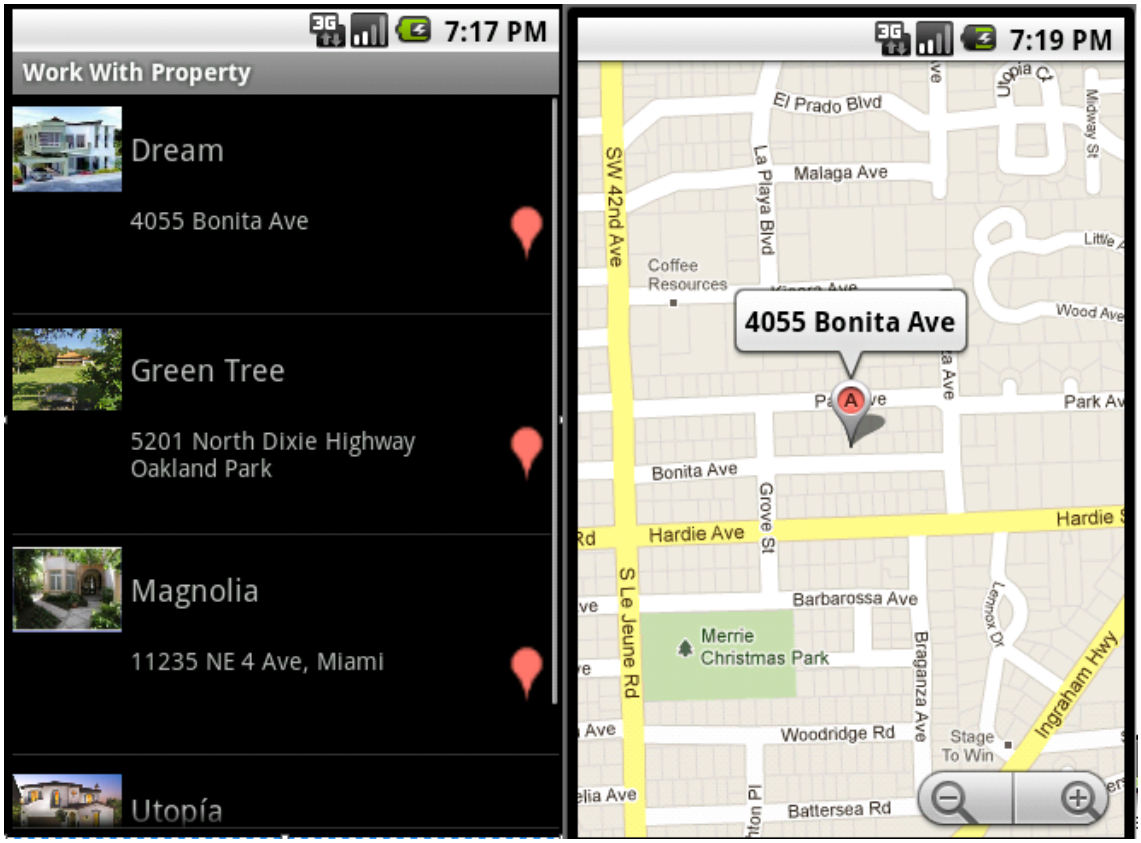
Por exemplo, podemos incluir o endereço: PropertyAddress…



Também podemos personalizar as propriedades deste controle: por exemplo, a posição da etiqueta… aqui podemos vê-la à esquerda, mas suponhamos que a

queremos na parte superior (como costumam aparecer no Android)…(hacerlo) ou não queremos que apareça (hacerlo)… O valor “Platform Default” é como um curinga: estabelece que, ao ser executado, será mostrado de acordo com o padrão da plataforma escolhida. No caso de plataforma Android, a etiqueta será mostrada na parte superior, ainda que nesta tela de desenho esteja à esquerda.

Vamos vê-lo em execução.



Vemos que, junto ao endereço, aparece o ícone que permite visualizá-la num mapa (graças ao domínio Address no qual se baseia o atributo, {dejar tiempo muerto para poder poner un callout imagem con la estructura de la trn} a semântica correspondente é incorporada, o que significa para nós que o programa gerado para o Smart Device incorpora a lógica para utilizar seu mapa **nativo**). Vemos, assim, que o aplicativo gerado se **integra** às funcionalidades nativas do dispositivo. {callout que diga “Device integration… with native functionalities”}

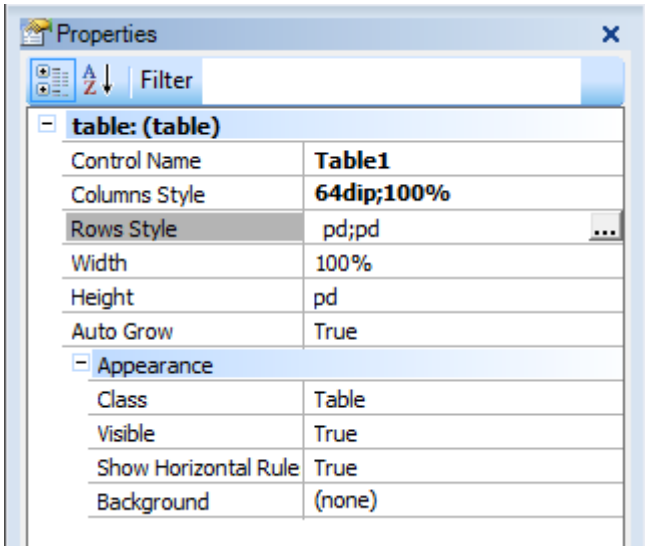
Agora, queremos modificar a distribuição dos controles e os espaços entre as informações.

Observaremos que a informação de cada propriedade é apresentada em duas linhas {poner callout para señalarlas, por lo que hacer un **silencio** en el audio para que entren}

e em duas colunas {poner callout para señalarlas, y hacer un silencio}. Isto significa que, numa tabela (interna ao grid), {Mostrar la tabela en GX } a imagem da propriedade e o seu nome aparecerão na primeira coluna, e o endereço aparecerá na segunda coluna.

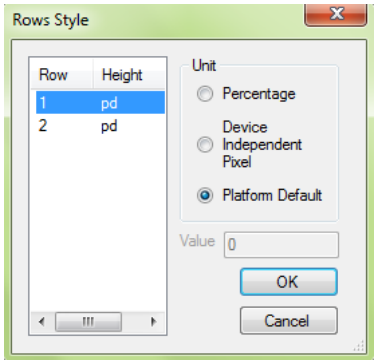
Se desejarmos que a coluna com a imagem seja um pouco mais larga... podemos modificar o estilo das colunas nas propriedades da tabela. De acordo com o padrão, a primeira ocupa 64dips (Device independent pixels) e a segunda se estende até alcançar 100% de largura.

{callout que diga “Dip?”} Dip? É uma unidade de medida que representa uma abstração sobre o pixel, de modo que se pode fazer a escala das telas em diferentes tamanhos. Logo, o aplicativo gerado faz a conversão para pixels físicos, de acordo com a plataforma do dispositivo.



Aqui fixamos um tamanho fixo de 64 dips para a primeira coluna. Se quisermos, pode-se determinar o contrário, ou seja, que a primeira coluna ocupe 30% de largura e a segunda fique com o 70% restante. {hacerlo en GX y mostrar resultado en el emulador}

Ao observarmos as linhas, percebemos que ambas têm o mesmo tamanho. Este tamanho pode ser modificado. {ir a GX y posicionarse en la tabla, fila Rows Style}



Vemos que as duas linhas têm o valor “pd”. O que significa isso? “Platform Default”. Quer dizer que o tamanho (na altura) de cada linha será tomado do valor padrão da plataforma. O valor padrão de Android/Phones é de 64 dips.

Podemos modificar os valores de cada linha, tanto fixando en dips (por exemplo, a primeira linha ocupará 65 dips e a segunda 35), como através de uma porcentagem relativa à parte superior da tabela. Se a primeira linha ocupa 65 por cento da parte de superior da tabela, e a segunda 35 por cento, o alto da tabela está configurado através da propriedade Height que, por padrão, assume o valor “Platform defaul” mas que pode ser variado a 100dips, por exemplo.

{mostrar el resultado en el Emulador}

Também poderíamos querer estender a imagem para ocupar as duas linhas.

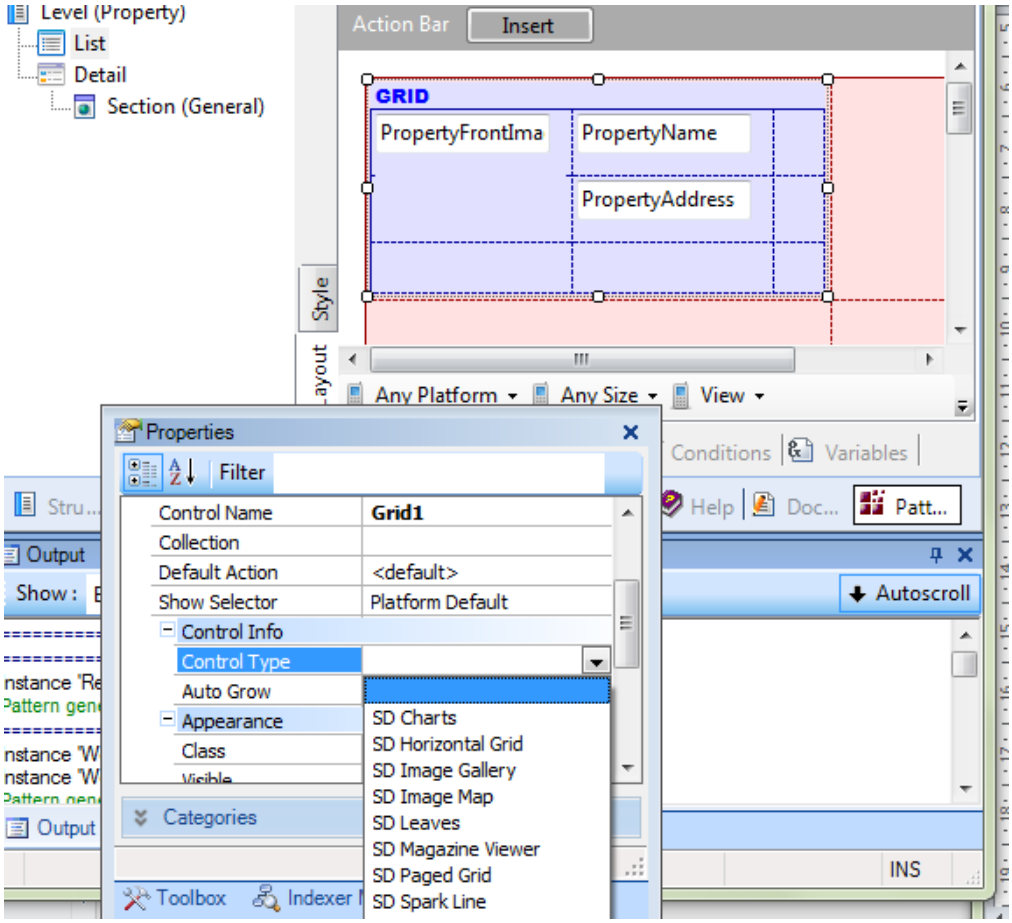
{hacerlo en GeneXus y mostrar el resultado en el emulador}

Agora suponhamos que, ao invés de visualizar a lista de imóveis como uma sucessão de elementos em linha, quisermos visualizá-la de outra maneira, por exemplo, como uma galeria de imagens de fachada.

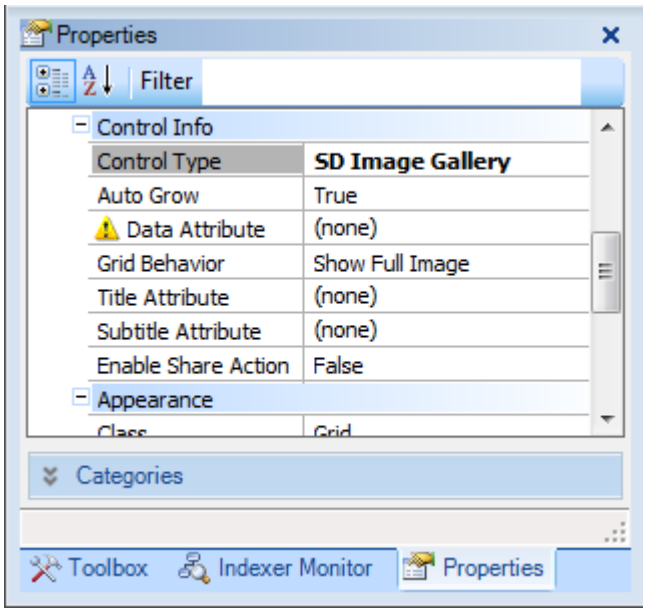
Quem implementa a lista em GeneXus?

{ir a GX y pararse sobre el grid}

O controle grid fará isso. Vamos buscar, entre suas propriedades, uma que represente a forma como se apresenta a informação. O tipo de controle cumpre essa função. Quando não tem valor atribuído, representa o tipo de listado padrão.



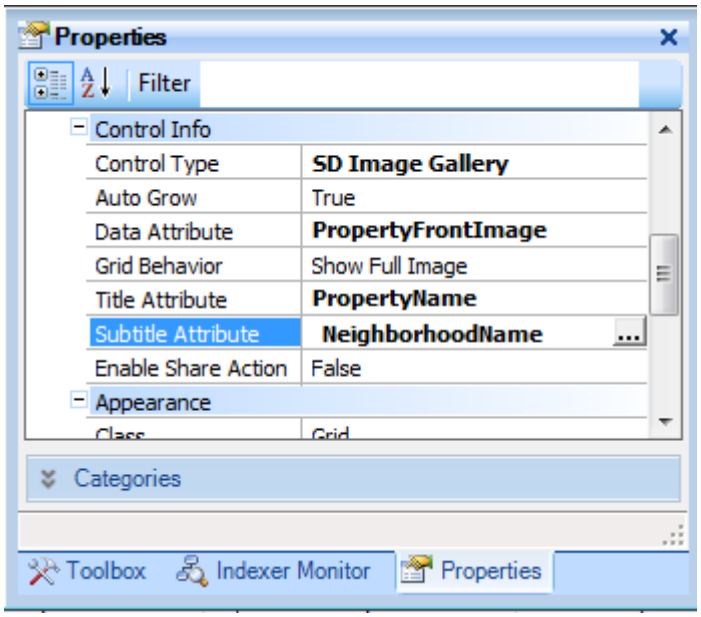
Mas podemos modificá-lo, escolhendo entre os User Controls específicos para Smart Devices que nos oferece nesta lista. Entre eles, escolhemos o user control Image Gallery.



Temos que configurar o atributo que contém a imagem utilizada para a galeria {poner Data Attribute = PropertyFrontImage}

E, se assim desejarmos, o atributo que utilizaremos como título da imagem {hacerlo}

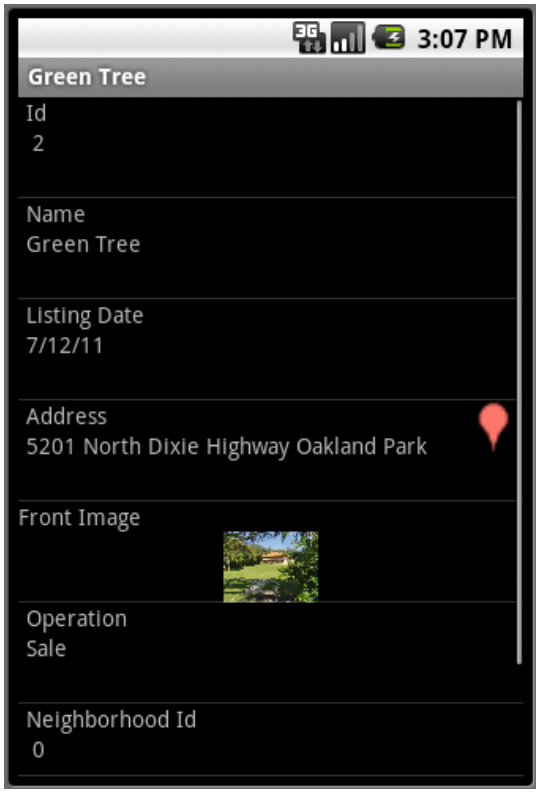
e o que usaremos como subtítulo {hacerlo}.



{mostrar el resultado en el emulador}

Podemos nos mover pela galeria tanto entre as imagens em miniatura, como através da imagem ampliada.

E fazer tap sobre a imagem da propriedade, como no detalhe:



De acordo com os dados da entidade, os user controls terão sentido nesse contexto. Por exemplo, para utilizar o user control SD Maps, que mostra a lista como pontos em um mapa, precisaremos contar com um atributo de tipo de dado para o imóvel: o domínio **Geolocation**, que armazena latitude e longitude geográfica dessa propriedade.

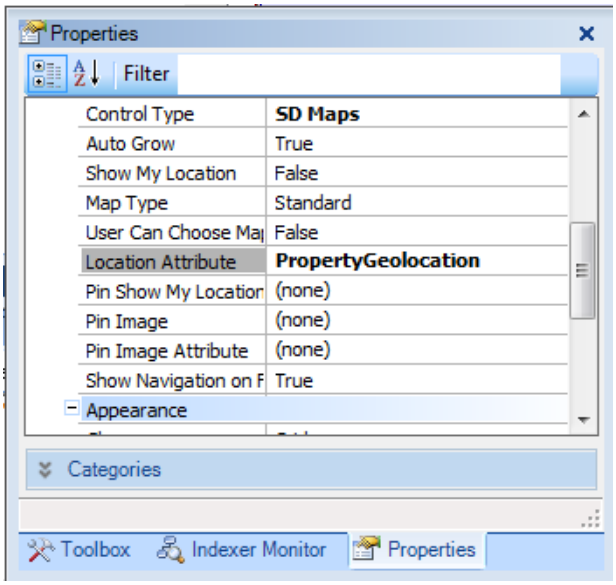
{en GX ir a la Structure de la trn y agregar el att PropertyGeolocation y grabar.

Vamos defini-lo na estrutura da transação.

Vemos que assume automaticamente o domínio Geolocation.

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

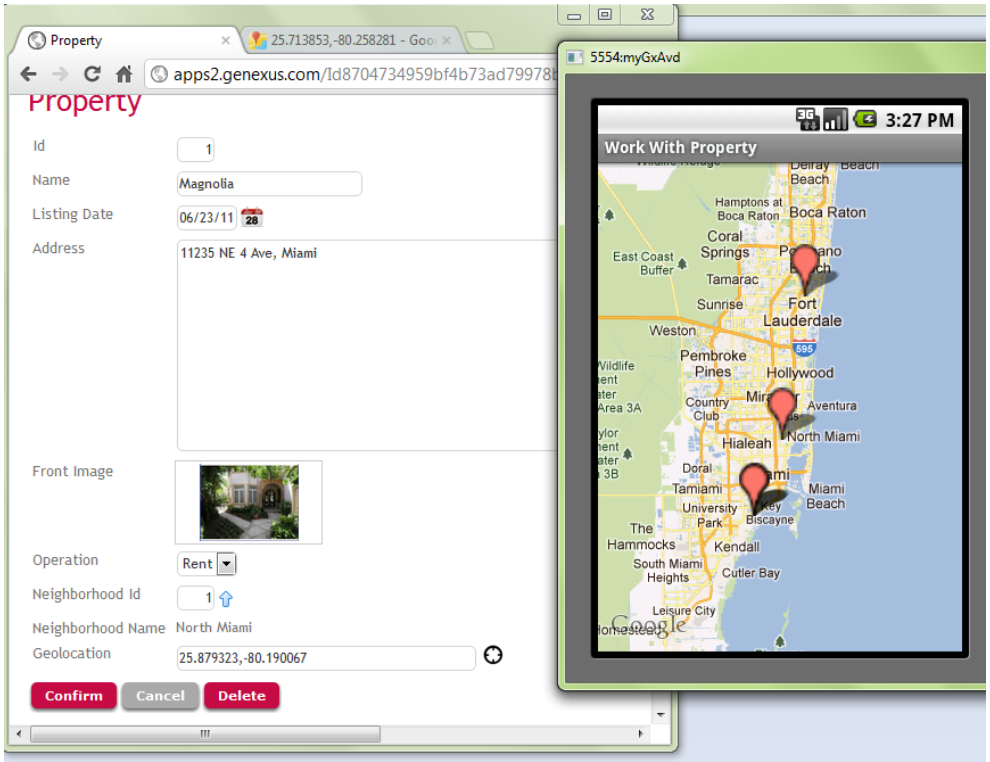
E agora sim, indicamos que o atributo que contém a localização geográfica é o que acabamos de criar, PropertyGeolocation.



O GeneXus fará uma reorganização a fim de incluir o novo atributo à tabela e, uma vez gerado novamente o aplicativo, teremos que indicar a localização geográfica dos imóveis armazenados …

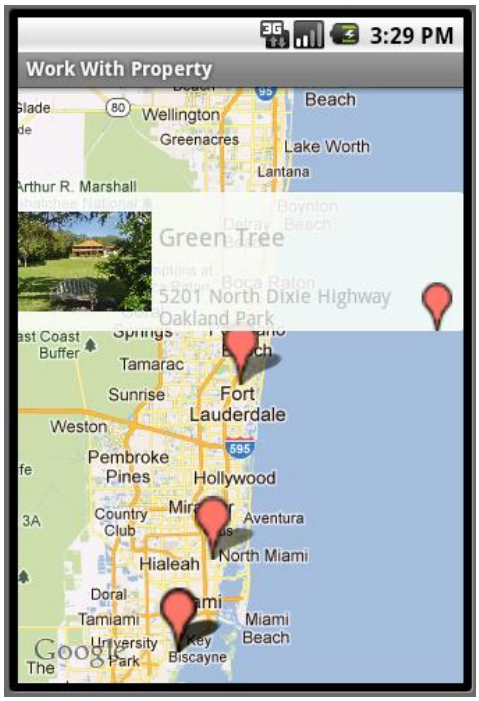
{transición para saltear todo esto }

Uma vez feito isso, vemos que, ao executar a lista…

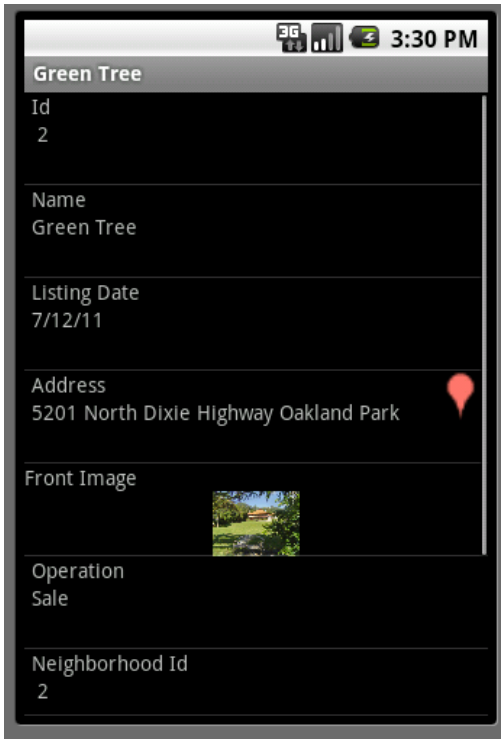


E ao fazer tap sobre um ponto exibido…

Ou outro…



E se quisermos ver o detalhe completo do imóvel, ao fazer tap:



A tela de List continua desempenhando suas funções, por mais que tenha sido modificada a forma de exibição das informações. Por exemplo, podemos fazer um Search por endereço… {hacerlo}

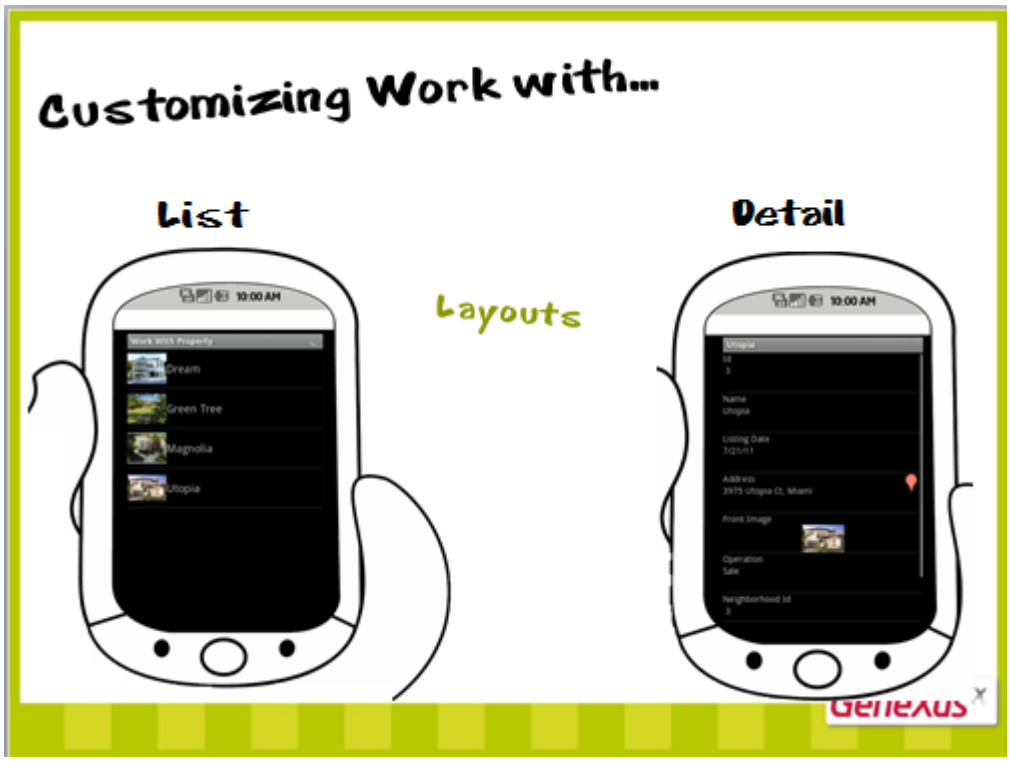
Filtrar por bairro… {hacerlo por Coral Gables}

Ou inserir uma nova propriedade imobiliária {invocar al Insert sin inserir nada}

{transición hacia la ppt siguiente}

Aqui temos outros exemplos de User Controls para mostrar, de maneira particular, as informações das listas.

O desenvolvedor também poderá criar seus próprios User Controls. Para isso, recomendamos buscar a documentação em nosso wiki.



Acabamos de personalizar o Layout da lista de imóveis…

Também queremos personalizar a informação detalhada de um imóvel em particular...

Para isso, contamos com o hub Detail, que tem a seção geral aninhada. Vamos ver um pouco de cada um…

FIN VIDEO 1

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

Acabamos de personalizar o Layout da lista de imóveis…

Também queremos personalizar a informação detalhada de uma propriedade em particular...

Para isso, contamos com o hub Detail, que possui uma seção aninhada, a geral. Vamos investigar um pouco cada coisa…

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

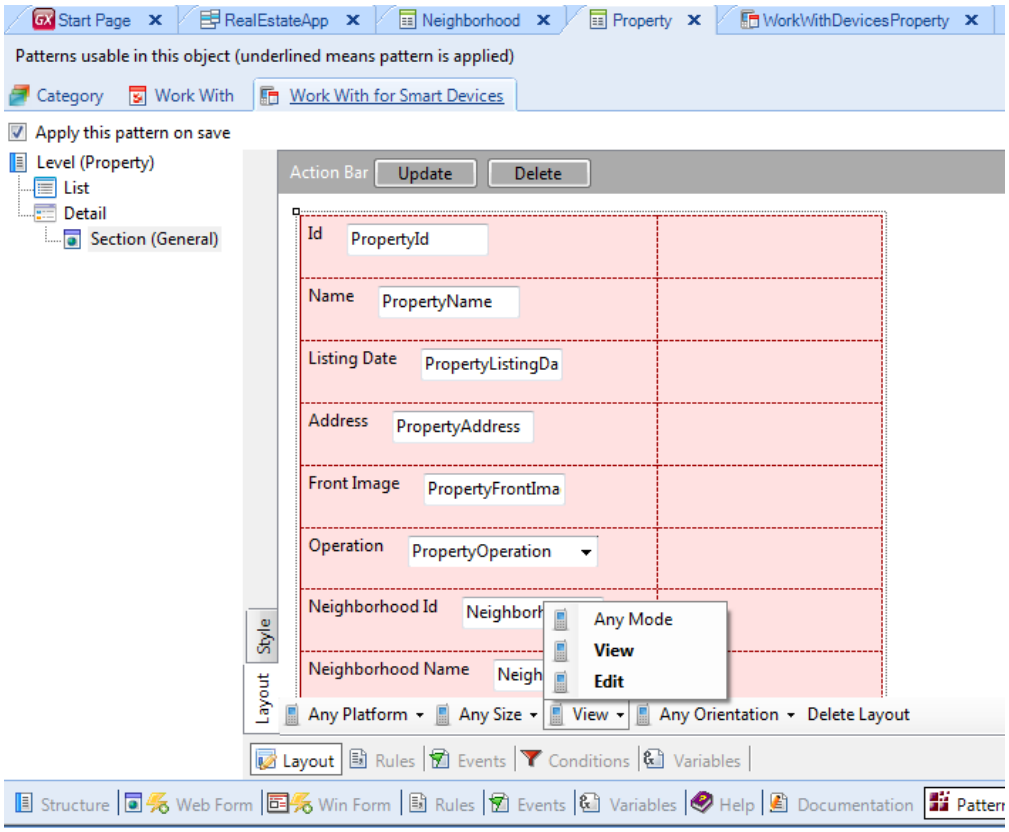
Vemos que o layout do node Detail tem um atributo, o que está marcado na transação como Description attribute, e um contêiner de seções, onde as mesmas serão carregadas. Nesta transação {Property} temos somente uma seção, mas, por exemplo, nesta outra {Neighborhood}, temos duas e Ali terá mais sentido este contêiner. Vamos tratá-lo mais adiante.

Até agora observamos que ao ser executado, {mostrar el List Property} quando um imóvel da lista é selecionado, aparece antes de tudo o nome da propriedade e, em seguida, toda sua informação. Ao selecionar um bairro, aparece antes de tudo seu nome, e depois dois tabs com sua informação: uno para cada seção.

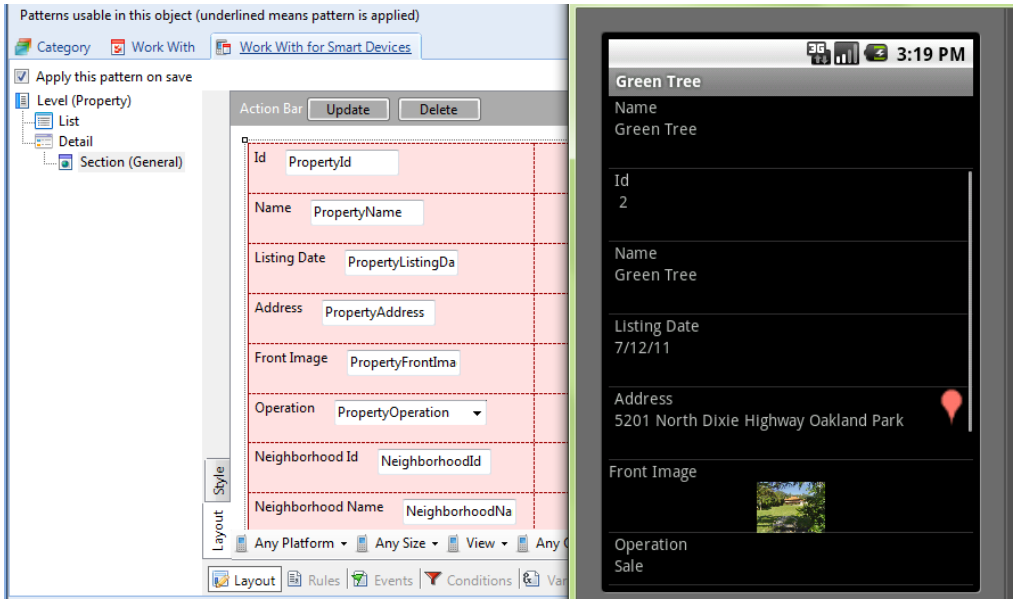
Se eliminarmos este atributo da parte fixa…{hacerlo sólo de Property} F5…{transición} vemos que não aparece...e agora somente são mostradas as informações da seção geral.

Agora vamos para o Layout da seção geral. Aqui se encontram, por padrão, todos os atributos presentes na estrutura da transação.

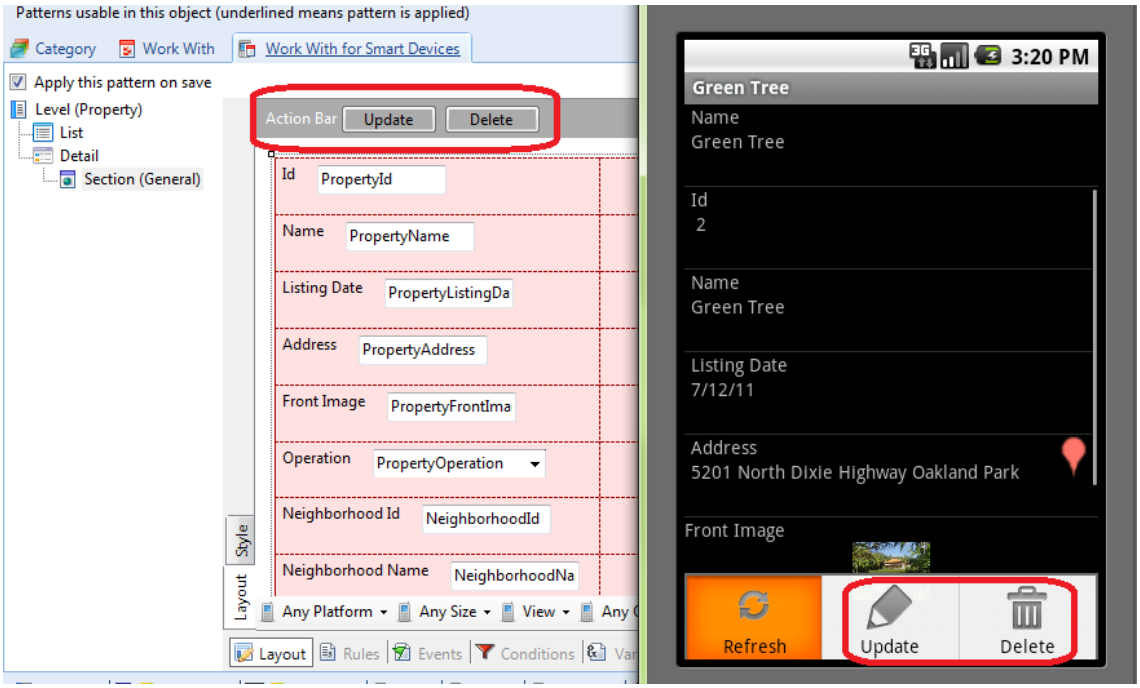
Para sermos mais exatos, estamos no Layout da seção geral, no **modo View**.



Este é o layout que será utilizado toda vez que quisermos visualizar a informação geral de um imóvel. Por exemplo, quando acionamos o tap sobre um imóvel da lista de propriedades do bairro... ou quando o acionamos sobre a lista geral de imóveis. {hacerlo en el emulador}



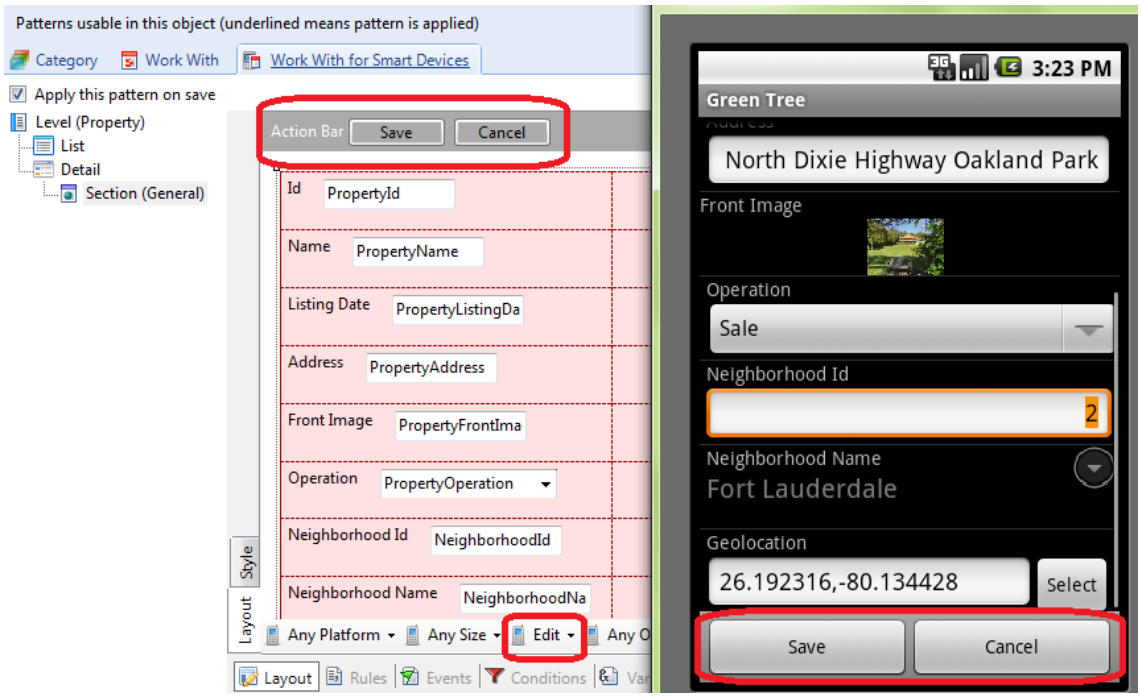
A partir deste layout podemos modificar ou apagar o imóvel. {menú: mostrar Update y Delete} Ações que, não por acaso, podem ser vistas aqui:



Quando pensamos na arquitetura, quando abrimos esta tela, a partir do dispositivo será feito um request para executar a função Rest (um data provider) que realizará a consulta da base de dados e devolverá um Json com a informação. No dispositivo aparecerá a tela a partir da informação devolvida.

Se quisermos atualizar este imóvel, como podemos mudar los valores com o layout que está sendo mostrado? {mostrar date picker, el combo de operation, y los botones etc.?

Este é o layout da Section(Geral), mas em seu **modo Edit**. {mudar de modo} Vamos ver que, neste caso, as ações oferecidas servem para gravar ou cancelar.



Portanto, estamos trabalhando com dois layouts distintos (ainda que tenham sido inicializados com os mesmos atributos).

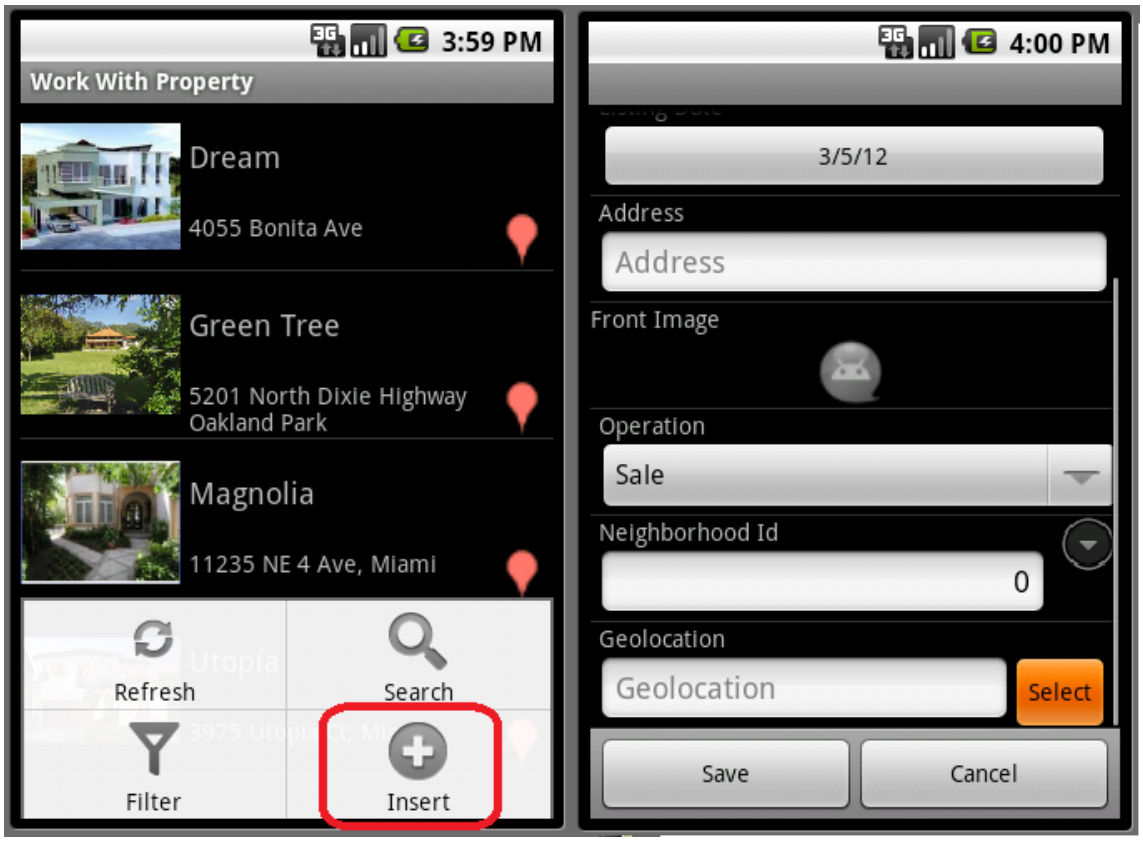
Poderíamos querer que, no modo Edit não apareça o atributo inferido NeighborhoodName { quitarlo}... e que no modo View, ao contrário, não apareçam os identificadores, pois são internos ao sistema {pasar a View y quitar los ids}. E também não nos interessa mostrar a geolocation. {quitarla y eliminar las filas sobrantes}

Através da toolbox podemos inserir outros controles ao layout.

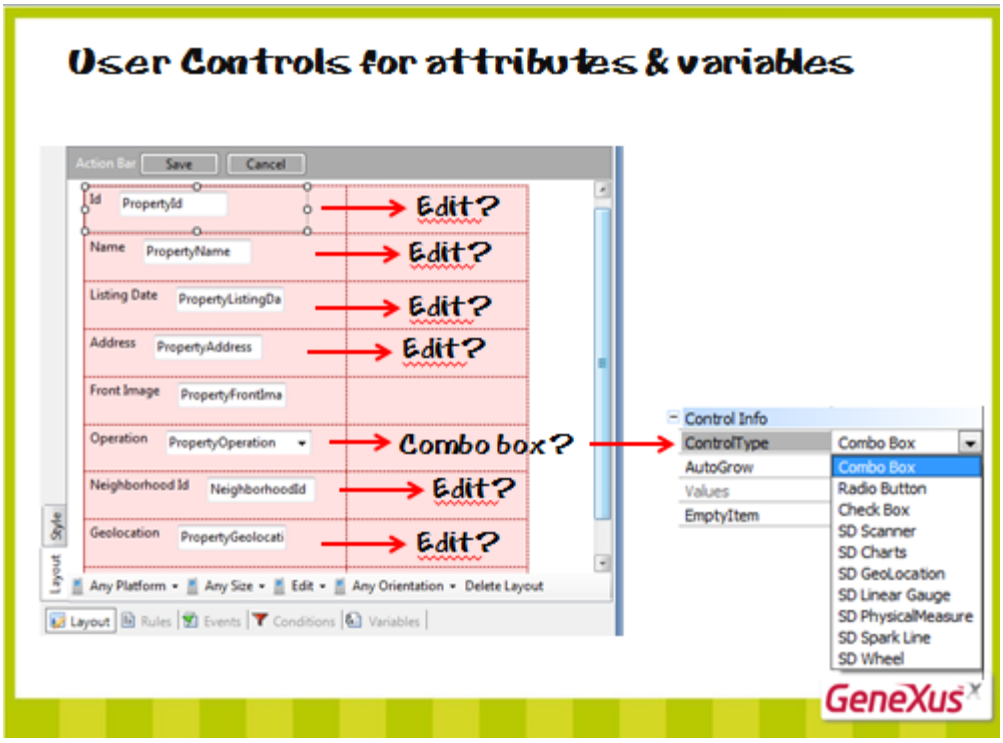
Vejamos como ficou informação detalhada de uma propriedade em execução. F5...

Quando é visualizada, não estão os ids e nem a geolocation... e quando é editada, por sua vez, eles aparecem, mas não aparece o nome do bairro.

A tela de detalhe em modo edit também é utilizada quando se insere um novo imóvel:



Podemos pensar neste Layout como um Form da transação. A diferença será que este layout é implementado como uma tela a mais do Work With e a manipulação da base de dados será feita de forma transparente para o desenvolvedor, através del Business Component associado (que será um serviço Rest executado no Server).



Podemos configurar os aspectos da tabela, como fizemos com o grid…

…e também mostrar os atributos utilizando algum outro tipo de controle para exibir as informações. Ou seja, dependendo do tipo de controle do atributo ou da variável no layout e de seus tipos de dados, serão oferecidos User Controls tanto para inserir a informação (no layout Edit) como para mostrá-la (no layout View).

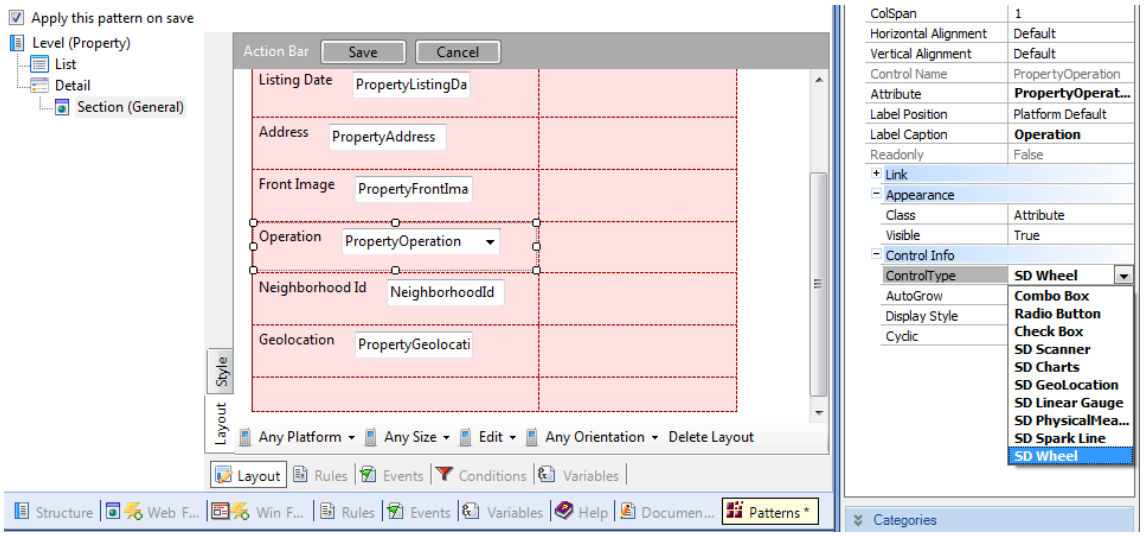
Por exemplo, podemos dar ao usuário a possiblidade de escolher a operação (venda ou aluguel) utilizando um campo, no lugar de um combo box, através do user control Wheel…

{demo}

{hacerlo} {pasar a Edit}

Será feito para inserir a operação, portanto para o Layout no modo Edit.

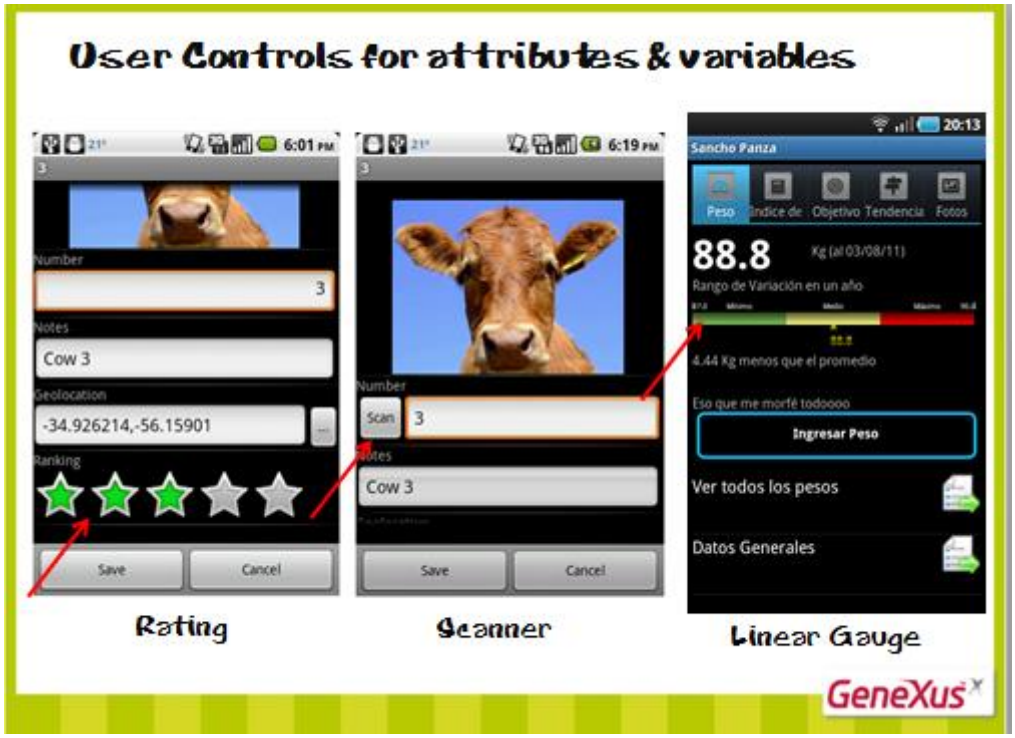
Aparecem alguns imóveis para serem configurados…



Vamos vê-lo em execução, F5… {transición}

Este caso tem más sentido quando se trata de vários valores e não apenas dois. Também é aplicado a atributos numéricos, e mostra um campo com todos os dígitos. É muito útil nos dispositivos touch screen, de modo a evitar a digitação via teclado.

{transición a ppts}



Entre outras coisas, podemos utilizar um atributo numérico para que o usuário faça o ranqueamento de uma informação... Para isso foi criado o user control Rating.

Podemos utilizá-lo para inserir um atributo escaneando um código de barras ou um QR Code. Ao configurar como tipo de controle **Scanner**, será apresentado um botão Scan ao lado do atributo e, ao pressioná-lo, será aberto o leitor instalado no dispositivo.

Também pode ser necessário mostrar a informação de um atributo dentro de uma determinada faixa. Para isso, será utilizado o controle **Linear Gauge**.

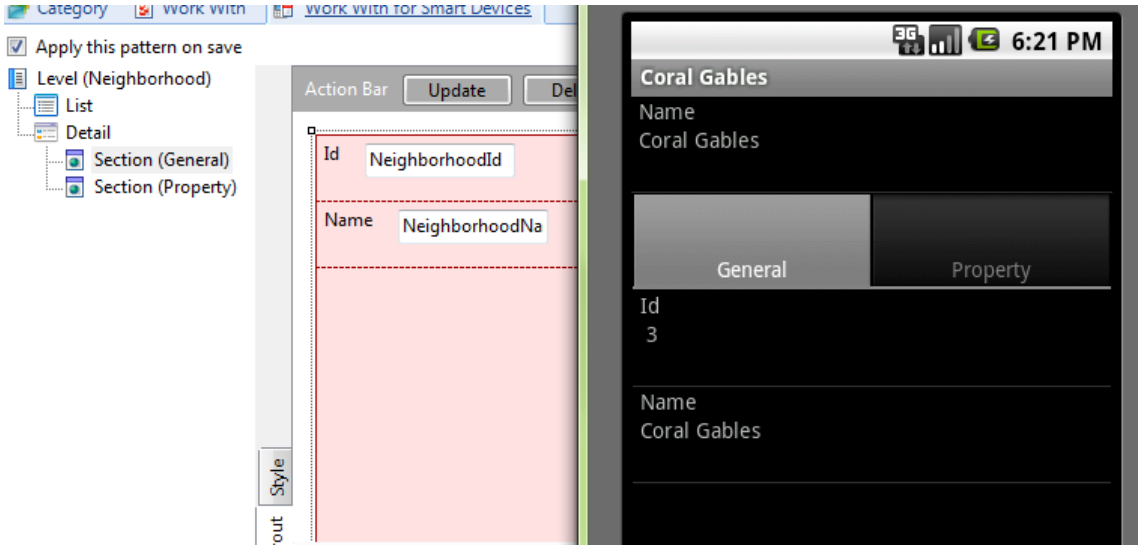
Estas e outras opções estão documentadas no nosso wiki.

E quando, além da seção geral, são apresentadas seções que correspondem à informação relacionada, como exibir toda essa informação?

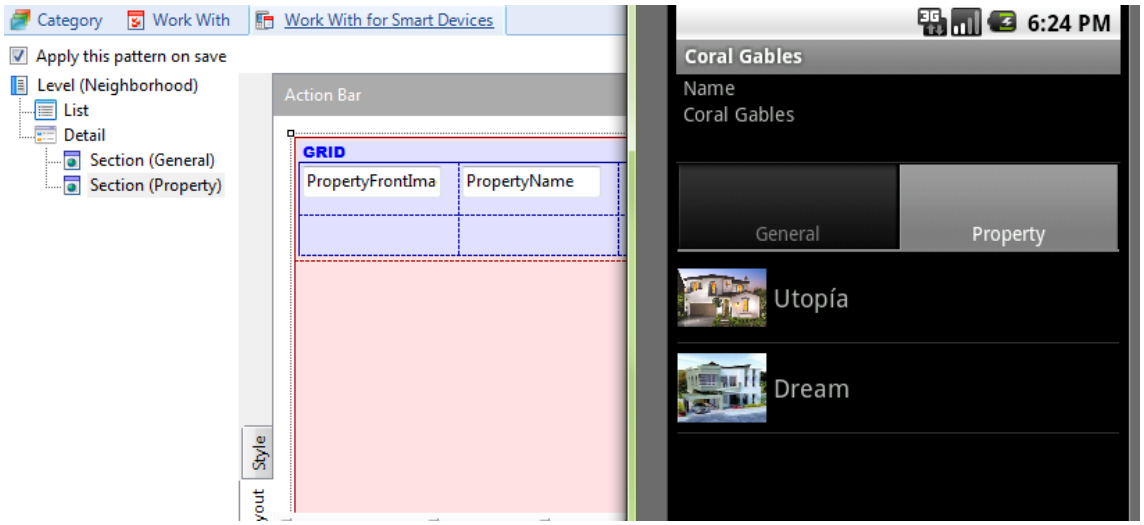
Como vimos, no caso do bairro, para o padrão do Android, são mostradas duas guias (tabs), cada uma com a informação de cada seção. Esta forma de apresentação dependerá de cada plataforma.

O layout de cada seção, assim como seu comportamento, são personalizados separadamente… Assim, por exemplo… {demo}

Aqui …… personalizamos o layout desta tela {mostrar layout de section(Geral) viendo a la vez el emulador)



E {moverse en el emulador al tab Property} aqui, personalizamos a informação desta outra tela, que mostra os imóveis deste {señarlar el nombre arriba en el emulador} bairro.



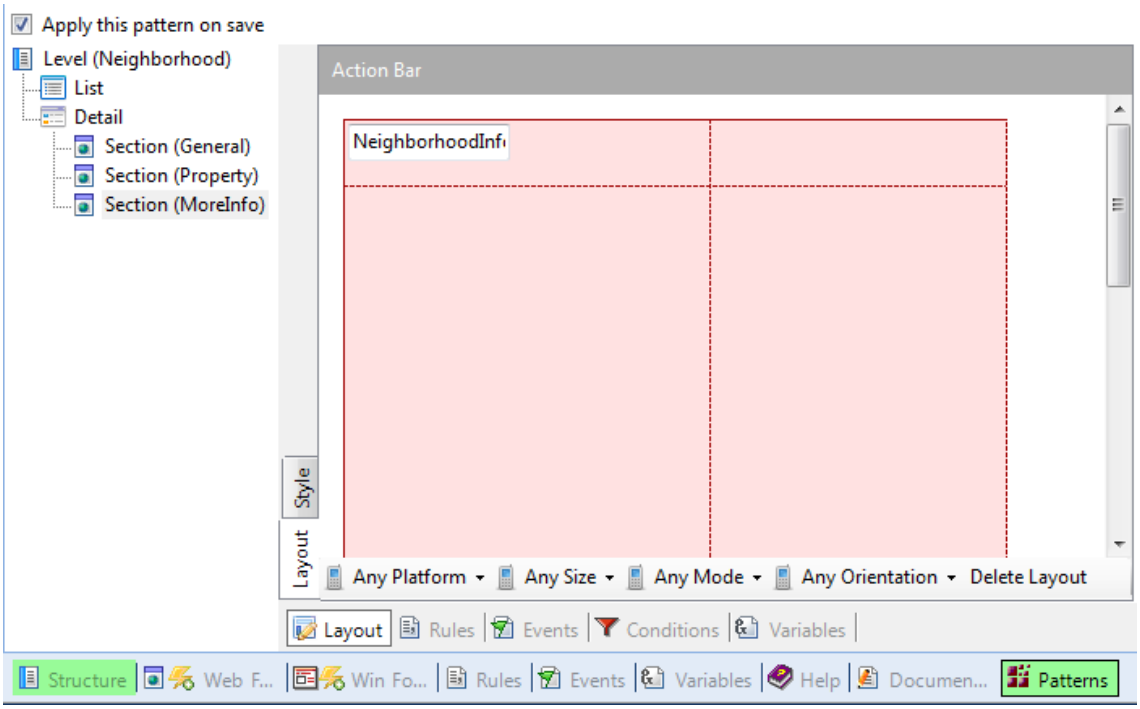
Como o programa executado sabe qual é o bairro em questão? Porque é recebido como parâmetro em cada seção {mostrar regla parm en c/section}, assim como no detail…

Podemos personalizar o layout desta seção assim como fizemos antes com o List de imóveis, por exemplo, pedindo que mostre as propriedades num mapa, ao invés de uma lista.

Se incluirmos o atributo *NeighborhoodInfo* à transação Neighborhood, do tipo Varchar(1024), de modo a guardar a informação geral do bairro…

…e quisermos mostrar essa informação numa seção independente, apagamos o atributo automaticamente colocado na Section(Geral) e criamos uma nova seção, que chamamos, por exemplo, 'MoreInfo' que, por padrão, aparece com o Layout vazio e em "Any Mode" (isto é, esta tela será a utilizada tanto em Edit como em View).

… e inserimos ali o controle atributo *NeighborhoodInfo*. Tiramos a etiqueta e especificamos que o campo possa crescer de acordo com seu conteúdo.



Na seção de regras, determinamos para ele a regra **parm** para que, automaticamente, filtre por bairro.

Especificamos o Caption que será visto em execução… {Info} e também podemos associar a ele uma imagem, assim como a outras secciones. {hacerlo}

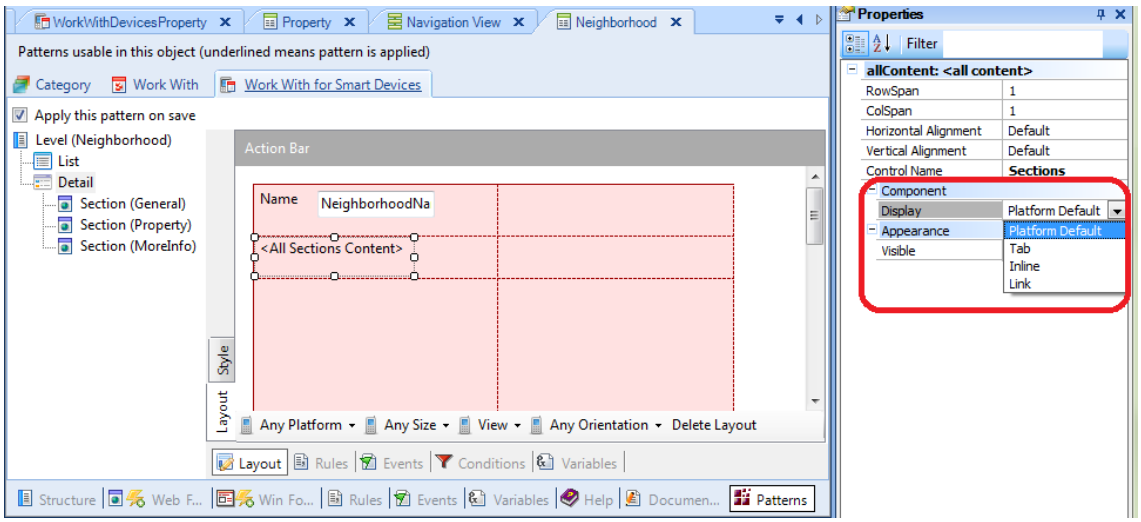
Vamos vê-lo em execução. {(Acá hay que compilar para que se vean las imágenes)}…

Uma vez determinada a informação dos bairros…

Mas poderíamos querer modificar esta forma de apresentação das seções em guias, para que seja exibida de forma diferente. Onde fazer? Quais possiblidades me oferece cada plataforma?

Vamos ao hub Detail…

Este layout tem um controle container <All Sections Content> onde serão carregadas as diferentes seções com o tipo que especificado na propriedade Display que, por padrão, está com o valor “Platform Default”. Isto se deve ao fato de que, dependendo da plataforma, será a forma mais adequada de mostrar a informação (está relacionado ao tamanho da tela).



Vemos que aparecem 3 opções: **Tab** corresponde ao default do Android e do BlackBerry, e estabelece que cada seção seja mostrada em uma guia. **Inline** significa que cada seção será apresentada na mesma tela, uma continuação da outra. **Link**

significa que cada seção, ao invés de ser apresentada na tela do detail, aparecerá em telas independentes através de links.

El default de iOS será para seções com grid mostradas como links e, as demais, inline.

Com esta propriedade {señalar la detail} e dentro das possiblidades de cada plataforma, pode variar a opção padrão, de modo que todas as seções sejam apresentadas uniformemente de outra maneira. Por exemplo, no Android se consegue mostrá-las como Links.

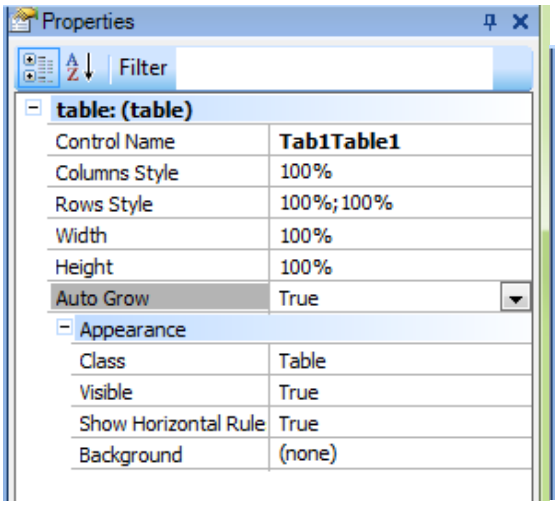
Mas, se quisermos, por exemplo, apenas mostrar a informação Geral e a More Info dentro de tabs, e mostrar os imóveis do bairro dentro da informação geral como um Link, então esta propriedade que aplica a todas as seções uniformemente não nos serve. {callout con imágenes que representen en ejecución lo que digo}

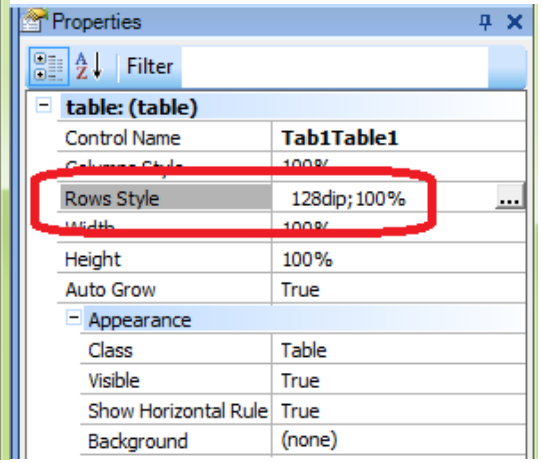
Vamos apagar este controle. Ele aparecerá na toolbox para poder ser reinserido, mas também aparecerá um controle para cada seção do Detail, para que possam ser inseridas independente das outras e, assim, deixar seu comportamento independente.

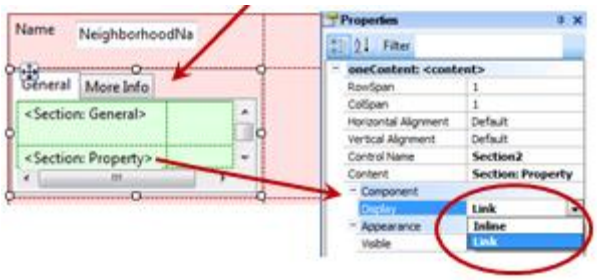
Por exemplo, podemos inserir um tab control da toolbox e na primeira página colocar a <Section Geral> … {hacerlo} e a <Section Property> {hacerlo}. Entre as propriedades deste item, configuramos o Caption {Geral} y podemos associar a ele uma imagem. {hacerlo} Na segunda página do tab control podemos colocar a <Section MoreInfo> {hacerlo, Caption= More Info y también agregarle imagem}. O terceiro tab foi eliminado. {hacerlo}

{ir a la primera tab page}

Na tabela, vamos ajustar o tamanho das linhas para que caiba toda a informação:





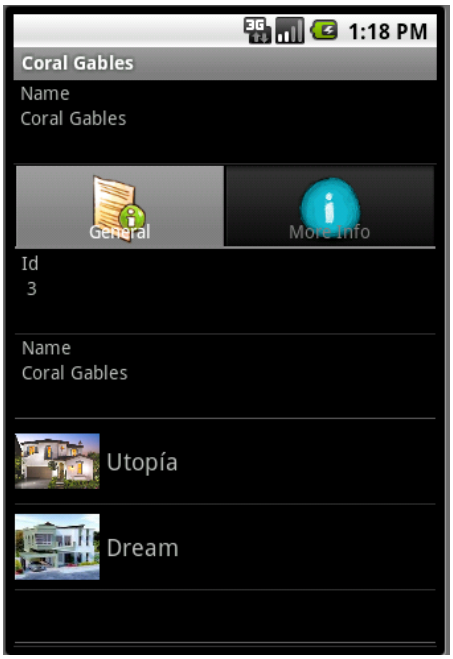


Podemos observar que a propriedade **Display** das seções individuais assume 2 valores ao invés de 3:

- Inline (default)
- Link

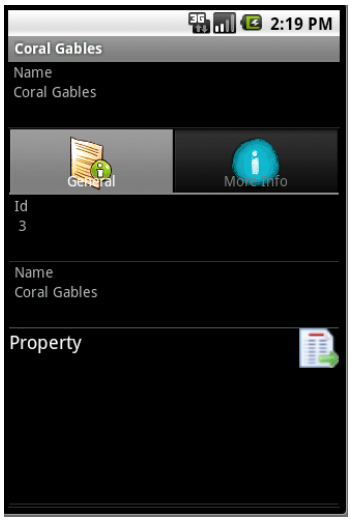
Por exemplo, no caso de <Section Property>, se refere a como queremos que apareça a lista de imóveis associados ao bairro que está sendo visualizado:

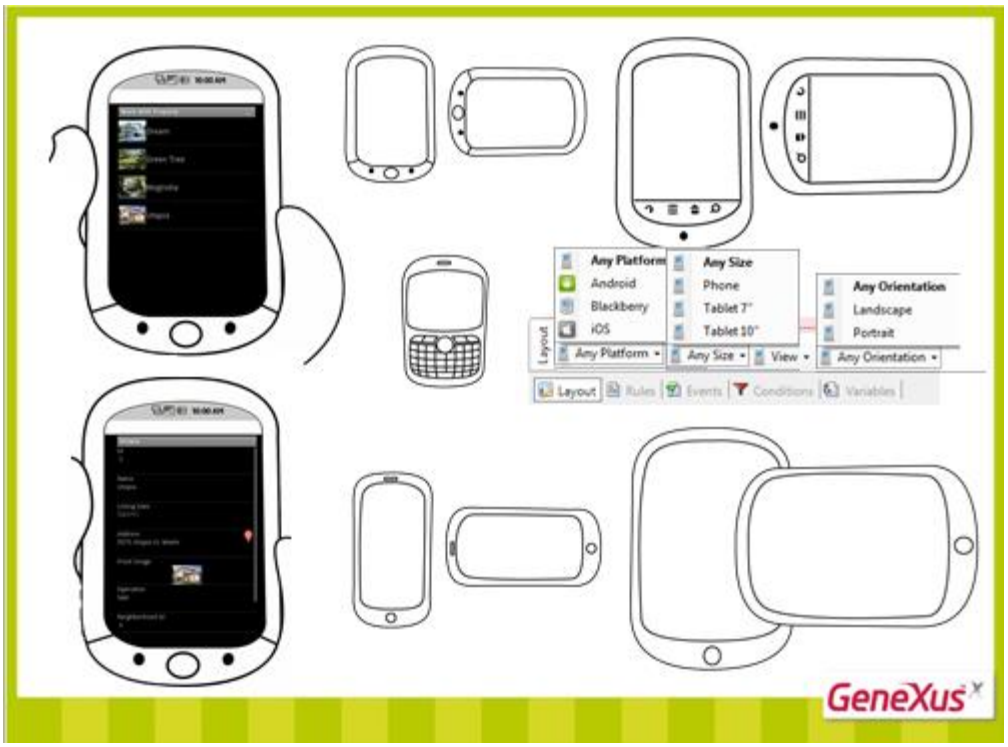
1. No mesmo form, como linhas sob a informação plana geral (vamos ver em execução)... {mostrar en ejecución, con callout } ó



(callout TabcontroleCon2SectionsInline.jpg)

2. Como um link, que é o que queremos. {elegir esta opción}





As telas do List e do Detail foram desenhadas tendo em mente um Android Phone, que tem um determinado tamanho de tela. Mas podemos variar este desenho se a tela é a de um tablet (porque nela cabe mais informação). Neste caso, também de acordo com a medida de 7 ou 10 polegadas.

Ou podemos variar a tela de acordo com a plataforma, para seguir as guidelines de cada uma,

…ou também desenhar diferentemente as telas quando vão ser visualizadas verticalmente (portrait) ou horizontalmente (landscape).

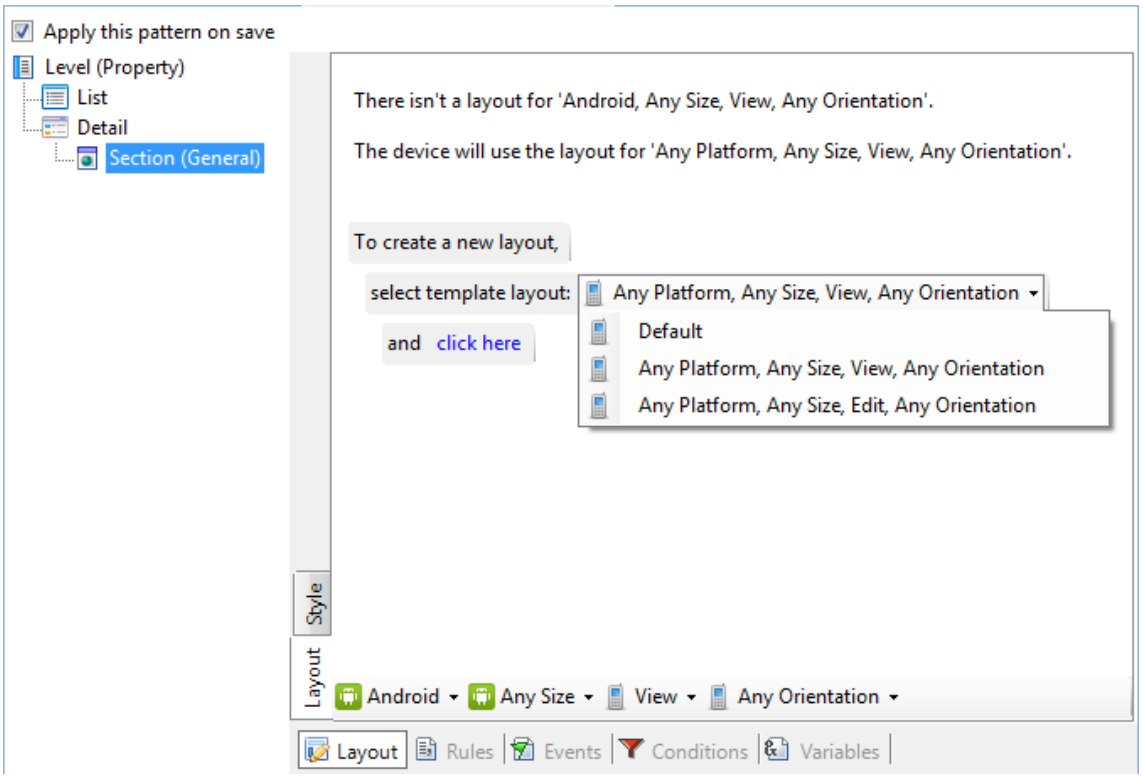
Como podemos considerar estas variações?

Para isso, contamos com combos na borda inferior de cada Layout.

Um permite escolher a plataforma, outro permite selecionar o tamanho, e outro permite a orientação.

A opção padrão é Any Platform, Any Size, Any Orientation, o que significa que teremos apenas um Layout para View e um para Edit, que valerá para todas as plataformas, todos os tamanhos e todas as orientações.

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



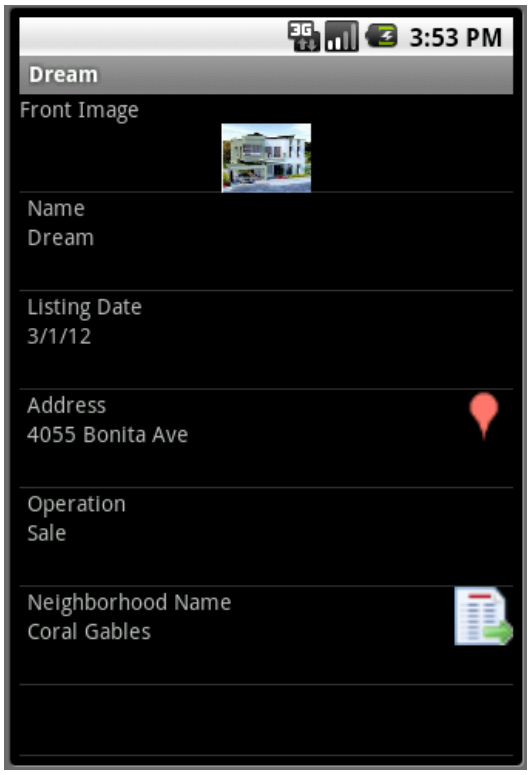
Mas se criarmos um layout específico para uma plataforma determinada, por exemplo, para Android, aparecerá uma mensagem como a que vemos acima, que nos indicará que este Layout não existe, e nos escolher se for inicializado vazio, (Default) {hacerlo} ou (apaga-se este layout) {hacerlo} de acordo com a tela de Edit {hacerlo} (que não tinha NeighborhoodName), ou {borrarlo} com a de View (que não tinha os identificadores nem a geolocation). Observamos que agora nos mostra as etiquetas acima, tal como é o padrão do Android.

Vamos mudar, por exemplo, para o layout do view no Android, o lugar onde a imagem é mostrada. {llevarla para arriba}

Neste momento, temos dois Layouts diferentes definidos para o modo View. {elegir Any Platform y luego vovler a Android}

Qual deles será utilizado no aplicativo gerado? Quando for gerado para Android, este que vemos, e quando for gerado para qualquer outra plataforma, este. {pasar de uno a otro}

Vamos ver o que mostrava o emulador do Android antes da mudança {mostrarlo}, e na próxima geração… F5 {mostrarlo}



Um mesmo aplicativo provavelmente requisitará um look & feel diferente ao variar a plataforma.

Assim, esta agenda de um Encontro de Usuário de GeneXus, apresenta diferenças de desenho de acordo com a plataforma escolhida.

Também sem mudar a plataforma, podemos associar certo desenho a um aplicativo desenvolvido para um determinado cliente, e outro para o mesmo aplicativo, só que para outro cliente.

Precisamos separar os aspectos de desenho dos de funcionalidade. Para que se possa tornar as tarefas paralelas e deixar as de desenho para um especialista e nos concentrar como desenvolvedores na construção do aplicativo.

Como?

Continua.

FIN VIDEO 2

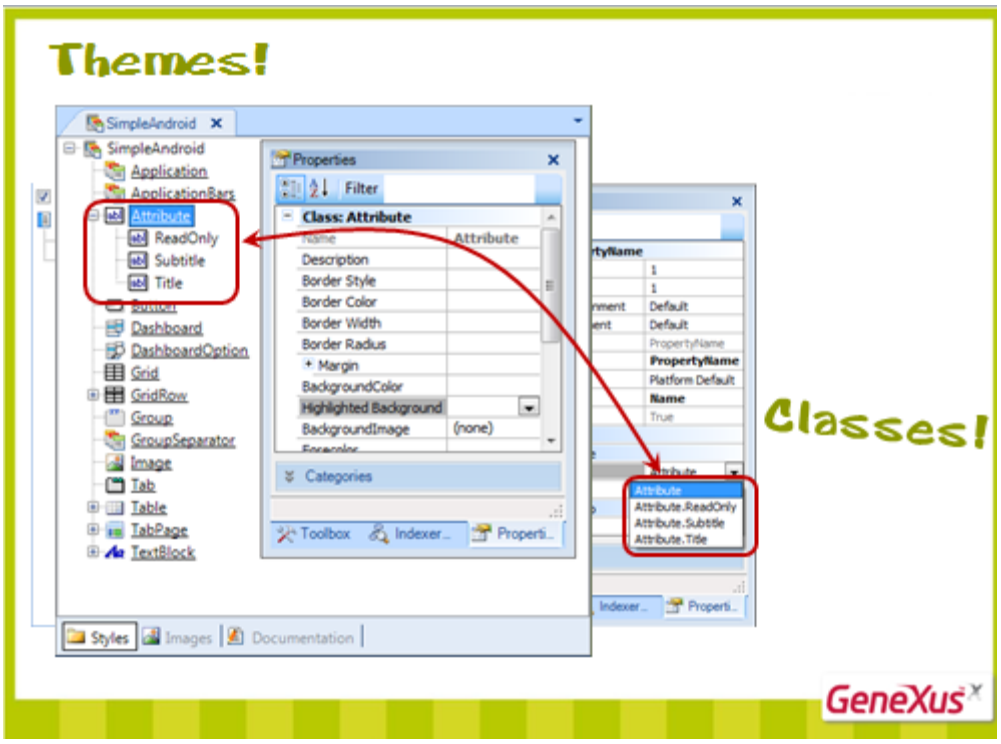
Video 3: Desenho através de classes e Themes

Um mesmo aplicativo provavelmente requisitará um look&feel diferente ao variar a plataforma.

Assim, esta agenda de um Encontro de Usuários de GeneXus, apresenta diferenças de desenho.

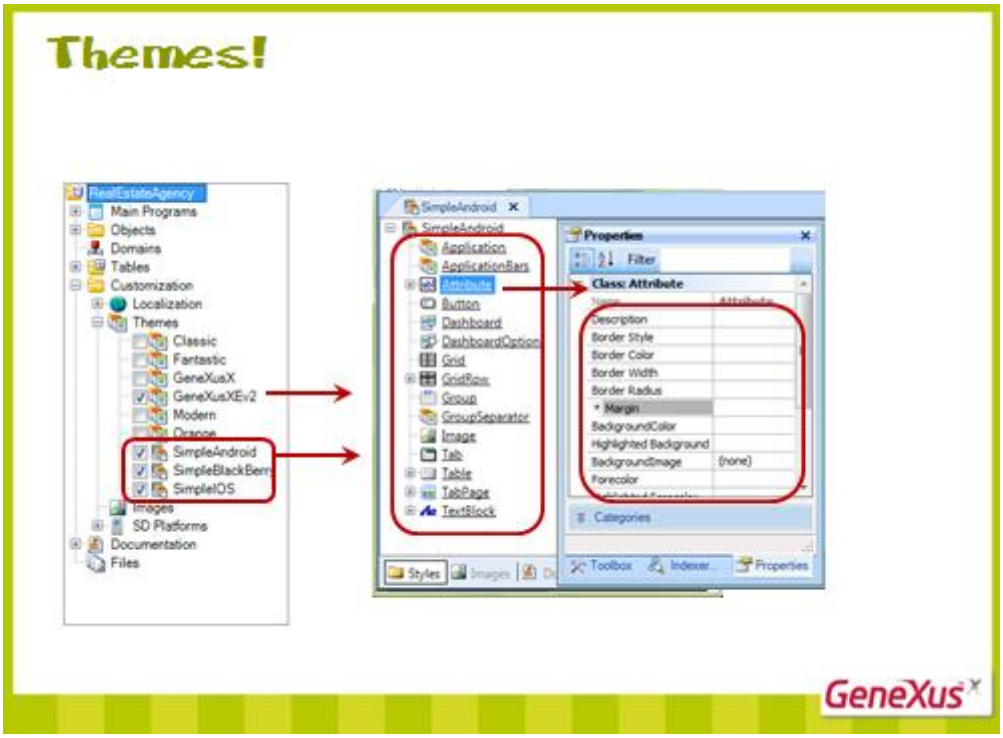
Também sem mudar de plataforma, podemos associar determinado desenho a um aplicativo desenvolvido para um cliente, e outro para o mesmo aplicativo desenvolvido para outro cliente.

Para separar os aspectos de desenho dos de funcionalidade, de modo que se possa paralelizar as tarefas, deixando as de desenho para um especialista e nos concentrar, como desenvolvedores, na construção do aplicativo, são associadas “classes” aos controles dos Layouts e, assim, existem os objetos themes (temas) que concentram as especificações de desenho de cada uma dessas “classes”.



Deste modo, um controle atributo pode receber uma destas quatro classes, que estão configuradas em um lugar centralizado, neste caso, para esta plataforma.

O lugar centralizado é o objeto Theme.

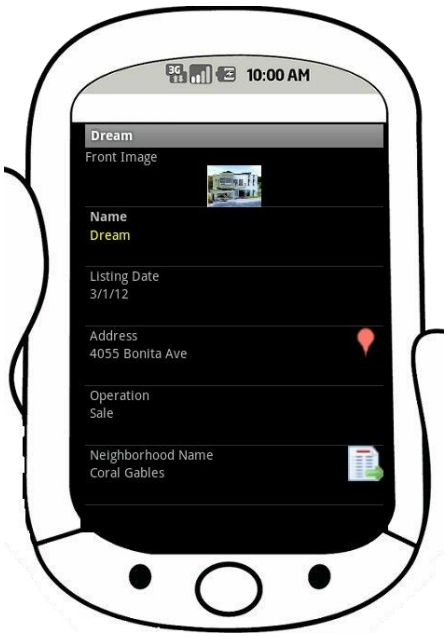


Por padrão, além do Theme default para o aplicativo web…

existem três themes para Smart Devices, um para cada plataforma.

Em cada Theme são configuradas as propriedades de cada classe (de controle, de usuário, etc.)

Por exemplo, queremos que o atributo PropertyName saia em negrito e amarelo no Android.



Para isso…

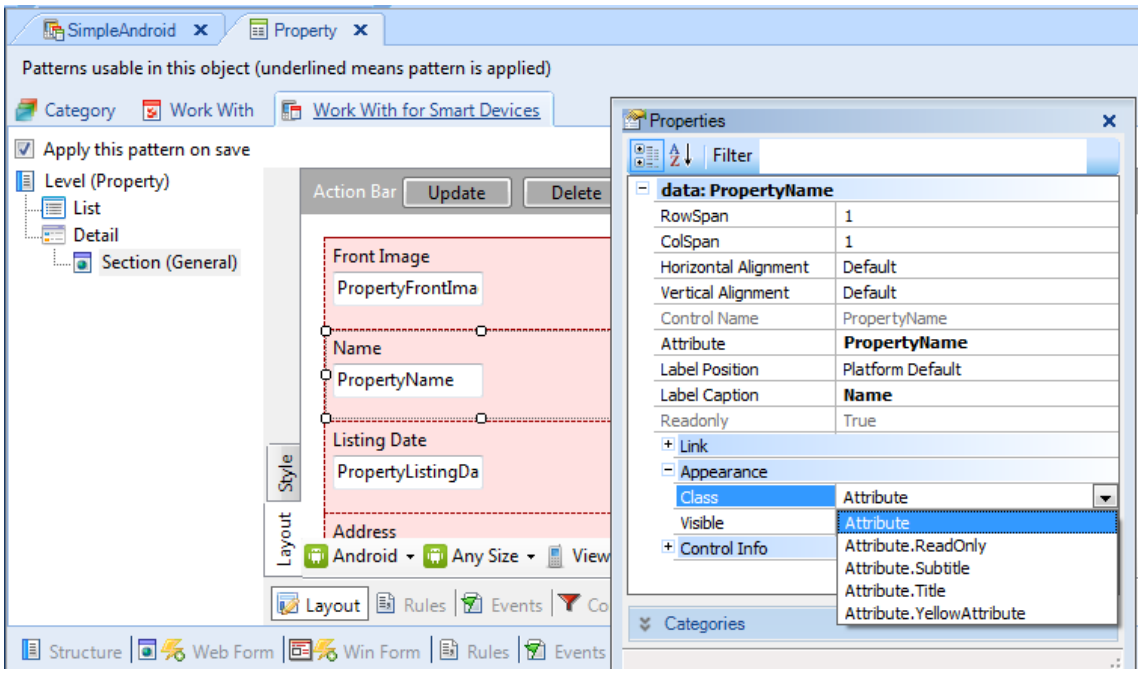
{demo}

Abrimos o theme padrão para Android. Vamos criar uma nova classe de atributo {hacer AddClass} chamada YellowAttribute.

Configuramos a propriedade Forecolor em yellow e a Label class Textblock.Title, que corresponde a esta outra classe {mostrar abajo la class TextBlock} e a colocamos em negrito.

{grabar}

Agora vamos ao layout do view para Android e vamos mudar a classe do atributo PropertyName, associando a ele uma nova classe.



Executamos… {mostrar}

Onde se configura o theme que será utilizado?

Nas Preferences {ir} Aqui os valores gerais default estão centralizados e todas as instâncias do pattern Work with para Smart Devices.

Entre as coisas que podem ser configuradas, se encontram os themes.

A plataforma AnyAndroid tem configurado o theme que acabamos de modificar.

Já o Blackberry tem… e iOs tiene o terceiro theme que vimos.

Podemos modificar o theme de cada um.

As precedências são as usuais: Se estamos gerando para iPad, como não tem um theme definido, utilizará o de AnyIOS.

FIN VIDEO 3

Video 4: Work With for Smart Devices – Ordens, buscas e filtros

Após ver como personalizar os Layouts, vamos estudar como personalizar a ordem como as informações são apresentadas, além dos searchs e filtros disponibilizados ao usuário…

{demo, mostrar ejecución}

Se observamos o List de imóveis, se está ordenando a informação mostrada por nome da propriedade {pasar el mouse por la letra inicial del nombre de c/propiedad}. Além disso, está permitindo filtrar a informação por data {clicar} dentro de um intervalo, por operação {clicar} e por bairro {clicar} (como é a foreign key), abrirá uma lista de seleção.

Logo veremos por quais atributos se pode realizar o Search.

Onde se configura tudo isso?

Na seção conditions da instância do pattern.

Existe apenas uma ordem definida, de nome Name {señalar la propriedade Name}, composto pelo atributo PropertyName (escolhido por ser o description attribute da transação).

Podemos ver que o Search está definido por PropertyName e por PropertyAddress {ir a ejecución} Assim, se podo filtrar por endereço, {poner Bonita} ou por nome da propriedade {poner Uto}

E com o **Advanced Search** podemos ver os atributos pelos quais podemos filtrar.

Tudo isso é personalizável. Por exemplo, se deseamos que o usuário possa escolher entre ordenar os imóveis por Nome ou por Bairro, então devemos incluir uma nova ordem {hacer Add Order y nombrarlo Neighborhood} composta por {hacer Add Attribute} NeighborhoodName.

Vamos aplicar a mudança. F5…

{transición para mostrar en ejecución}

O Order foi incluído nesta ação. Está selecionado ordenar por nome, mas o usuário pode escolher ordenar por {cliquear} bairro…

Se quisermos que, al escolher esta opção, as propriedades apareçam agrupadas por bairro, então… {volver a GX}

…especificamos que realize um corte desta ordem…{marcar Break by true} pelo atributo do order.

{ejecutar}

E se quisermos que, além de agrupar por bairro, ordene, dentro do bairro, pelo nome da propriedade…

{hacer Add Attribute y agregar PropertyName} aqui é importante a ordem na qual parecem listados estes atributos (não é o mesmo que ordenar por bairro e, dentro do bairro, por nome, ao contrário). Falta apenas que queremos indicar que o tipo de agrupamento seja feito desde o primeiro atributo do order até o atributo… Bairro (neste caso é o mesmo). {hacerlo}

{ejecutar} Vemos que agora está ordenando dentro de bairros.

{volver a GX}

Podemos incluir ou apagar atributos de busca {hacer botón derecho sobre Search y Add], e podemos fazer o mesmo para as buscas avançadas, que são as que aparecem

como Filter em execução e que têm como agregado que se especifica a expressão que determina o tipo de filtro a realizar {pasar por los atts y mostrar la propriedade Expression}. Estas variantes internas {mostrarlas en el Expression} correspondem aos controles que aparecem para que o usuário inclua valores {mostrar en ejecución}…

Mas também temos a seção de Conditions gerais, que estabelece condições sobre a informação que não serão apresentadas ao usuário para serem escolhidas. Se quisermos que sempre saiam listadas as propriedades que estão à venda, estabelecemos então que as propriedades que liste cumpram sempre esta condição (além das outras que o usuário escolha através do Search e Advanced Search)…

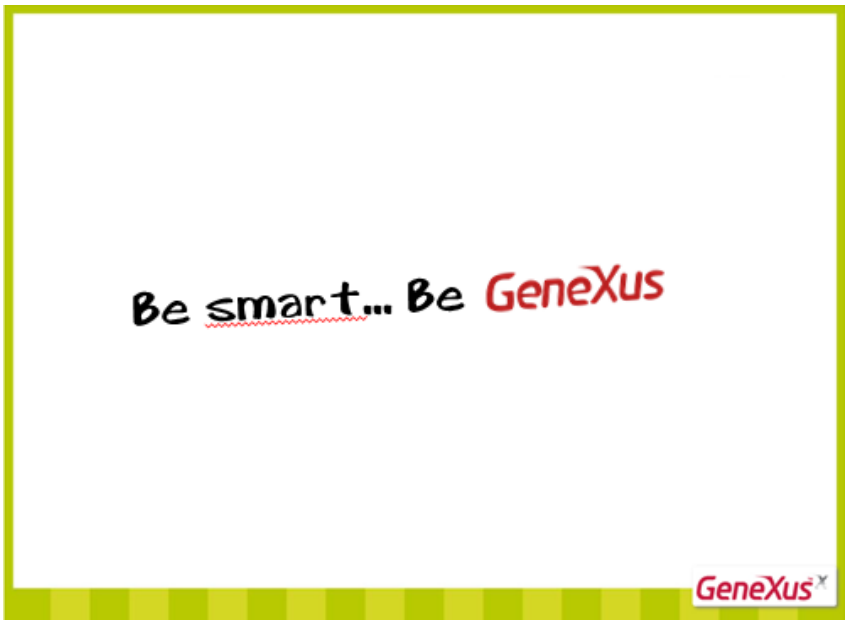
{agregar PropertyOperation = Operation.Sale;)

No Detail está mais limitado, já que a informação mostrada, em geral, não é repetitiva. {pasar por el detail}

Deseja programar comportamento?

Continua…

FIN VIDEO 4



Video 5: Work With for Smart Devices – Comportamento através de eventos

Após ver como personalizar os Layouts, a ordem pela qual se lista a informação, as buscas e os filtros, agora veremos como personalizar o comportamento através de Eventos.

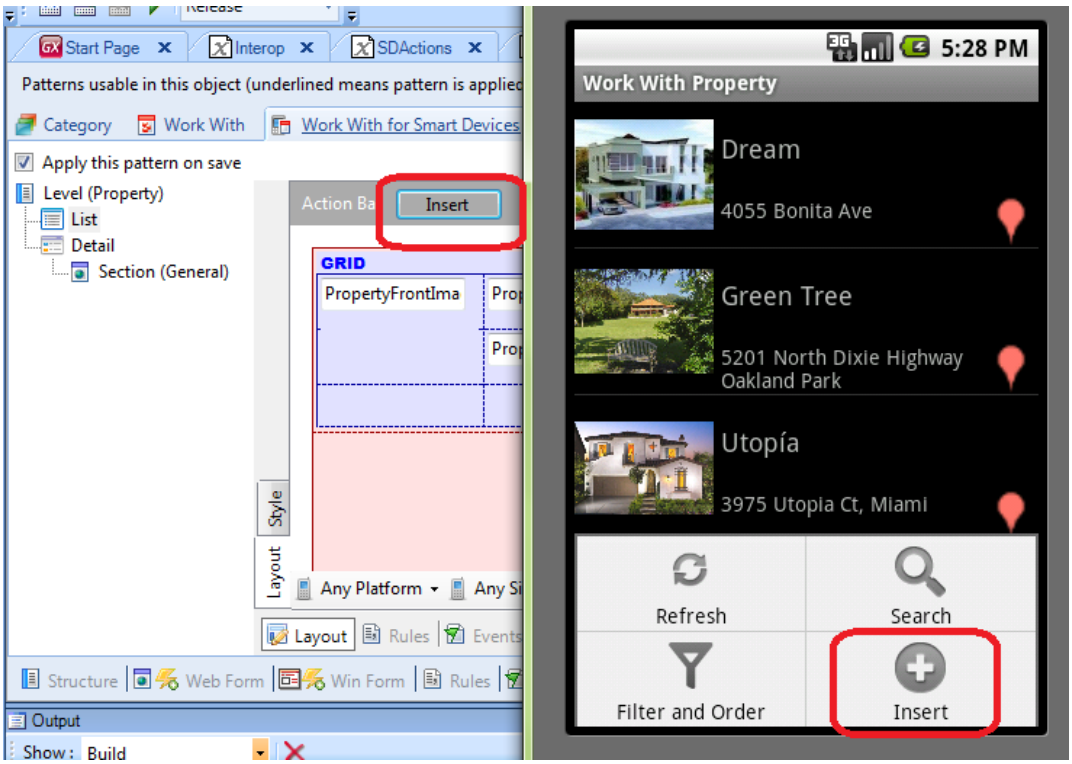
Existem ações pré-determinadas para cada tela. Por exemplo, quando estamos no List, temos no **menu** um botão para inserir um novo elemento na lista.

Na tela de Detail, se estivermos no modo Edit ou no View, teremos algumas ações pré-determinadas. Por exemplo, se estamos em View, podemos ver no menu as ações Update e Delete, que nos levarão à mesma tela, mas no modo Edit.

As ações no GeneXus são conhecidas como eventos.

{demo}

Vamos começar pelo List. Na Action Bar se encontram as ações que serão incluídas ao menu do 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: }

Trata-se de um botão.

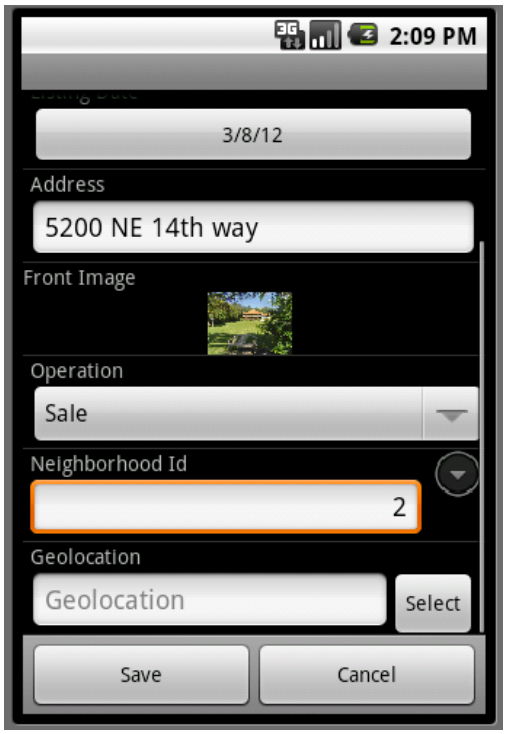
Quando pressionado, {presionar en el emulador} abre a tela do Detail no modo Edit, para que o usuário inclua um novo imóvel. Onde isto está programado? {volver a GX}

Na seção de Eventos. {abrirla}

Aqui vemos o Evento associado e o código que será executado toda vez que este evento for produzido. Será acionado o WorkWith for Smart Devices da transação Property, level Property {mostrar con el mouse} e Detail {mostrar con mouse} , no modo Insert {mostrar sobre código de evento}, sem passar parâmetros, o que significa que será aberto o Layout de Edit… {mostrarla, para eso primero pasarse al Layout, luego mudar al nodo Section(Geral) y ahí elegir Edit. Mostrar enseguida la pantalla abierta del emulador que quedó en esa misma pantalla}

{volver al Layout}

Nesta tela de Edit, que corresponde a Section(Geral), também podemos ver dois botões no menu. Uma vez carregados os dados da nova propriedade {cargar nombre MyProperty…} vamos gravá-la, Save {señalar} ou, pelo contrário, podemos cancelar {señalar} e voltar à tela anterior. {presionar esto}



{pasar a GX} Na Action Bar vemos esses dois botões. {presionar sobre Save y mostrar en las propiedades arriba del todo action:Save y hacer lo mismo sobre Cancel}.

O que esperamos encontrar como código do evento Save? Estamos no dispositivo, {volver al dispositivo con todo cargado} temos toda a informação incluída pelo usuário para este novo imóvel, ao fazer Save {mostrar botón} teríamos que enviá-la ao servidor (neste caso, na nuvem) {callout con server en la nube} para pedir que execute a lógica da transação Property com esses dados {callout con trn}, criando um novo registro na base de dados {callout con DB} para essa propriedade. Ou seja, temos que pedir que

execute o BC Rest {hacer Save}e que nos responda se a operação teve ou não sucesso, de modo que possamos informar ao usuário do dispositivo. {mostrar el texto que sale} Mas toda esta lógica, que depende da plataforma do dispositivo {ir a GX, y editar los Events}, está encapsulada em uma API fornecida pelo GeneXus, SDActions {señalarla}. Ao acionar o seu método Save() {señarlarlo} será feito o que esperamos.

Quando o usuário pressiona Cancel, voltaremos à tela da qual partimos. Por exemplo, {ir a emulador} se quisermos fazer um Insert… {desde el List hacer Insert y allí cancelar} voltar ao List. E se queremos fazer um update {ingresar a una propriedade con tap y luego presionar Update} desde o View da propriedade, {presionar Cancel} voltar a esse View. {volver a GX} Isto é o que implementa o método Cancel() da Api.

Por outro lado, no View temos os botões Update e Delete {mostrarlo en el emulador en la ventana que quedó abierta}. Esses botões são vistos na Action Bar do layout correspondente {movernos en GX al View}

Ao selecionar Update, o comportamento esperado é que abra o Layout Edit, para que assim o usuário possa modificar a informação desse imóvel. Vamos ver o código do evento {volver a GX, hacer botón derecho, Go to Event} Se está acionando o Detail, do WorkWith de Property {señalar}, level property {señalar}, no modo Update, passando como parâmetro o PropertyId recebido como parâmetro. Desta forma, entende que tem que abrir o Layout Edit, listando os valores do imóvel com esse identificador. {mostrar el emulador. Luego hacer Cancel }

Por último, se pressionamos Delete, {hacerlo} também queremos que se acione o Detail, mas no modo Delete, de modo que o usuário possa escolher se elimina a propriedade {señalar botón}, ou cancela a operação {señalar botón}.

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

Por que aparece este return? Porque quando estamos no View, {vovler al view} ao apagar o registro que estávamos vendo, desde este botão {mostrar Menú/Delete} queremos apagar o registro que estamos visualizando e, depois de apagá-lo, queremos voltar ao que chamou a este View em primeira instância. {hacerlo}

{ir a GX, código de evento Delete} Aqui podemos ver que temos dois comandos consecutivos dentro do evento. Quando, dentro de um evento executado no dispositivo, precisamos fazer duas ações, temos que utilizar o comando Composite. Vamos estudá-lo em breve.

{ppts}

Acabamos de ver eventos pré-determinados tanto para o List como para o Detail, programados automaticamente pelo pattern.

No entanto, podemos definir outros eventos, colocá-los em diferentes lugares do Layout (não apenas na Action Bar para que apareçam no menu) e utilizar diversos comandos para programar o comportamento desejado.

Temos dois tipos de eventos: os do sistema, cujo código será executado no servidor,

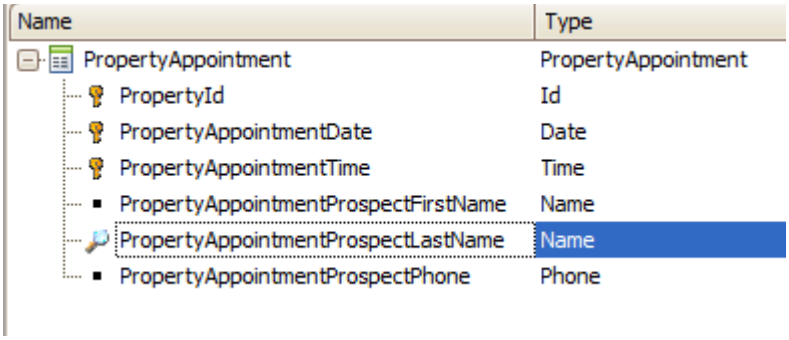
e os de usuário, cujo código será executado no dispositivo (independentemente da possibilidade de estabelecer comunicação com o servidor através de serviços de chamada rest).

Os eventos do servidor serão vistos mais adiante. Vimos alguns eventos do cliente, pré-definidos,

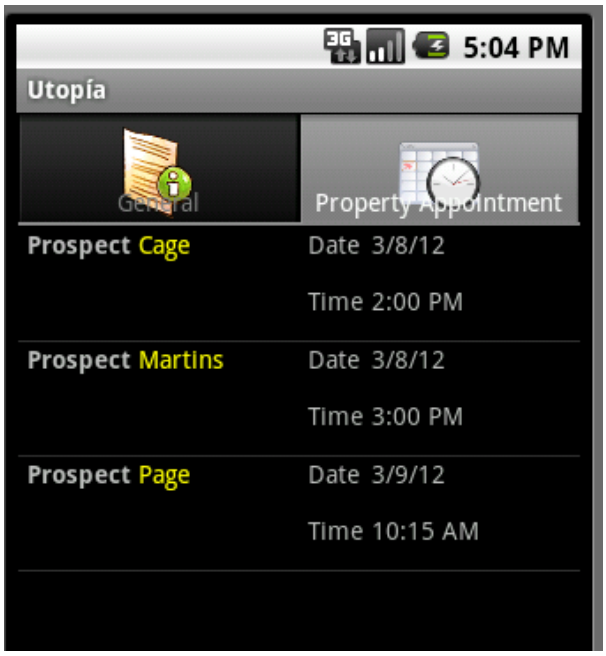
e agora definiremos algum nosso…

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

Incluimos uma transação {señalar el nome} para agendar visitas aos imóveis {señalar PropertyId}, numa data {señalar } e uma hora {señalar }determinadas, de possíveis clientes, dos quais armazenamos seu nome {señalar }, sobrenome {señalar }e telefone {señalar }.



Aplicamos a ele o pattern work with. {mostrar que está marcado} Foi incluída ao Work With de Property, automaticamente, {pasarse a ese} uma Section que mostrará as visitas: sobrenome do prospect, data e hora {señalar} associadas à propriedade visualizada {mostrar regla parm}. A esta propriedade foram associadas imagensàs seções e melhoramos um pouco o aspecto do grid, para que, em execução, seja visto… assim…{mostrar en ejecución el dashboard, de ahí el list de propiedades y de ahí elegir una propiedad y ver las appointments}:



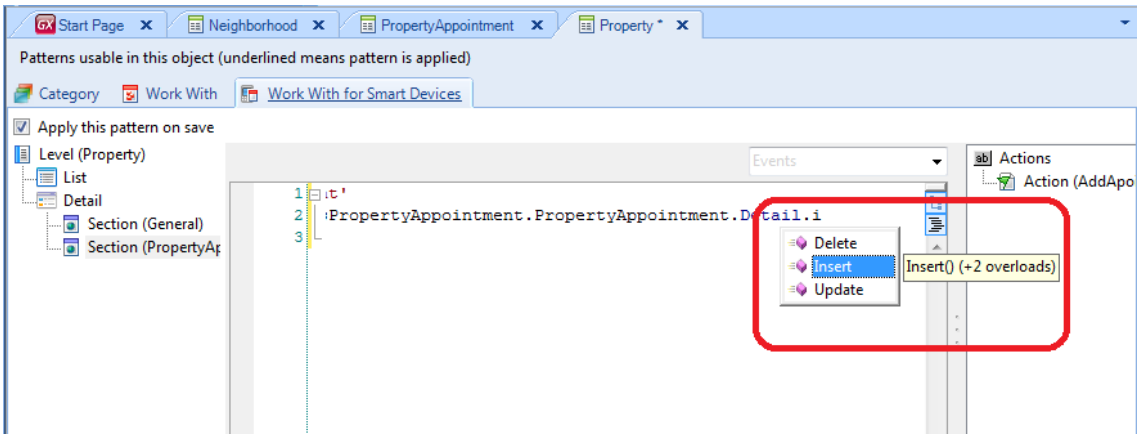
Se fizermos tap sobre uma das visitas {hacerlo}, vemos que nos mostra sua informação geral: isto é {ir a GX}: o node Detail del Work with de PropertyAppointment, em modo View. Em todo o grid se configura a ação padrão que ocorrerá ao fazer tap sobre uma linha {ir a GX a mostrar}. O Default é sempre abrir o view.

{volver a emulador} No menu {abrirlo} só vemos a possiblidade de atualizar a tela, mas não há outras ações definidas. Queremos dar a possiblidade de inserir uma nova visita para esta propriedade {señalar el nombre arriba del caption} usando esta tela. Ou seja, devemos acionar este mismo detail, {volver a hacer tap} mas ao layout de Edit para fazer un Insert.

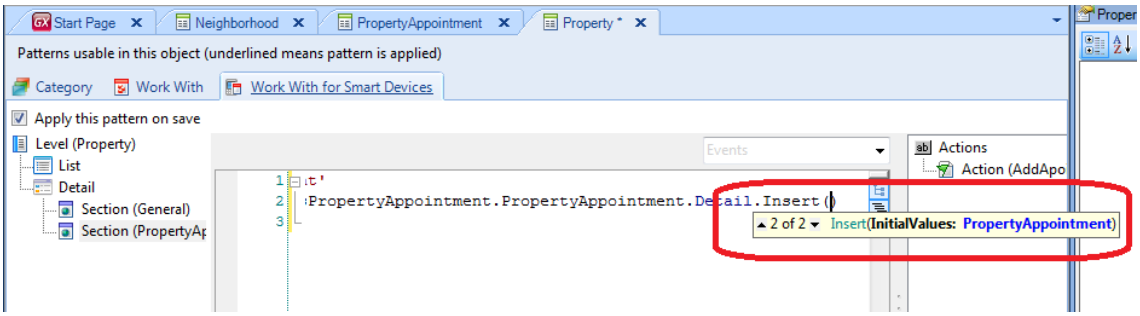
Para isso, {volver a Gx} com o botão direito sobre a Action Bar {hacerlo}, selecionamos inserir uma ação {hacerlo}, a qual chamaremos AddAppointment {hacerlo}.

Editamos as propriedades do botão e diminuímos o Caption (o que será visto em execução). Agora programemos o código do evento associado {botón derecho Go to Event}.

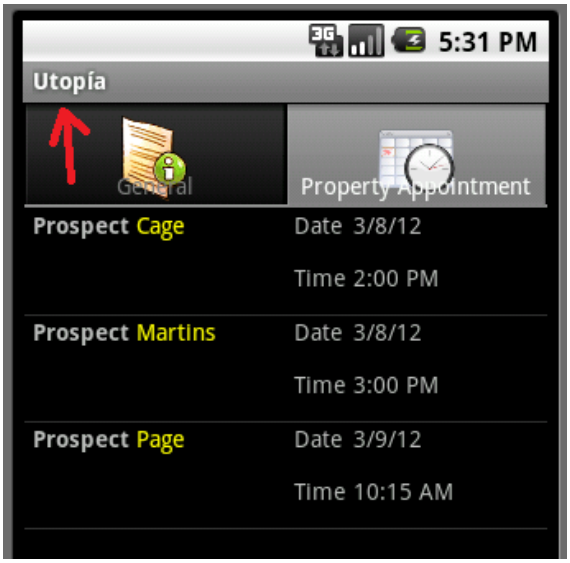
Arrastamos o nome do Work with {hacerlo}, pressionamos ponto {hacerlo} o que segue é o nível. É colocado automático porque há apenas um {mostrar en el ide, cambiando de tab al de PropertyAppointment}, y logo a quem queremos telefonar desse work with? Al Detail {elegir propiedad} no modo Insert {elegir}



Isso indica que temos 2 possibilidades: não enviar parâmetros, ou enviar como parâmetro uma variável do tipo business component da transação PropertyAppointment.

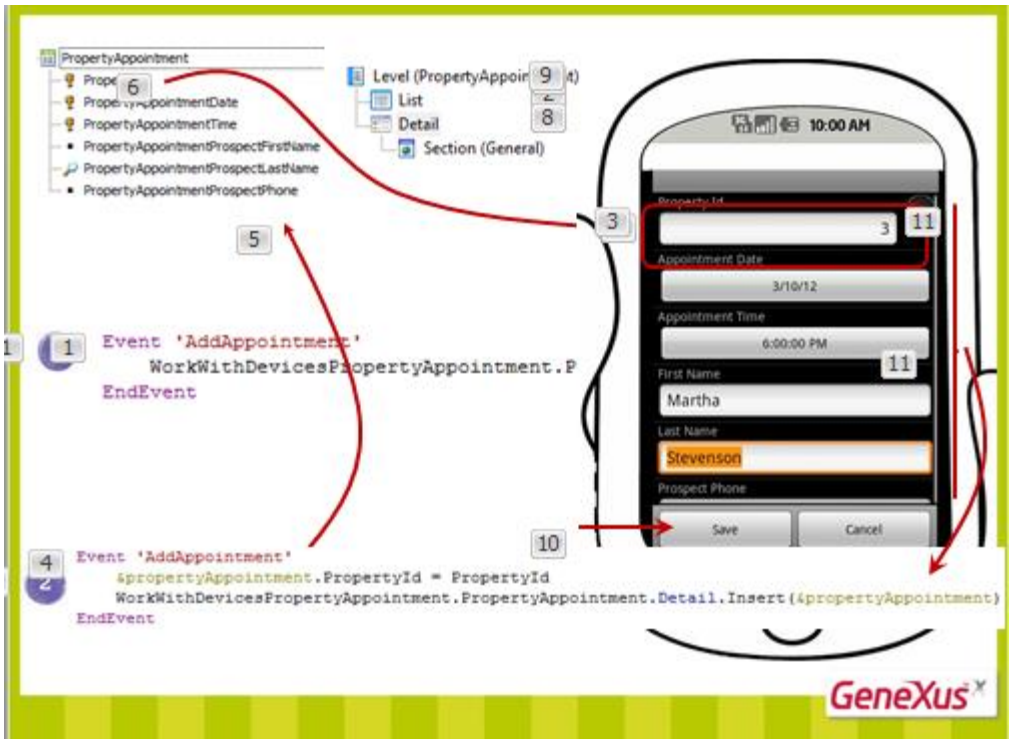


Neste caso, precisamos enviar um parâmetro, o identificador de propriedade, porque vamos inserir uma nova visita,



Mas, para esta propriedade {señalarla}

{ppts}



Então, esta primeira alternativa não nos serve, pois queremos que a propriedade fique instanciada.

Temos que usar a segunda alternativa. Nela definimos uma variável de tipo BC de PropertyAppointment e, nessa variável somente inicializamos o campo que nos interessa enviar com o valor, que es PropertyId, depois do qual…

Será interessante que, ao fazer isso, uma vez completados pelo usuário os campos desta tela, ao acionar o Save, será devolvida a mesma variável carregada com todos os valores, para que possa ser utilizados logo, como veremos…

{demo}

Feito isso em GeneXus… já definida a variável {señalarla} do tipo BC de PropertyAppointment. E já atribuída sua propriedade PropertyId com o valor do atributo PropertyId recebido pelo parâmetro {mostrar} e enviada por parâmetro, vamos ver como ficou em execução…

Na lista de propriedades, escolhemos uma e vemos suas visitas agendadas… Agora no Menu aparece o botão Add (poderíamos associar-lhe uma imagem) e ao pressionar… vemos que inicializou com o id dessa propriedade.

Incluímos valores {fazê-lo} e ao fazer Save, volta y atualiza a tela onde podemos ver a visita recém incluída.

Se fizermos tap sobre ela, serão mostrados todos os seus dados.

Aparece esta imagem {la del telefonito}, e ao fazer tap sobre ela, será aberta a agenda de contatos do dispositivo, para telefornarmos para esse número. Como sabia disso? {volver a GX}

Porque o domínio do atributo é Phone. Vemos, mais uma vez, que com os domínios semânticos, o aplicativo se integra com as funcionalidades do dispositivo de forma transparente para nós.

Mas além desta forma transparente de integração, temos outras explícitas, que são através das distintas APIS que nos fornece GeneXus.

Por exemplo, si quisermos que, una vez incluída a visita à base de dados {hacer back y mostrar el view de la visita}, os datos do prospect {señalarlos} fiquem registrados na agenda de contatos do dispositivo, {ir al Evento} teremos que incluir aqui uma chamada à agenda de contatos passando-lhe os três dados que temos: nome, sobrenome e telefone do contato. Mas, cómo interagimos com essa funcionalidade do dispositivo, que vai depender de cada plataforma?

Através de uma api provista por GeneXus, AddressBook… {ir al folder SmartDevicesApi}

Ao abrir este objeto externo, vemos que nos fornece métodos para incluir um contato, eliminá-lo, ou visualizá-lo.

Então, o que faremos é…

acionar a agenda do dispositivo {arrastrar al Evento}, através desta api, utilizando o método AddContact… a ele vamos passar todos os parâmetros requisitados (todos e na mesma ordem). Primeiro, o nome. De onde tiramos? Veio no BC… {&propertyAppointment.ProspectName}, depois nos pede o sobrenome {elegirlo}, em seguida o e-mail (como não temos, deixamos vazio) {'' hacerlo}, logo o telefone {elegirlo}, depois o nome da companhia; como não temos, deixamos vazio, depois a foto e, por último, uma mensagem.

Vemos que, ao gravar, pede o comando Composite, já que estamos realizando mais de uma ação. {agregarlo} Vamos estudá-lo em breve.

Vamos executar…

Depois de gravar a visita, será aberta a agenda de endereços para permitir a inclusão do contato com os dados que enviamos.

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

Temos todas estas APIS, que se agregam à kb no momento de utilizar o gerador para Smart Devices. Algumas requerem o uso de dados estruturados, pelo qual também são criados, automaticamente, os SDTs correspondentes para sua manipulação.

Por exemplo, vemos esta api GeoLocation, {abrirla} que necessita, para alguns métodos oferecidos, destes dois SDTs {señalarlos}. Esta api permite interagir com o gps do dispositivo e obter, por exemplo, sua localização em um dado momento, através do método GetMyLocation. Entre outras coisas, também permitirá registrar o caminho efetuado com o dispositivo em um intervalo de tempo dado, com o método StarTracking {señalarlo} , deter o registro com o EndTracking {señalarlo}, disparar alertas de proximidade, etc.

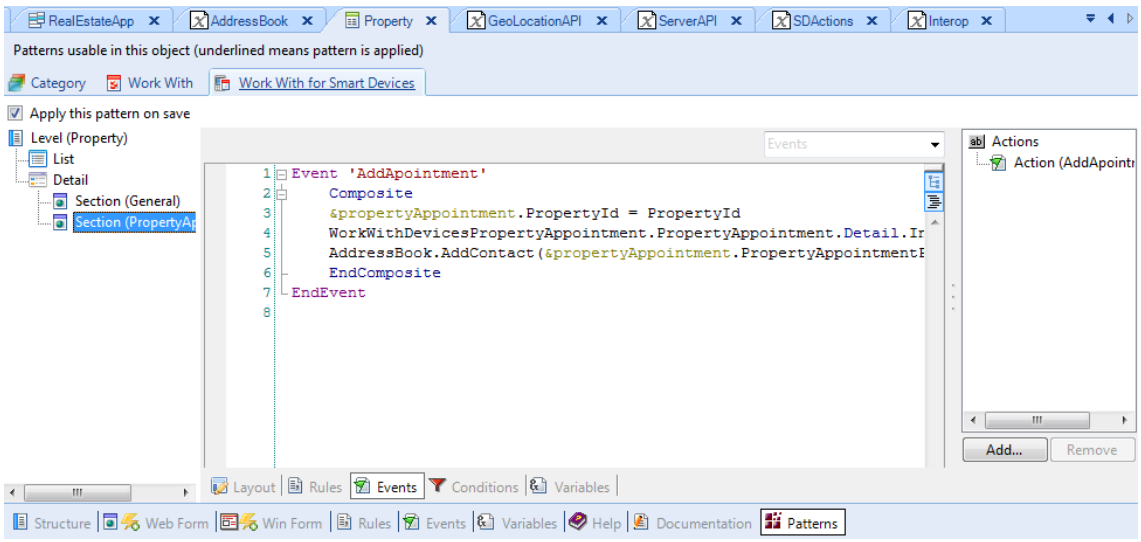
{abrir SDActions} Por seu uso generalizado, os métodos de algumas apis poderão ser utilizados sem que se refira ao nome da api. Já vimos um caso, o do método return {señalarlo} de SDActions. {abrir WW property → Section(Generla)→ evento Delete}

{volver a SDActions} O mesmo destino tem o método Refresh {señalar}, que será utilizado para obrigar a atualizar a tela. E alguns da api Interop {abrirla}.

{agregar callout con flecha sobre Msg } para mostrar uma mensagem na tela, {agregar callout con flecha sobre Confirm } para pedir confirmação ao usuário para continuar, {agregar callout con flecha sobre OpenInBrowser} para, por exemplo, criar um link dinâmico.

Esta api implementa a possibilidade de enviar mensagens, escanear códigos de barras, reproduzir um vídeo, ou áudio, enviar um e-mail, uma mensagem, etc.

{Volver en GX al evento AddAppointment}



Até aqui programamos o código para o evento de usuário que incluímos. O que mais podemos fazer dentro de um evento executado no cliente?

FIN VIDEO 5