Objeto Web Panel

Los elementos de los web panels son:

Web Form: Cada web panel contiene un form Web, el cual debe ser diseñado por el analista agregándole variables, atributos, así como otros controles, para que el usuario pueda interactuar con el mismo.

Rules: Las reglas de un web panel permiten definir ciertos comportamientos puntuales de dicho objeto. Por ejemplo, declarar qué parámetros recibe.

Events: Los web panels emplean la programación orientada a eventos. Este tipo de programación permite definir código ocioso, que se activa en respuesta a ciertas acciones provocadas por el usuario o por el sistema. En esta sección de un web panel es donde se define el código ocioso asociado a los eventos que pueden ocurrir durante la ejecución del web panel. Aquí también pueden programarse subrutinas a ser invocadas por varios eventos.

Conditions: Es para definir las condiciones (globales) que deben cumplir los datos a ser recuperados (filtros). Análogo a la solapa de igual nombre del objeto Procedure.

Variables: aquí se declaran las variables que serán utilizadas dentro del objeto. Recordar que son locales.

Help: Permite la inclusión de texto de ayuda, que los usuarios podrán consultar en tiempo de ejecución del web panel.

Documentation: Permite la inclusión de texto técnico en formato tipo wiki, como documentación para el equipo involucrado en el desarrollo de la aplicación.

Propiedades: Son características a ser configuradas para definir ciertos detalles referentes al comportamiento general del web panel. Recordar que se acceden con F4.

En el ejemplo, el web panel contiene dos variables *&startDate* y *&endDate* como se muestra arriba. En tiempo de ejecución, el usuario podrá ingresar valores en las variables dado que en los web panels las variables son por defecto de entrada (salvo las incluidas en grids, como veremos).

En el **evento Enter** del web panel (asociado al botón **Billing Generation**), se obtienen para cada cliente las facturas cuyas fechas se encuentren en el rango especificado y se genera recibo para el cliente (invocando al Data Provider que devuelve la colección de Business Components que ya hemos visto oportunamente cuando estudiamos esos temas).

De modo que este **web panel** tiene como finalidad permitir al usuario ingresar un rango de fechas, y presionando el botón **Billing Generation**, ejecutar el procesamiento de las facturas.

El web panel mostrado arriba ha sido creado para exhibir los datos de un cliente. Se necesita invocarlo desde otro objeto (como ya veremos), pasándole por parámetro el código del cliente del cual se quiere mostrar la información.

Un web panel es un objeto típicamente utilizado para mostrar información. Entonces, cuando GeneXus encuentra atributos en el form, ¿qué intención adjudicará al programador que los puso allí? Pues, que le está pidiendo implícitamente que vaya a buscar los datos correspondientes a la base de datos para desplegarlos.

En definitiva, con el web panel anterior, GeneXus irá a la tabla CUSTOMER y filtrando por el atributo recibido por parámetro, *CustomerId*, mostrará la información del registro encontrado. ¿Qué información? La que reside en los atributos que figuran en el form. Pero el atributo *CountryName* no se encuentra en la tabla CUSTOMER. Pues sucede lo mismo que con un for each, un grupo de Data Providers, etc., es decir, desde el registro de la tabla base, se accede al registro relacionado de la tabla extendida que se necesite, para obtener el valor del atributo requerido (en nuestro caso accede a la tabla COUNTRY para recuperar el valor de *CountryName*).

Las inferencias que GeneXus realiza son las siguientes:

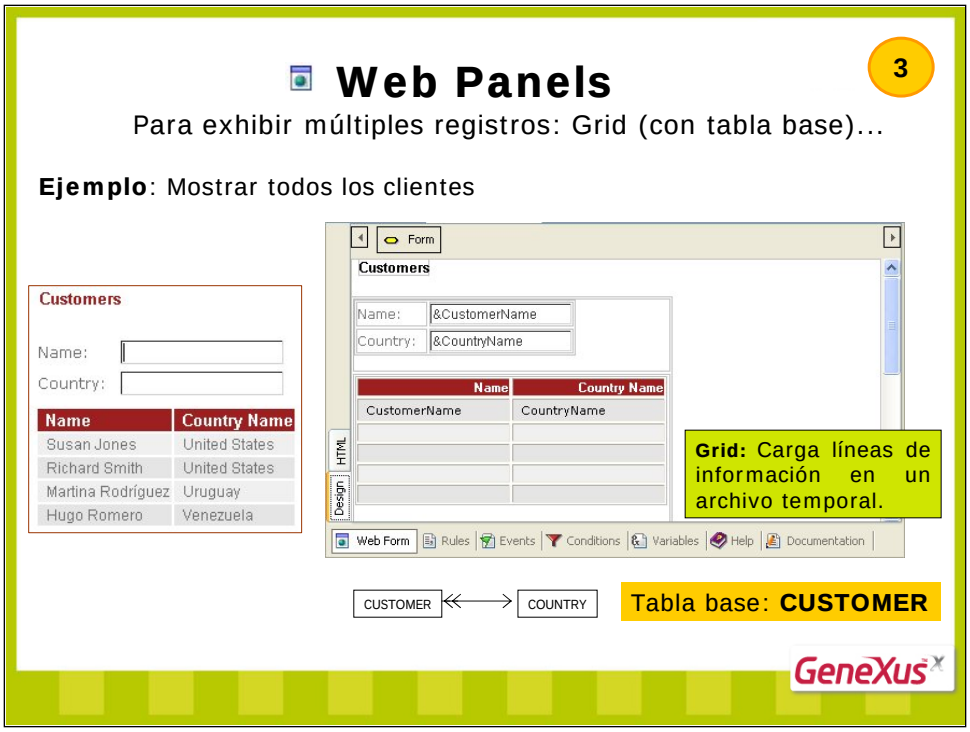
Al tratarse de un web panel, los atributos que figuren serán de consulta[1]. Entonces GeneXus deberá acceder a la base de datos para recuperar sus valores. ¿A qué tablas? Dependerá de si el Web panel tiene grids o no:

• Web panel **plano** (sin grid): están los atributos “sueltos” en el form, como en el caso que estamos mostrando. Si esto sucede, es porque el analista necesita acceder a información de **un registro** de una tabla (y eventualmente de los registros relacionados por tabla extendida), como es el caso del ejemplo. En este caso, el web panel estará bien programado si además se agrega un filtro que determine “ese” registro a mostrar. En nuestro caso, tenemos la regla parm que al recibir en atributo PK de la tabla sólo recuperará un registro. En definitiva, GeneXus determina una **tabla base** del web panel, así como lo hacía para un for each. ¿Cómo? Buscando la mínima tabla extendida que contenga a los atributos...

• Web panel **con uno o más grids**: lo veremos en lo que sigue, pero ya podemos intuirlo... ¿qué sentido tendrá colocar un grid? Mostrar información repetitiva. En el caso general: cada grid mostrará muchos registros de una tabla (y su información asociada).

---

[1] Al contrario de lo que sucede con los atributos en las transacciones (salvo los inferidos o los que tienen regla noaccept o propiedad Enabled deshabilitada).



Este web panel, a diferencia del anterior no es plano.

Cuando se incluye un grid en un form, se está indicando que se va a mostrar una cantidad indefinida de datos (en este caso, clientes).

Dado que en este web panel hay involucrados atributos, GeneXus infiere que se desea acceder a la base de datos. Simplemente diciéndole qué atributos se desean desplegar, sin necesidad de más información, ya sabrá qué hacer. Existe un grid, hay atributos, entonces, necesariamente habrá **tabla base**, esto es, la tabla de la base de datos que se navegará en busca de la información requerida.

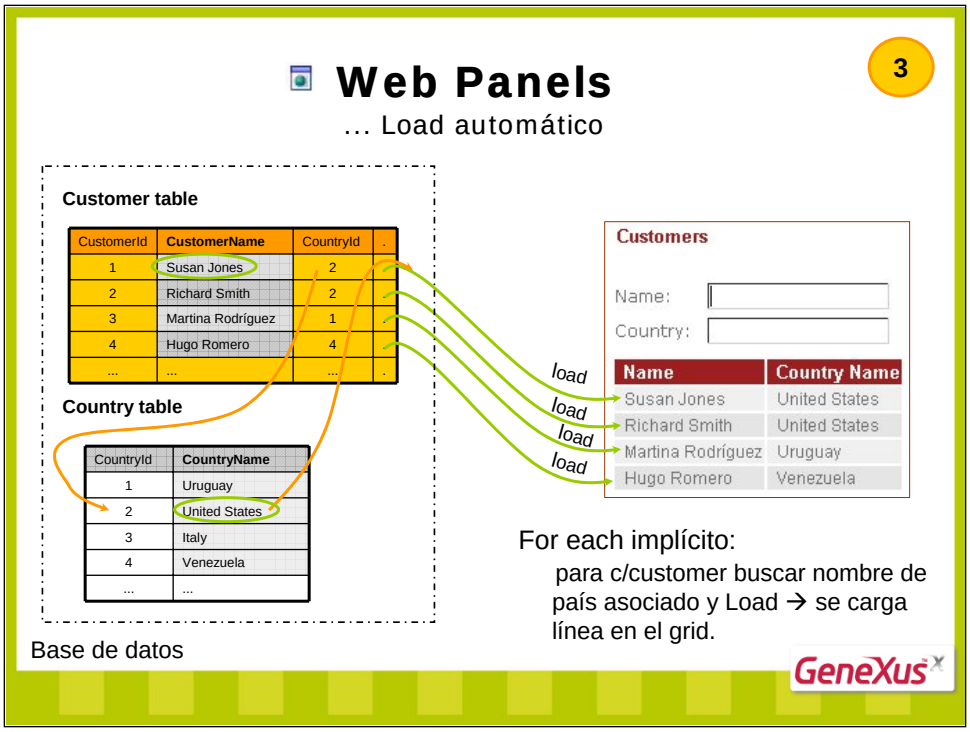
Si en lugar de mostrar esta información en pantalla, la quisiéramos en un listado, crearíamos un procedimiento con un print block 'customer' con los atributos *CustomerName* y *CountryName* y en el source programaríamos:

```
for each
    print customer
endfor
```

Esto es análogo, solo que el for each está implícito, no hay que especificarlo. Cada línea del grid que se carga, es como el print que se ejecuta en cada iteración del for each del listado.

De las tablas CUSTOMER y COUNTRY y de la relación entre ambas es Na1, GeneXus determina la **tabla base** de este web panel (aquella tal que su tabla extendida sea la mínima que cumple que contiene a todos los atributos referidos). Obsérvese que este ejemplo solo difiere del anterior, en que los atributos aparecen en un grid, y por tanto, no habrá que filtrar quedándose con un solo registro de CUSTOMER. En este web panel no hay regla parm que establezca filtro alguno.

También conseguiremos más adelante que este web panel no solamente se remita a mostrar los clientes de la base de datos en el grid, sino que además permita filtrar los clientes que se deseen ver en cada oportunidad, ya sea por nombre de cliente, por país del cliente, o por ambos. Por eso las variables que aparecen en el form.



Cuando GeneXus puede determinar automáticamente una tabla a recorrer para cargar las líneas del grid, lo hace, y en ese caso no habrá necesidad de brindarle esa información. Es por ello que decimos que en ese caso hay un for each 'implícito'. Luego veremos casos en los que esto no ocurre (grids sin tabla base).

Obsérvese en el ejemplo, que si en la tabla base existen 4 registros, se cargarán uno a uno los cuatro.

Si ahora queremos que el usuario pueda filtrar los clientes que desea ver... ahí entran en juego las variables que hemos definido en la parte fija del form, como veremos en la página siguiente.

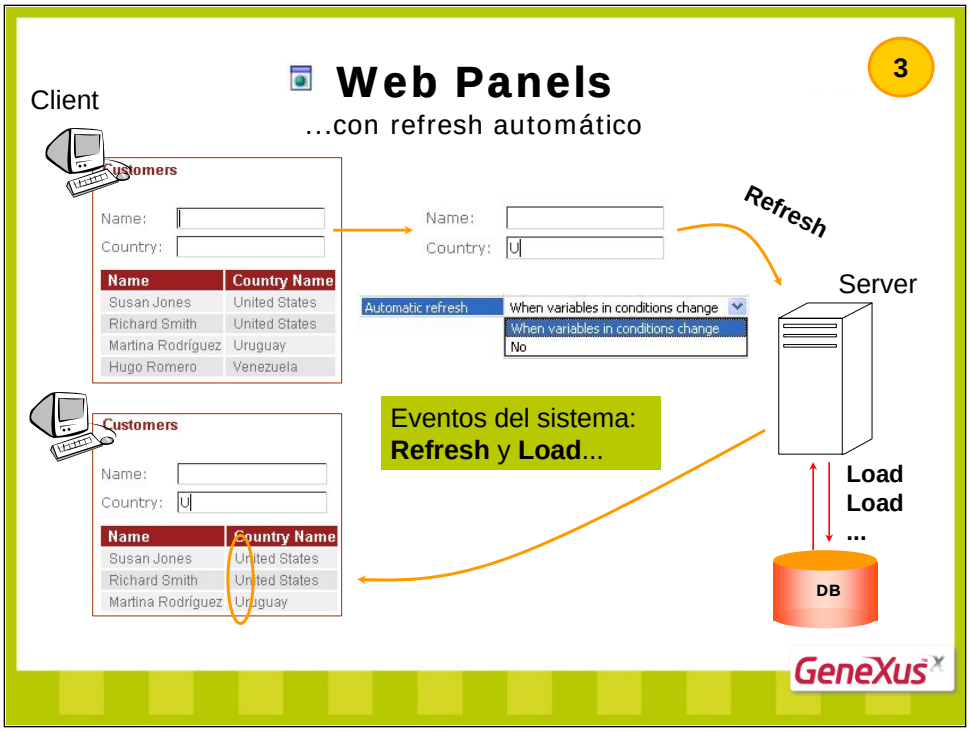
Obsérvese cómo en la ventana de propiedades del control Grid, aparece una de nombre 'Conditions'. Cliqueando en el combo se abrirá un editor para especificar las condiciones booleanas que deberán cumplir los registros para ser cargados como líneas del grid.

Es por esa razón que se han agregado las variables *&CustomerName* y *&CountryName* al form del web panel, de manera tal que el usuario pueda ingresar allí valores que operen como filtros sobre los datos a mostrar. En nuestro caso, hemos establecido filtros con el operador like. Las variables en la parte plana del form de web panels son por defecto de **entrada**. Luego veremos que en principio si están en grids serán por defecto de **salida**.

Obsérvese que las condiciones (separadas con ";") equivalen a las que aparecían en las cláusulas where de un **for each** (o **grupo** repetitivo de data provider).

De la misma manera, por cuestiones de optimización, puede determinarse, al igual que se hacía en un for each, criterio de ordenamiento de la tabla a recorrerse, como se muestra arriba en el ejemplo.

Nota: Asimismo, igual que en un procedimiento, no sólo pueden establecerse condiciones locales al grid (for each), sino también generales, mediante el selector Conditions. Esto tendrá sentido cuando se tenga más de un grid (for each), para no tener que repetir la misma condición cada vez.



La propiedad Automatic Refresh que se encuentra a nivel del Web Panel puede tomar los siguientes valores:

• **When variables in conditions change** (valor por defecto): luego de provocarse automáticamente el refresh, se dispara el evento Load cargándose el grid según los nuevos valores de las variables.

• **No**

Dependiendo del tipo de datos del filtro, y del control web utilizado para filtrar, la condición será aplicada cuando se está digitando o al abandonar el campo.

En el caso de filtros en controles edit, para tipo de datos Character, son aplicados cuando el usuario los va digitando. Para tipo de datos Date, DateTime y Numeric, las condiciones se evalúan cuando se abandona el campo.

En el caso de filtros combo boxes o dynamic combos, las condiciones son evaluadas cuando se abandona el campo. Para Check boxes y Radio buttons, las condiciones son evaluadas cuando el valor es cambiado.

Ejecución: ¿qué sucede en el cliente y en el servidor al tener la propiedad por defecto y un grid con tabla base?

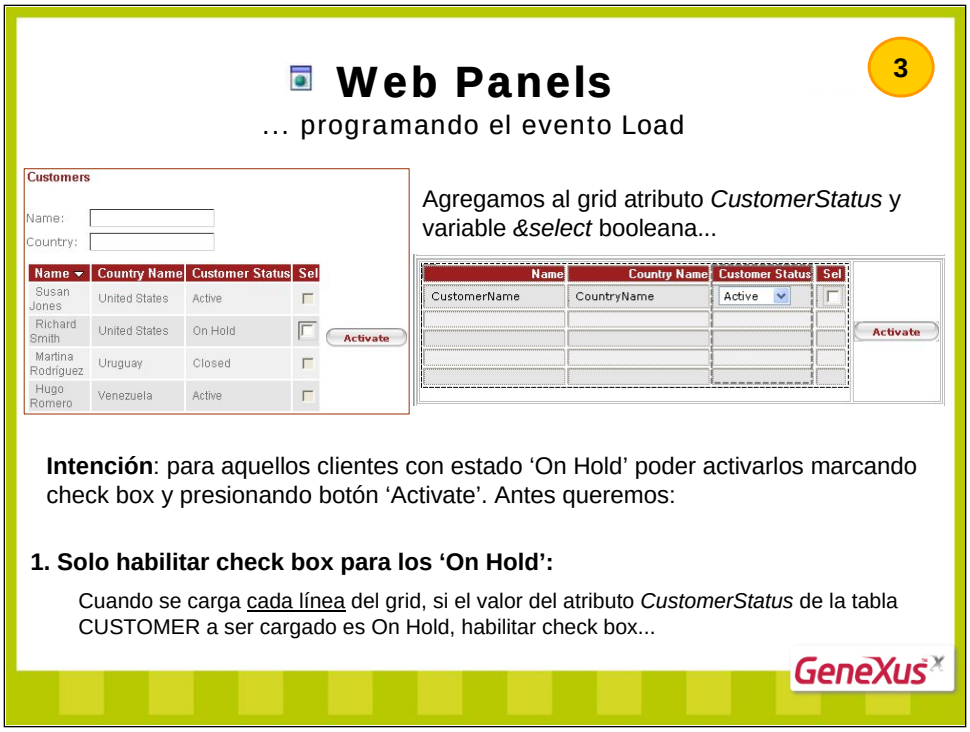
1ª ejecución (variables vacías):

For each CUSTOMER
guardar en memoria CustomerName
acceder a registro de COUNTRY relacionado
guardar en memoria CountryName
cargar (**load**) línea en el grid con ambos valores

N-ésima ejecución (cambia valor de variable de las conditions):

Automáticamente en el browser del cliente se detecta cambio → **refesh** en el servidor para volver a cargar el grid con los registros que cumplan las condiciones (con los nuevos valores de las variables) → **load** por cada línea.

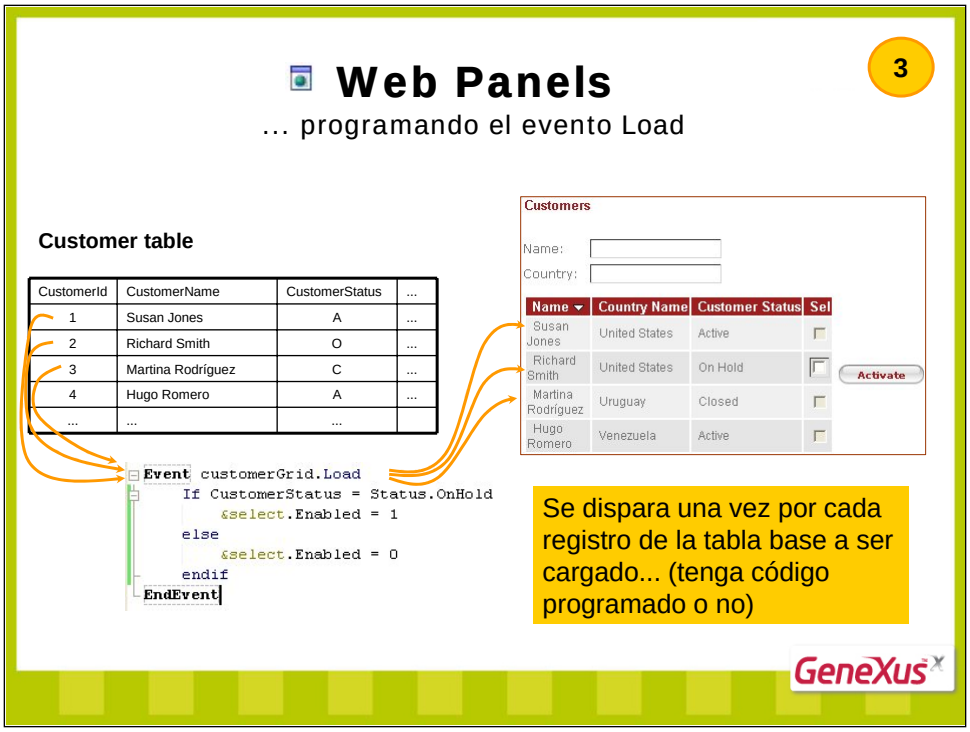
Conclusión: Existen dos eventos del sistema que ocurren en el momento de carga del form (Refresh) y de carga de cada línea del grid (Load). Como veremos, a estos eventos puede asociárseles código para que se ejecute en esos momentos específicos...



Ampliaremos la funcionalidad de nuestro web panel, permitiendo ver el estado de cada cliente, y para aquellos que estén 'On Hold', brindando la posibilidad de pasarlos al estado 'Active'.

Para hacer esto, agregaremos al grid el atributo *CustomerStatus* que muestra el estado, y una variable booleana *&Select*, que permitirá al usuario seleccionar aquellos clientes que desea 'activar'. Además un botón para efectivamente activar todos los clientes marcados. Pero esto lo haremos en un segundo paso.

Obsérvese que al ser el tipo de datos de la variable &select booleano, por defecto aparece en el grid como un check box. Para asegurar que el usuario solamente intente 'activar' clientes 'On Hold', deseamos que solamente aparezca habilitado este check box cuando corresponde... para ello necesitaremos programar la carga de cada línea, esto es, el evento Load...



En la sección **Events** del web panel, programamos el evento Load del grid que vemos arriba. En el ejemplo, customerGrid es el nombre que hemos dado al control grid en el form.

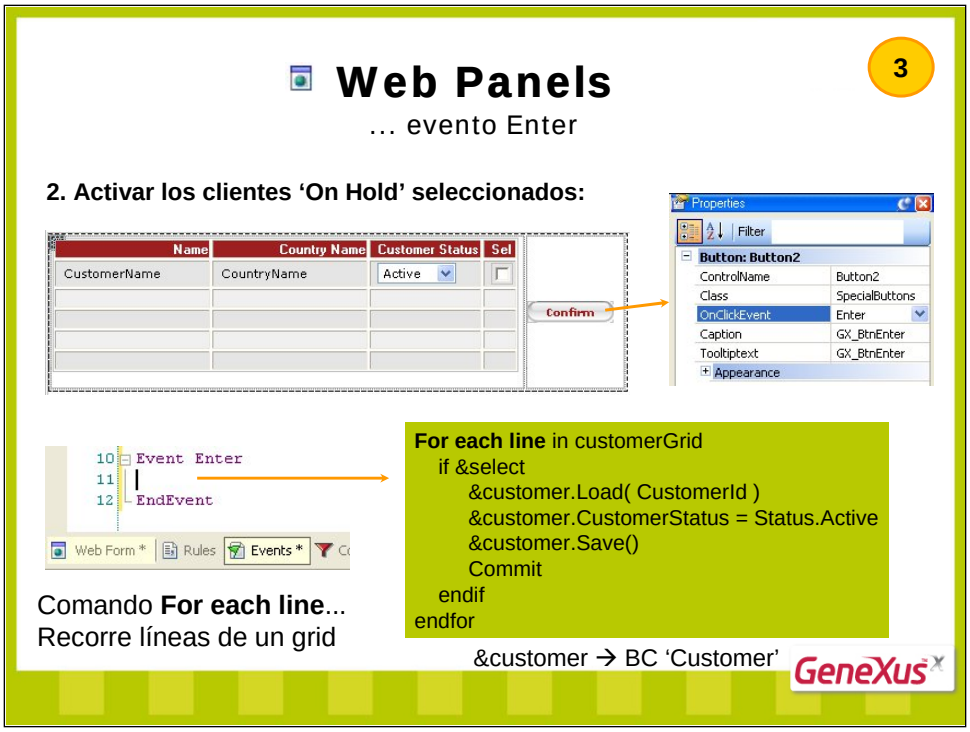
Al tratarse de un web panel con un único grid, también podríamos haber programado el evento Load a secas:

```
Event Load
    if CustomerStatus = Status.OnHold
        &select.Enabled = 1
    else
        &select.Enabled = 0
    endif
endevent
```

Es decir, en el caso de un web panel con un único grid, no es necesario calificar el evento con el nombre del grid. Igualmente recomendamos hacerlo, anticipándonos a la posibilidad futura de agregar otro grid al form del web panel.

El evento Load se disparará por cada línea que haya de cargarse en el grid, se encuentre éste programado o no. Lo que hacemos en el ejemplo es aprovechar ese momento inmediatamente anterior a la carga, para efectuar una acción. Y ese es el código que incluimos en el evento.

Ahora sí, estamos en condiciones de implementar la 'activación', para lo que necesitaremos asociar al botón un evento, que podrá ser el evento Enter o uno de usuario...



En el ejemplo, cuando el usuario presiona el botón 'Confirm', necesitamos recorrer todas las líneas del grid, y para cada una de ellas, si el check box de la variable booleana *&Select* fue marcado por el usuario, entonces debemos cambiar el estado del cliente correspondiente, de 'On Hold' a 'Active'.

Al insertar el botón en el form del web panel, y editar sus propiedades, podrá observarse que por defecto, el botón está asociado al **evento Enter** (ver propiedad **OnClickEvent**).

El Enter será un evento del sistema, que se ejecuta tanto cuando el usuario hace clic sobre el control asociado, como cuando presiona la tecla Enter.

En este ejemplo veremos a la vez:

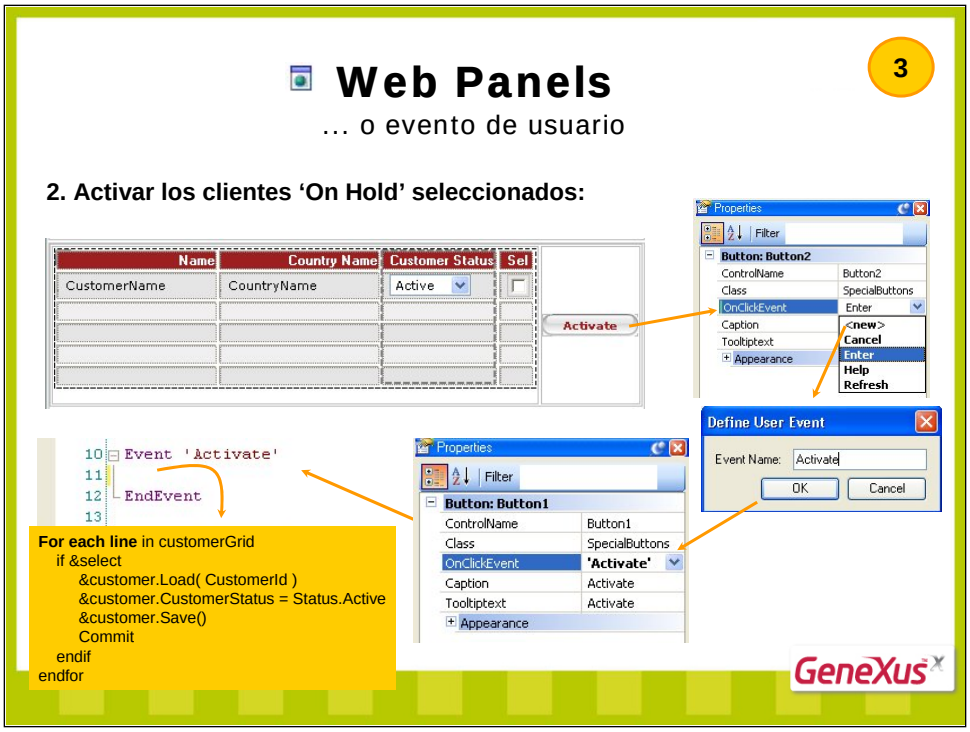
• Posibilidad de definir eventos de usuario y asociárselos a controles o de utilizar el evento Enter del sistema.

• Comando **For each line**, para recorrer las líneas ya cargadas en un grid.

• Variables en un grid pasan de ser Read only (de salida) por defecto, a ser de entrada, cuando se utiliza comando for each line en ese grid (también cuando se programan eventos OnClickEvent, Click, etc., sobre alguna columna del grid).

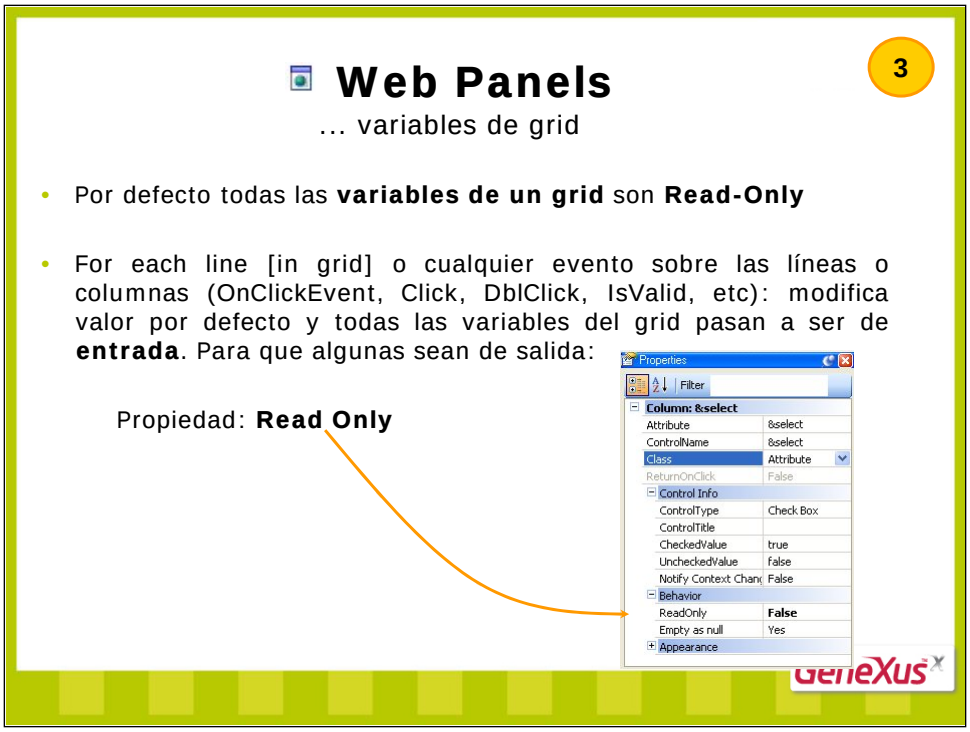
Obsérvese que el comando **For each line** sólo tiene en común con el comando **For each** estudiado antes, el hecho de representar una estructura repetitiva. La diferencia más importante: mientras el for each recorre registros de una tabla (**base**) de la base de datos, el **for each line** recorre las líneas de un grid.

En nuestro ejemplo hemos definido una variable &customer, Business Component Customer, mediante la cuál cambiaremos a estado 'Active' todas las líneas del grid marcadas por el usuario. Para ello recorremos el grid con el comando for each line.



Otra posibilidad, en lugar de utilizar el evento del sistema Enter, es definir un evento de usuario. Ello se consigue siguiendo los pasos que pueden verse arriba.

Como puede apreciarse, la mayoría de los controles presentes en un form tienen la propiedad **OnClickEvent** asociada. Esa propiedad permite especificar un evento a dispararse cuando el usuario haga clic sobre el control. Podrá ser un evento del sistema (Refresh, Enter) o uno definido por el usuario.



No así las de la parte fija del web panel. Esas son por defecto de entrada, como ya hemos visto antes para las variables que utilizamos para filtrar los customers mostrados en el grid.

Cómo desplegar datos en un grid

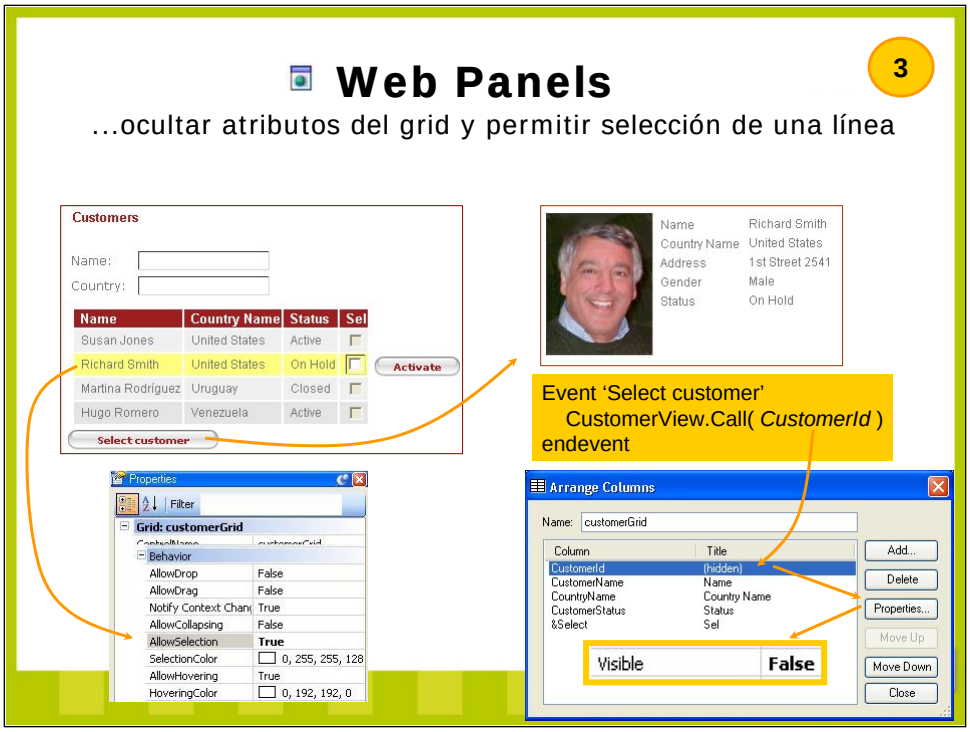
Por defecto todo atributo y variable que está dentro de un grid se despliega en ejecución como texto, es decir que es únicamente de lectura y por consiguiente no puede ser modificado.

Cómo aceptar datos en un grid

Es posible aceptar datos en las variables de un grid dependiendo de la programación de los eventos existentes en el objeto:

1. Si dentro de un evento del web panel se está utilizando el comando For each line, **todas** las **variables** que están dentro del grid pasan a ser de **entrada**. Es posible indicar en este caso cuáles son las variables que no van a poder ser modificadas a través de la propiedad ReadOnly.

2. Si dentro de la fila hay algún control con un evento click, dblClick, etc.. asociado (ó evento de usuario especificado en la propiedad OnClickEvent), sucederá lo mismo.



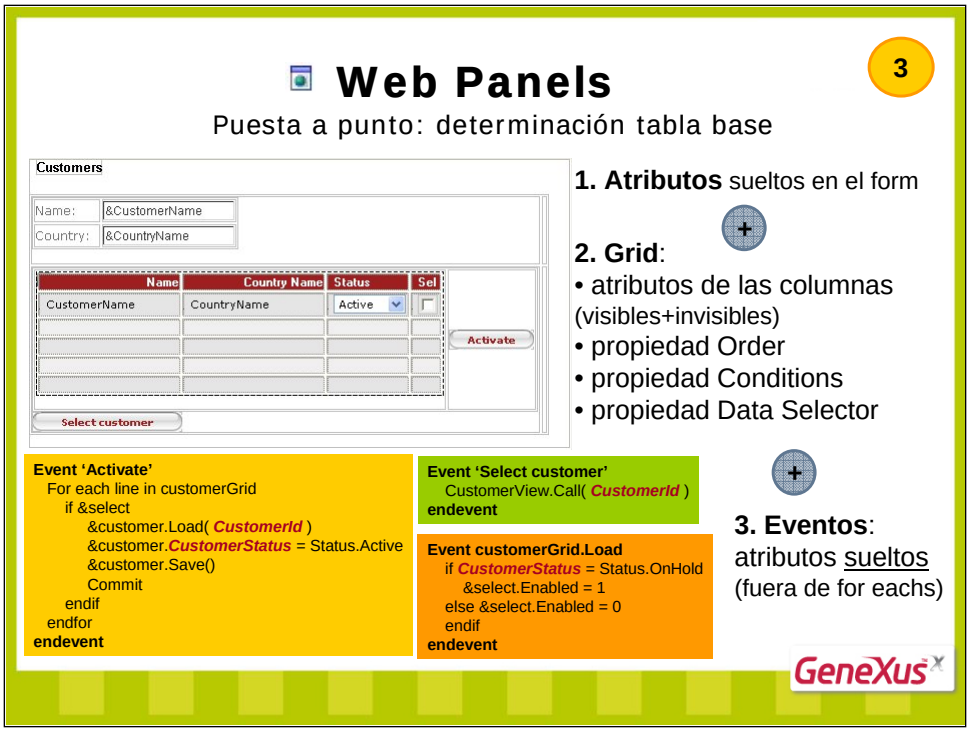
Seguimos ampliando nuestro ejemplo; ahora queremos que el usuario pueda seleccionar una línea del grid (un cliente) y presionando botón 'Select customer' poder invocar al web panel que habíamos implementado antes.

En este ejemplo vemos dos funcionalidades en juego:

- La necesidad de colocar una columna en el grid oculta (no visible).
- Permitir que el usuario seleccione una línea del grid para hacer algo con ella.

¿Por qué colocar la columna correspondiente a *CustomerId* y ocultarla, en lugar de no agregarla en absoluto? Reflexione sobre lo siguiente: el atributo *CustomerId* enviado por parámetro al ejecutarse el evento 'Select customer', ¿de dónde es extraído? ¿De la base de datos? No, es el que está cargado en el grid. Más específicamente, por cada grid existirá un archivo temporal que contendrá tantas columnas como las del grid, visibles y ocultas. Cuando el usuario selecciona en el grid la segunda línea, y presiona el botón 'Select customer', se está posicionado en ese archivo temporal, correspondiente a esa línea (¡y nunca en la base de datos!). Es por esta razón que de no colocar la columna correspondiente a *CustomerId* en el grid, no estaríamos pasando valor alguno a CustomerView.

Para permitir al usuario seleccionar una línea del grid, alcanza con 'prender' la propiedad 'AllowSelection' del grid.

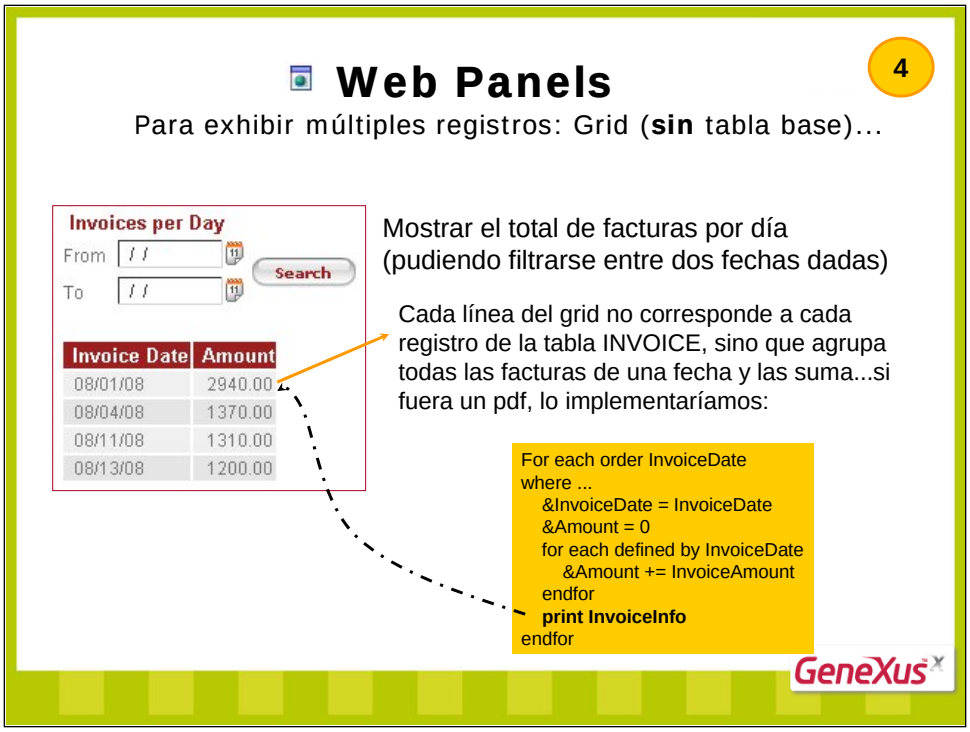


¿Cómo determina GeneXus una tabla base a recorrer automáticamente para cargar el grid del web panel?

De existir algún atributo en por lo menos uno de los 3 lugares mencionados, GeneXus podrá encontrar tabla base. Para determinarla extrae los atributos encontrados allí (parte fija del Form, en el Grid, tanto en sus columnas como en las propiedades Order, Conditions o using Data Selector, y en los eventos programados en el selector de Eventos, solamente considerando los atributos que estén 'sueltos' dentro del evento, es decir, no dentro de un comando for each de acceso a la base de datos), y determina la mínima tabla extendida que los contiene. La tabla base de esa extendida, será la recorrida automáticamente y cargada con el Load.

Observemos que solamente hemos programado el evento Load para tomar la decisión para cada línea a ser cargada en el grid, acerca de si se habilitará o no para la misma la variable &select que permitirá al usuario marcar el check box.

Importante: La forma de determinación de la tabla base de un grid dependerá de si existe otro grid en el web panel o es el único. El resumen presentado aquí corresponde a un único grid. En este caso se dice que el propio **Web panel tiene tabla base**. Es la del grid. Cuando existan más de un grid en el web panel, esto ya no tendrá sentido, y cada grid pasará a tener o no tabla base. Veremos la forma de determinación de las tablas bases en ese caso, cuando tratemos el caso de múltiples grids en un web panel.



Aquí presentamos otro ejemplo, para observar un caso en el que surge naturalmente la necesidad de implementar un **grid sin tabla base**.

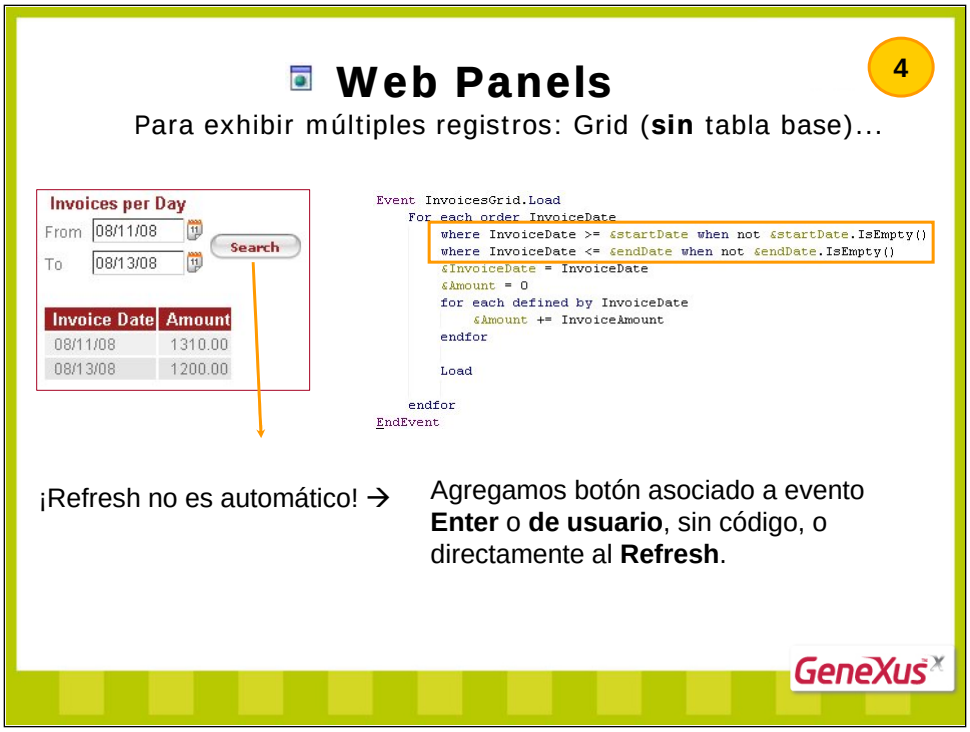
El caso natural de implementación de un grid con tabla base, es cuando se quiere cargar por cada registro de una tabla, una línea del grid (1 registro – 1 línea). Cuando el caso es que se quiere cargar una línea del grid como producto de recorrer varios registros de la base de datos (N registros – 1 línea), como productos de cálculos, etc., suele ser más natural implementar el grid sin tabla base.

Este es el caso del ejemplo: no queremos cargar por cada factura una línea, sino que queremos agrupar facturas por fecha, y por cada grupo, cargar una línea que sumariza sus Amount. Estamos hablando de un corte de control. Si tuviéramos que implementar un listado PDF en lugar de un web panel, sabríamos bien cómo programarlo (como lo hacemos arriba). Veamos cómo implementarlo con un web panel...

El objetivo del comando **LOAD** es agregar una línea en un grid. Es necesario cuando el grid no tiene tabla base, dado a que en ese caso no se agregará automáticamente línea alguna. El evento Load ocurrirá una vez, y si éste no se programa, el grid resultará vacío. Por tanto en este caso es indispensable programarlo, de acuerdo a la lógica que corresponda. Una vez que se haya asignado valor a cada variable, y se desee agregar una línea al grid, deberá ejecutarse el comando **LOAD**.

Solamente se puede especificar el comando **LOAD** dentro del evento **Load** del grid de un web panel.

Obsérvese que este caso es el que deja más a cargo del analista la implementación. Frente al caso de grid con tabla base, en este GeneXus realiza muchas menos inferencias.



En el caso de un grid sin tabla base, los filtros sobre los datos a mostrar son programados dentro del código que implementa la carga (el del Load). Aquí no habrá refresh automático. Esto es, cuando el usuario modifique los valores de las variables que intervienen en los filtros, GeneXus no detectará que debe volver a cargar el grid.

Para ello deberá, por ejemplo, agregarse un botón asociado al evento Enter, o a un evento de usuario, sin código, debido a que solo lo necesitamos para que se produzca un Refresh, es decir para que se vuelva al servidor a cargar el web panel.

Sobre los eventos disponibles y el orden en que se disparan, entraremos en lo que sigue.

En todo Web panel existen eventos del sistema que pueden programarse. Algunos ocurrirán siempre, en cada ejecución del web panel (como el Start, Refresh, Load), otros si se los declara y el usuario realiza las acciones necesarias para provocarlos (Enter, definidos por el usuario, TrackContext).

Asimismo, casi todos los controles que aparecen en el form brindan la posibilidad de disparar un evento cuando el usuario hace clic con el mouse sobre ellos (aparecen como hipervínculos en ejecución); se consigue de dos maneras distintas:

1. Editando las propiedades del control (F4), y definiendo un evento de usuario en la propiedad **OnClickEvent**, o asociándole el evento **Enter** o el **Refresh**.

2. Dándole un nombre al control y en la sección de Eventos programando:

```
Event nombreControl.click
   …
Endevent
```

Con esta última alternativa no tendremos que definir un evento de usuario, sino que estaremos programando el **evento click** del control. Lo mismo ocurre con los eventos DblClick, RightClick, IsValid... (para cuando se hace doble clic, botón derecho, etc.).

Sobre los eventos asociados a las acciones sobre los controles (click, dblclick, drag, drop, etc.) no nos explayaremos en el presente curso. En nuestro wiki encontrará información detallada, así como en el Help de GeneXus.

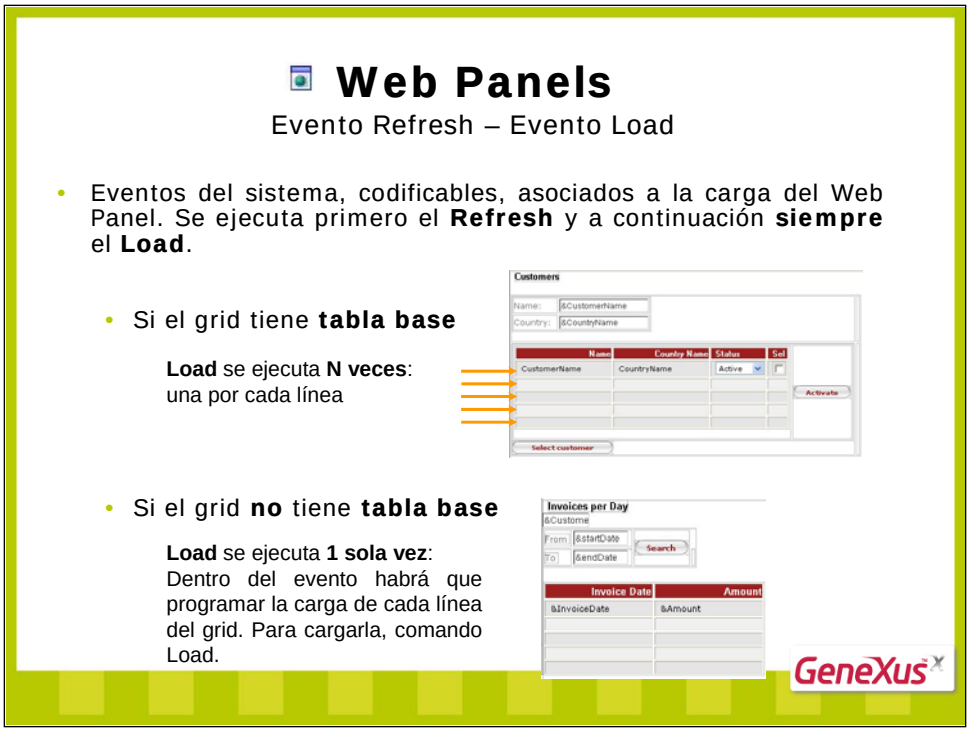
Sobre el evento TrackContext solo mencionaremos que será posible detectar cambios en el valor dado a un control (grid, variable, etc.) y en ese caso disparar este evento para en base al valor modificado, tomar una acción.

Web Panels

Evento Start

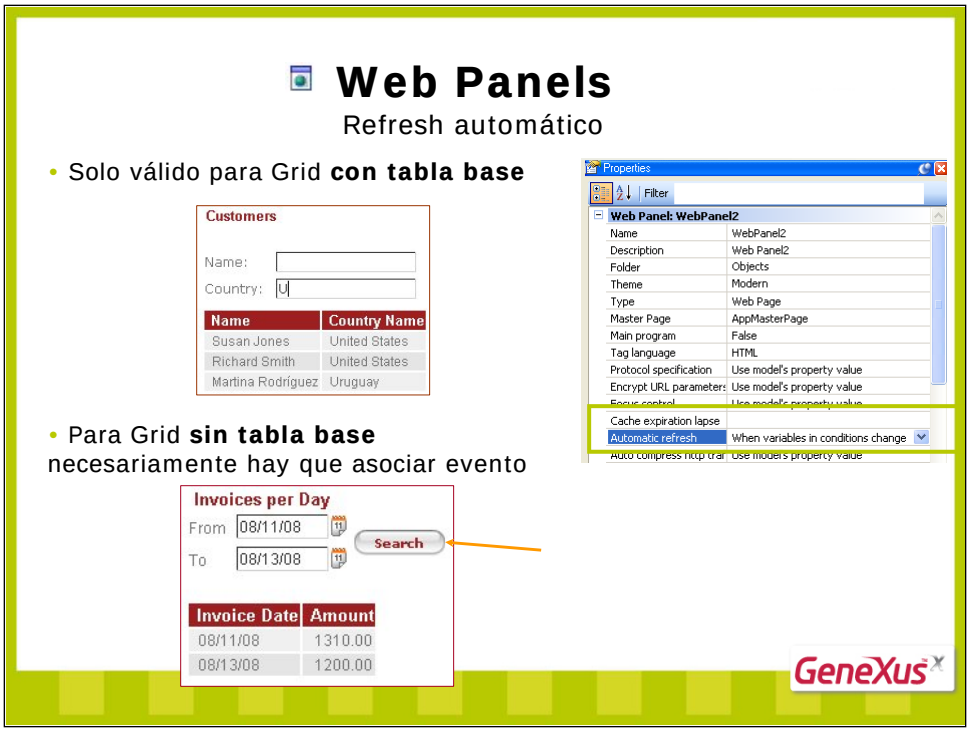
- Es un evento del sistema, que ocurre automáticamente siempre que se hace Get o Post y es el primer evento que se ejecuta.
- No se conocen valores de atributos, salvo los recibidos por parámetro. Esto se debe a que aún no se ha efectuado la consulta.
- Ejemplo: se puede utilizar para que un control del form no aparezca visible, para cargar un bitmap, para asociarle un Link a otro control, etc.:

```
Event Start
    &var.Visible = 0
    &Update = LoadBitmap("images/edit.gif")
    newControl.Link = Link(TCustomer)
endevent
```



Cuando el web panel es con tabla base, al producirse el evento Refresh se accede a la base de datos, a esa tabla base (la asociada al web panel), y se la recorre cargando los registros que cumplan las condiciones (conditions del grid y generales). Ocurrirá en ese proceso **un evento Load por cada registro en el que se esté posicionado**, inmediatamente antes de cargarlo. Esto nos permite realizar alguna operación que requiera de ese registro (y de su extendida), antes de efectivamente cargarlo en el grid. Inmediatamente luego de ejecutado el código asociado al evento Load, se cargará la línea en el grid y se pasará el puntero al siguiente registro de la tabla base, para realizar lo mismo (evento Load, carga de la línea). Este proceso se repetirá hasta cargar todas las líneas del grid.

Si un web panel es sin tabla base, GeneXus no puede determinar automáticamente una tabla de la base de datos a recorrer para mostrar la información que se presenta en el form. En este caso en el form solamente aparecen variables (y no atributos) y también ocurrirán los eventos Refresh y Load, sólo que el **evento Load se ejecutará una única vez**, dado que no se estará posicionado en ningún registro de ninguna tabla. Dentro de ese evento habrá que codificar la carga, que podrá requerir acceder a la base de datos (ej: comando for each) o no (supóngase que se desea cargar el grid con información obtenida de recorrer un SDT collection, tras efectuar alguna transformación sobre sus items... o cargar líneas en el grid producto de cálculos). El control de la carga del grid, queda aquí en manos del analista, utilizando el **comando** Load. Este comando solo es válido dentro del evento de igual nombre. Obsérvese cómo en el caso de grid con tabla base, este comando se torna innecesario.



Estudiemos con más detalle la propiedad **Automatic Refresh** que se encuentra a nivel del Web Panel puede tomar los siguientes valores:

• **When variables in conditions change** (valor por defecto): luego de provocarse automáticamente el refresh, se dispara el evento Load cargándose el grid según los nuevos valores de las variables.

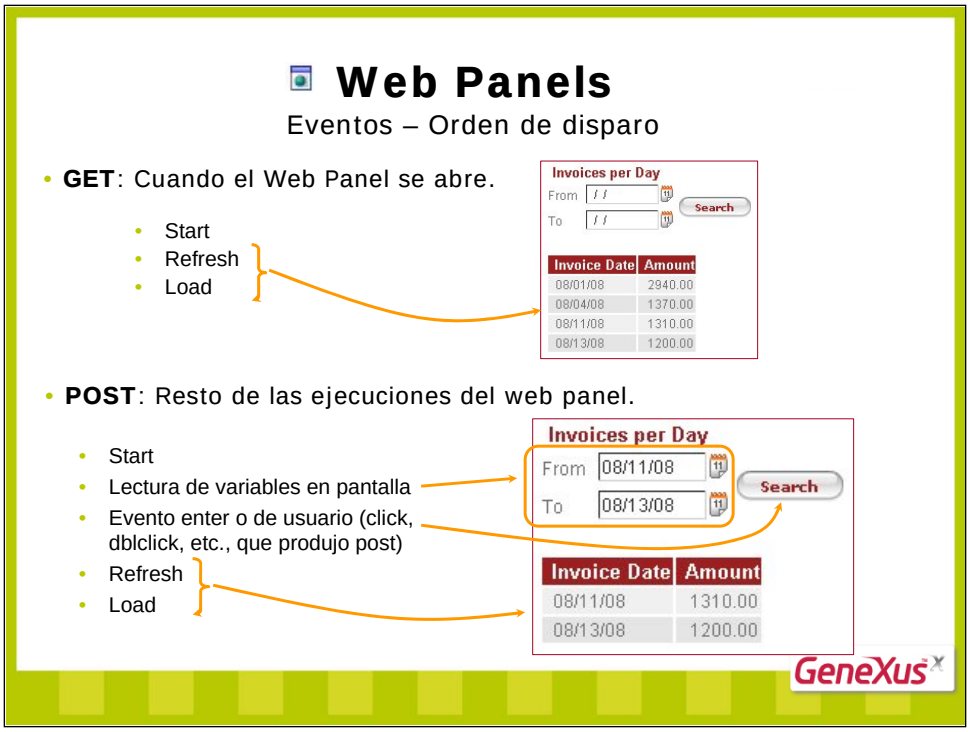
• **No**: para que el contenido del grid se refresque luego de cambiar los filtros, el usuario debe realizar una acción:

Si el grid es <u>con</u> tabla base:

- Los cambios en las variables de los filtros se detectan automáticamente.
- Al presionar la **tecla Enter**, se dispara el Refresh de la página (no el código del evento Enter, aunque esté programado).
- Al hacer click en un **botón o en una imagen** asociados a un **evento de usuario**, inmediatamente después de cambiar el valor de una variable de las conditions del grid, se ejecuta este evento e inmediatamente a continuación el Refresh de la página. Si el evento fuera el Enter, este evento NO se ejecuta.

Si el grid es <u>sin</u> tabla base:

- Los cambios en las variables de los filtros NO se detectan automáticamente
- El usuario debe hacer clic en un **botón o en una imagen** asociados a un **evento Refresh** o a un **evento de usuario que invoque a un Refresh**.



Los eventos que se disparan y su orden depende de si se está abriendo el web panel (Get) o si ya estaba abierto y se está efectuando una acción posterior, como presionar un botón (Post).

Arriba mostramos con ejemplos el caso general.

1a. vez: Start + Refresh + Load

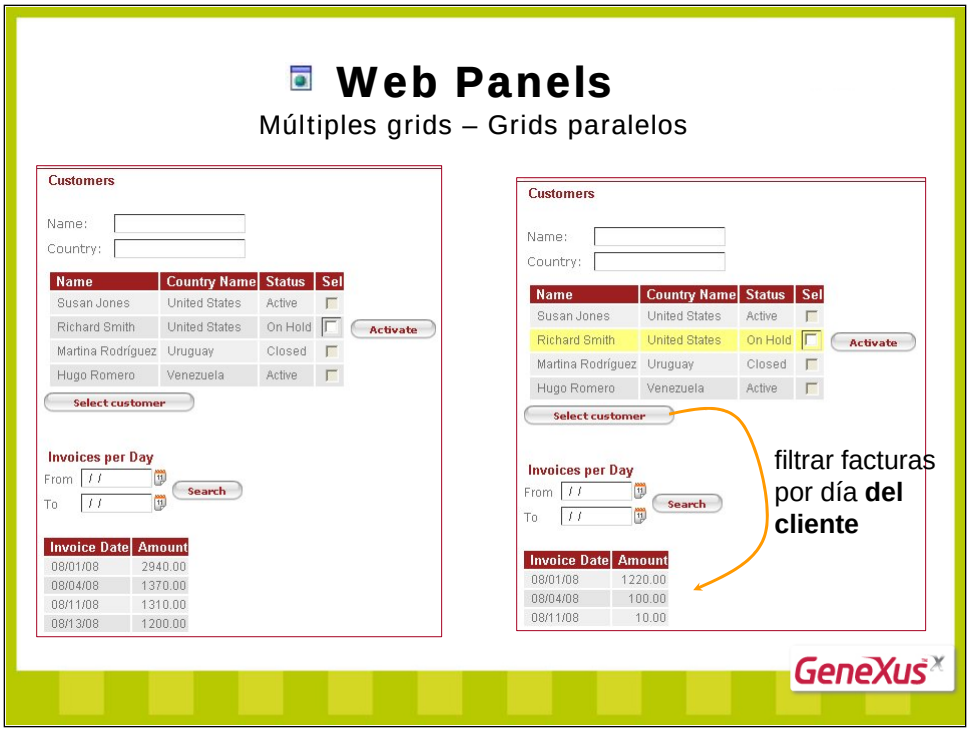
N-ésima vez: Start + Lectura de variables de pantalla + Evento que produjo el Post + Refresh + Load.

Este es el caso general... existe una excepción...

Internamente GeneXus determina las entradas y salidas de cada evento. Si en sus entradas, se requiere de acciones ejecutadas en el Server, entonces el evento se ejecutará en el Server. Por ejemplo, si entre las entradas de un evento de usuario se encuentra alguna de las salidas del evento Start (del server), entonces el evento de usuario se ejecutará en el Server.

Si el código del evento no requiere que se ejecute en el servidor, entonces por performance, se ejecutará en el cliente, como código javascript.

De todas maneras el analista GeneXus no deberá preocuparse de estos asuntos, dado que en todo caso será GeneXus quien tendrá la inteligencia de resolver dónde ejecutar el evento.



Aquí presentamos un ejemplo que reúne los dos casos que veníamos estudiando: el web panel mostrado en ejecución tiene dos grids paralelos: uno que muestra información de los clientes del sistema, y otro que muestra facturas, totalizadas por día.

En nuestro caso, querremos además relacionar los datos, de manera tal que si el usuario selecciona un cliente, se le muestren solo las facturas de ese cliente. Incluso al establecer filtros de fechas, también querremos que valgan para el cliente seleccionado (y no para todos los clientes).

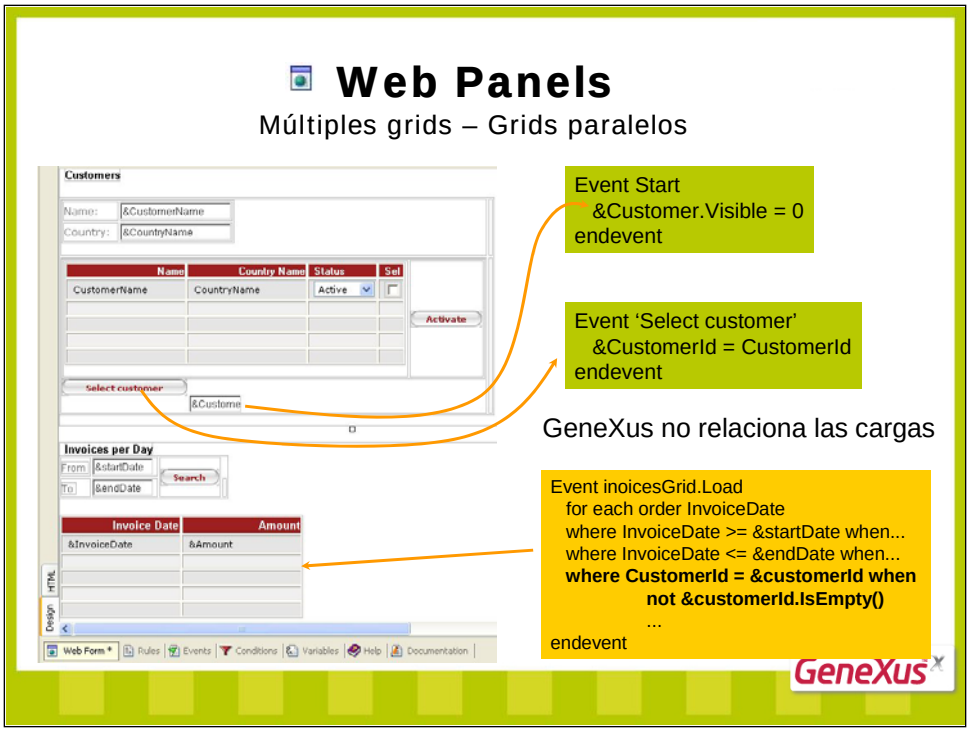
Cuando un web panel contiene más de un grid en su form, **GeneXus no determina una única tabla base asociada al web panel, sino una tabla base asociada a cada grid**.

Atributos que participan en la **determinación de la tabla base de cada grid**:

- Los incluidos en el grid (se tienen en cuenta tanto los atributos visibles como los no visibles)
- Los referenciados en Order y Conditions locales al grid

A diferencia de lo que sucedía para un web panel con un solo grid, en el caso de múltiples grids los atributos de la parte fija del web panel no participan en la determinación de la tabla base de ninguno de ellos, pero deberán pertenecer a la tabla extendida de alguno (para que sea posible inferir sus valores). De no respetarse esto, al especificar al web panel, se mostrará en el listado de navegación resultante, una advertencia informando de esta situación.

Los atributos utilizados en los eventos del web panel tampoco participan en la determinación de la tabla base de ninguno de los grids. Los atributos que se incluyan en los eventos fuera de comandos for each, deberán pertenecer a la tabla extendida de alguno de los grids (al igual que los de la parte fija).



Este web panel podría haberse implementado de varias maneras distintas, dando por resultado la misma ejecución.

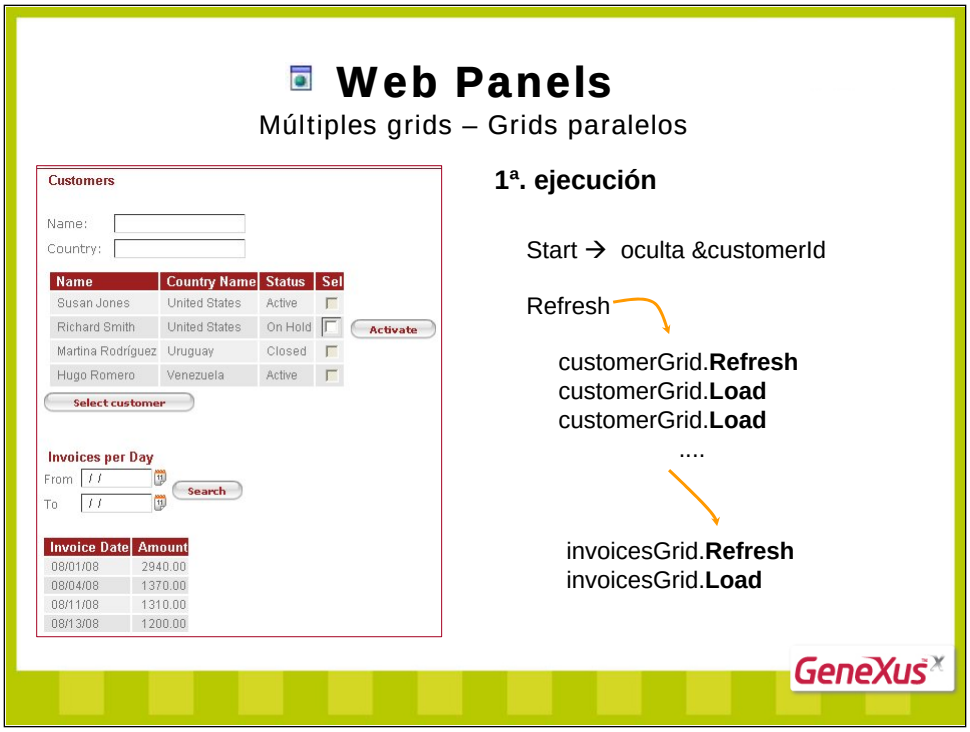
La implementación más natural es la que podemos ver arriba: el primer grid tiene tabla base y el segundo no. Pero podría haberse implementado al revés, con variables en el primer grid y teniendo que realizar la carga de los clientes a mano en el Load, y atributos en el segundo grid, y algunas cosas más para lograr el corte de control, siendo un grid con tabla base. O cualquiera de las otras dos combinaciones (ambos grids con tabla base, o ninguno con tabla base).

Lo importante es, una vez elegida la implementación más natural al caso, realizarla correctamente.

En cualquiera de los casos, aunque la información a cargar en un par de grids se encuentre relacionada en la base de datos, GeneXus no asumirá ninguna relación entre los datos a la hora de cargar un grid y el otro. **Es análogo al caso de un par de for eachs paralelos en el Source de un procedimiento**.

Como puede verse arriba, al web panel en el que teníamos el grid con los clientes, le hemos agregado la parte de visualización de facturas por fecha que habíamos implementado en web panel aparte. Pero no alcanza con simplemente unir ambos web panels... para poder relacionar las cargas de los grids, deberemos agregar cierta lógica. En nuestro caso deseamos que una vez que se seleccione un cliente del primer grid, las facturas que se carguen en el segundo no sean las de todos los clientes, sino las del seleccionado.

Para ello debimos hacer dos cosas: agregar una variable *&CustomerId* para almacenar el id del cliente seleccionado al presionar 'Select customer', y luego agregar un filtro por el valor de esa variable cuando se carga el grid de Invoices. Asimismo tuvimos necesariamente que colocar esa variable en el form para que todo funcione como esperamos... y la razón la encontraremos en el análisis que sigue.



Analicemos lo que sucede cuando se ejecuta este web panel por primera vez.

Primero se ejecuta el evento Start, que en nuestro caso oculta la variable *&CustomerId*.

Luego, como puede apreciarse en la imagen, se produce un evento Refresh genérico, luego del cuál se producirán las cargas de todos y cada uno de los grids que se encuentren en el web panel, de izquierda a derecha de arriba a abajo.

Por cada uno ocurrirá un evento Refresh propio y el evento Load (N veces si el grid tiene tabla base, 1 sola si no la tiene).

Obsérvese que en nuestro caso, el grid de clientes tiene condiciones para cargarse, por las variables de filtro *&customerName* y *&countryName*, pero al estar vacías no aplican (dado que ambas cláusulas condicionales tienen **when not &var.IsEmpty()**)

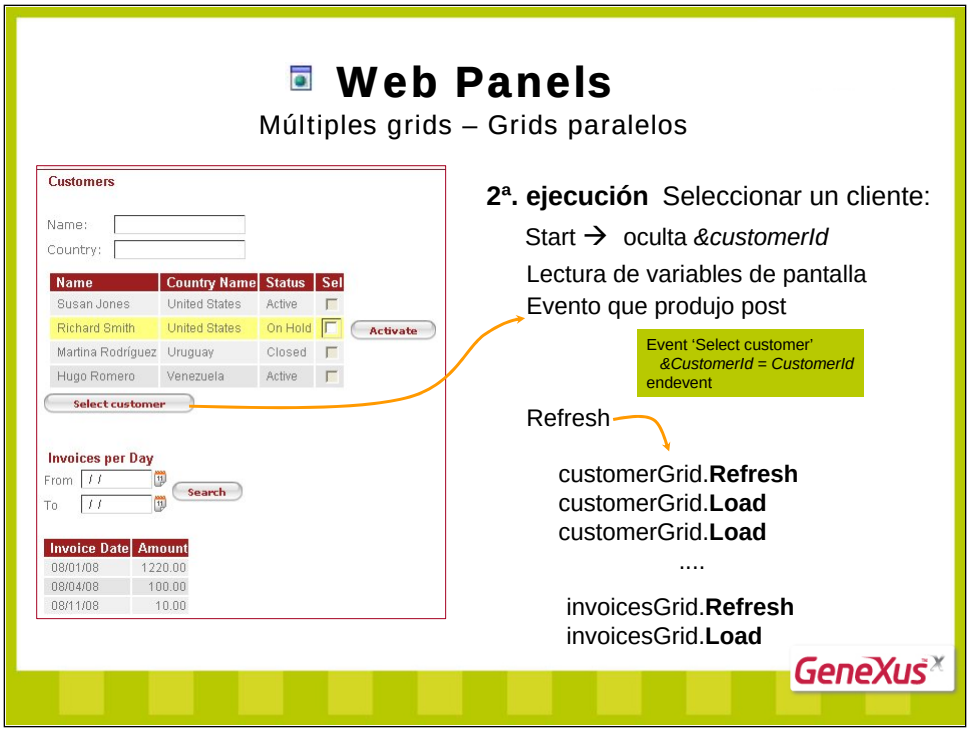
El segundo grid es sin tabla base, pero como ya vimos, el evento Load ejecutaba un for each con cláusulas where, por tanto se cargarán solo aquellas líneas para las que se cumplan sus condiciones. Estas son tres:

where InvoiceDate >= &startDate when not &startDate.IsEmpty()

where InvoiceDate <= &endDate when not &endDate.IsEmpty()

where CustomerId = &customerId when not &customerId.IsEmpty()

Obsérvese que en esta primera ejecución *&customerId* estará vacío, por lo que no se aplicará este filtro y se cargarán todas las facturas por día, de todos los clientes.



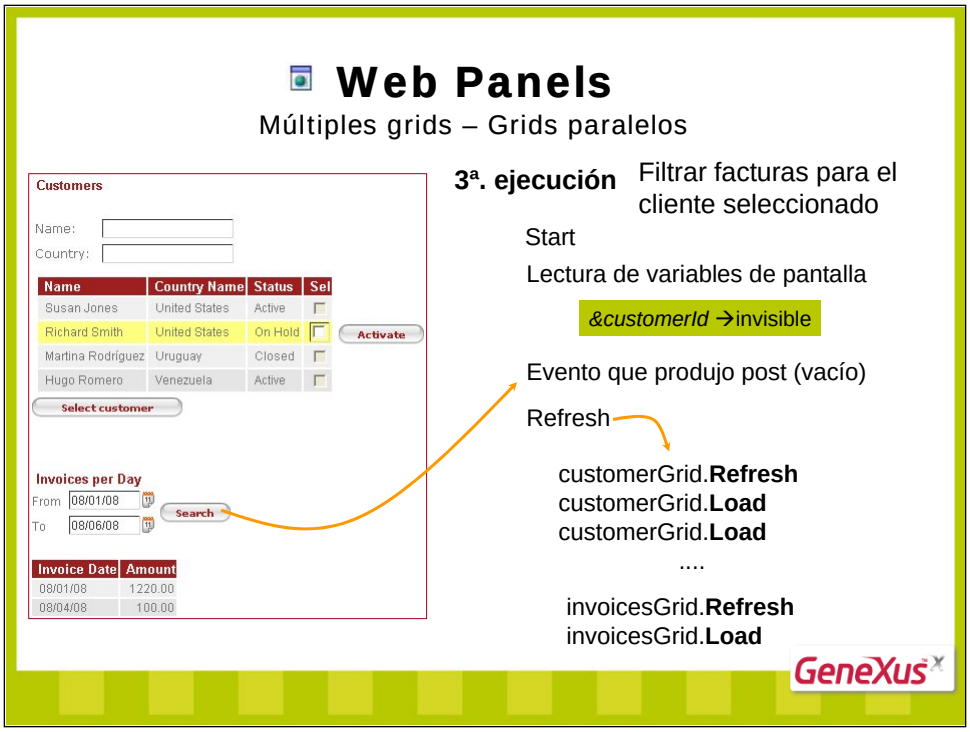
Luego el usuario selecciona del grid el cliente correspondiente a la segunda línea y presiona el botón 'Select customer'.

Se detecta una acción, y se realiza un post al servidor, quien ejecuta el Start (ocultando la variable), luego lee las variables en pantalla (*&customerId* por ahora está vacía, pero no sólo son consideradas variables de pantalla las variables definidas, sino, por ejemplo, la información completa de la línea seleccionada por el usuario con el mouse, entre ella, el valor de *CustomerId*, columna del grid), luego ejecuta el código del evento que produjo el post, en nuestro caso, 'Select customer'. Aquí la variable *&customerId* toma el valor del *CustomerId* de la línea elegida. Luego se produce el Refresh general que dispara el Refresh y Load de cada grid.

Por tanto, cada grid se carga ejecutando las conditions. El primero se carga igual que antes, porque ninguna de sus conditions ha variado. El segundo, ahora sí tiene un valor para *&customerId*, por lo que se mostrarán solamente las facturas del cliente.

Uno podría preguntarse por qué se tuvo la necesidad de colocar la variable oculta en el form. Por qué simplemente no podía usársela como variable dentro del programa, sin necesidad de colocarla y hacerla invisible. Con esta segunda ejecución todavía no podemos contestar la pregunta. De hecho, si se analiza, puede verse fácilmente que hubiésemos obtenido el mismo comportamiento si sólo le asignábamos valor a la variable en el 'Select Customer' sin colocarla en el form.

La razón surgirá claramente con la siguiente ejecución...



Ahora el usuario ya tiene seleccionado el segundo cliente, y lo que desea es filtrar sus facturas por fecha (no quiere visualizarlas todas).

Para ello especifica los filtros de fecha, y presiona el botón 'Search', que producirá un post al servidor.

Entonces se ejecutará el Start con su código, luego se leerán las variables de pantalla: aquí está la razón de haber colocado *&customerId* en el form. Se leerá entonces el valor de *&customerId*, que permanecerá incambiado hasta tanto el usuario no vuelva a seleccionar otro cliente del grid y presionar el botón 'Select customer'... esta variable presente en el form, es la forma de mantener la memoria entre ejecuciones.

Recuérdese que cada vez que se hace un post al servidor, es una nueva ejecución del web panel, por lo que las variables comienzan nuevamente vacías. Es por ello que el segundo paso: "Lectura de variables de pantalla" es fundamental.

Ahora sí, siguiendo con el ejemplo, se lee de pantalla la variable *&customerId* invisible, junto con las variables visibles *&startDate* y *&endDate*.

Y luego, como siempre, se ejecuta la carga (Refresh genérico + Refresh y Load de cada grid).

Han cambiado las condiciones del for each que carga el segundo grid, por lo que ahora aparecerán solamente las facturas del segundo cliente, entre las fechas estipuladas por el usuario.

Si ahora el usuario quiere cambiar de cliente, para ver sus facturas, lo seleccionará del grid de clientes y presionando 'Select Customer', el proceso volverá a empezar como vimos en la 2da. ejecución.

¿Y cómo se vuelve a trabajar con las facturas de todos los clientes y no con las de uno dado?

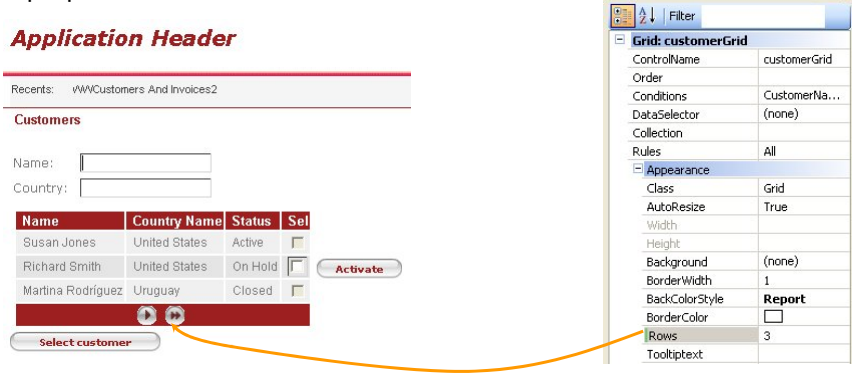
Piense qué pasará si el usuario no selecciona ningún cliente del grid con el mouse, pero presiona 'Select Customer'. El valor de la variable *&CustomerId* quedará vacío, pues *CustomerId* no tendrá valor.

Por este motivo, al botón 'Select Customer' podríamos haberle llamado 'Select/Unselect'.

Web Panels

Grid - Propiedades

- **Paginado automático:** GeneXus realiza un paginado automático si la propiedad Rows tiene un valor distinto de 0.



Inserta automáticamente los botones de paginado.

Los botones que se insertan dependen de la cantidad de registros a mostrar y la cantidad de líneas del grid.

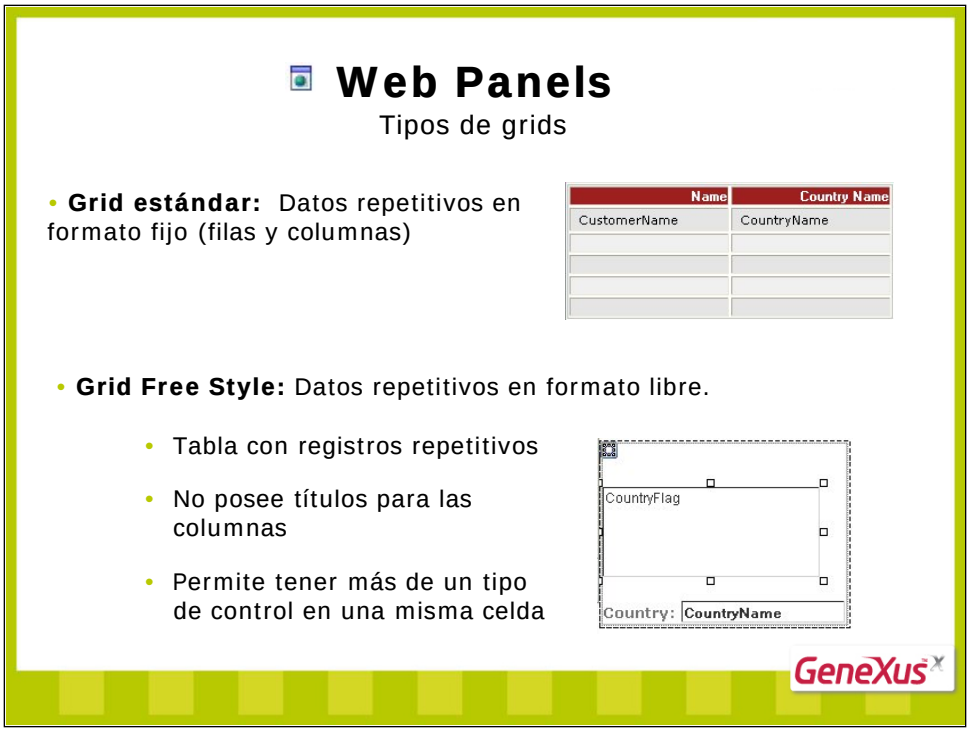
Web Panels

Grid – Ordenamiento automático de las columnas

Las columnas pueden ser ordenadas sin necesidad de programar ningún código adicional: clic sobre el título de la columna.

Esta funcionalidad es válida para grids en transacciones y en web panels.

Se ordena la **página del grid cargada**, por lo que no compite con el Order programado a nivel del grid.



Se dispone de dos tipos de grids:

• Grid estándar: el que vimos hasta ahora, en transacciones y web panels
• Grid Free Style

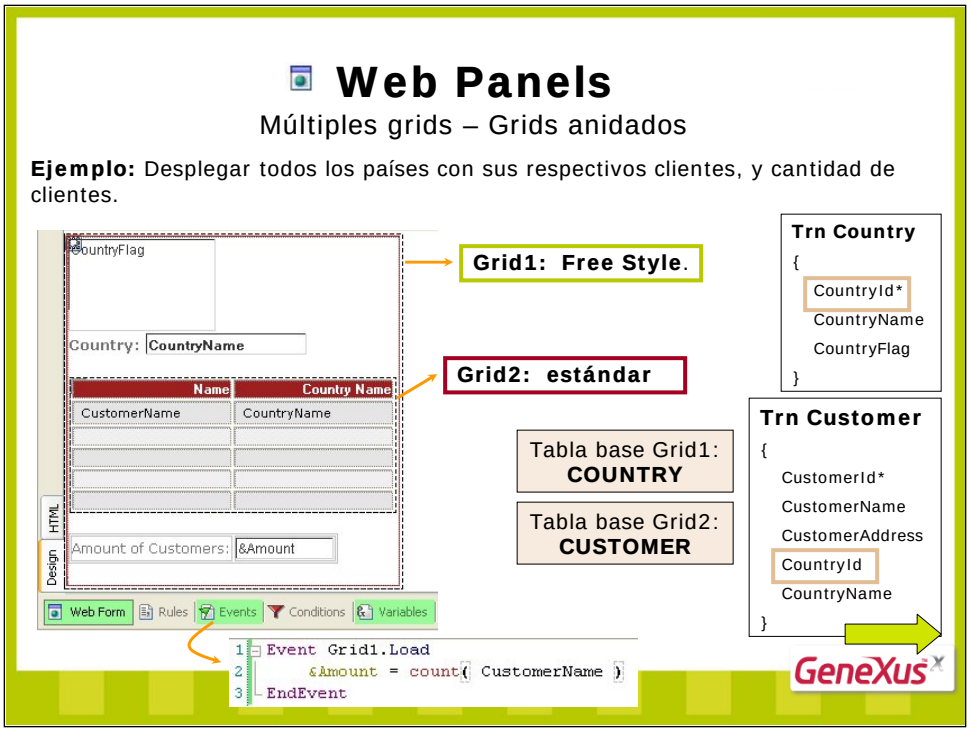
Estos grids agregan potencia al diseño de aplicaciones web, permitiendo al desarrollador mayor libertad a la hora del diseño.

El grid Free Style permite al usuario definir el formato de los datos a desplegar de una forma menos estructurada que el grid estándar.

El grid Free style es básicamente una tabla a la que se le pueden insertar los atributos/variables, text blocks, imágenes, botones, web components, embedded pages, grids freestyle y/o grids que se van a mostrar posteriormente en la pantalla. En este caso para poder visualizar las propiedades hay que seleccionar la tabla donde se encuentran los atributos/variables.

En el ejemplo presentado arriba queremos mostrar alguna información de los países. El atributo *CountryFlag* se ha incluido en la transacción 'Country' para almacenar la foto de la bandera de cada país (es un atributo de tipo Blob). Pero no queremos mostrar la información como lo haríamos en un grid estándar, con cada elemento de información en una columna distinta del grid. Aquí queremos mostrar la foto y debajo el nombre del país.

El comportamiento de las variables dentro de un grid Free Style es análogo al que presentan dentro de un grid estándar, por lo tanto también quedan de ingreso si existe un For each line dentro de algún evento, o si se asocia un evento a cualquier control de la fila. Nuevamente este comportamiento puede modificarse, cambiando la propiedad Read Only.



Este caso de grids anidados es como el de for eachs anidados en el caso de un procedimiento: esto es, aquí sí se relacionan las cargas, y se cargarán por tanto en el Grid2, todos los clientes pertenecientes al país cargado en el Grid1 en cada oportunidad.

De hecho, el orden de ejecución de los eventos estará anidado:

Refresh (genérico)

Grid1.Refresh

Grid1.Load → carga de un país

Grid2.Refresh

Grid2.Load → carga de cliente del país

Grid2.Load → carga de cliente del país

Grid2.Load → carga de cliente del país

Grid1.Load → carga del siguiente país

Grid2.Refresh

Grid2.Load → carga de cliente del país

Grid2.Load → carga de cliente del país

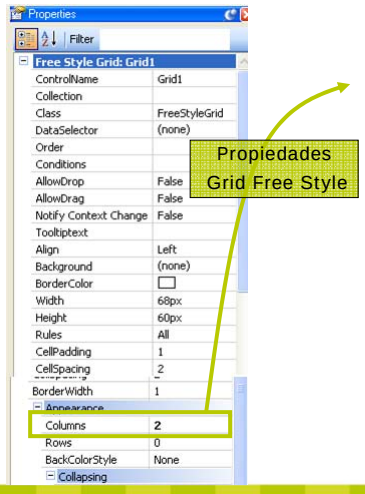
Grid2.Load → carga de cliente del país

.....

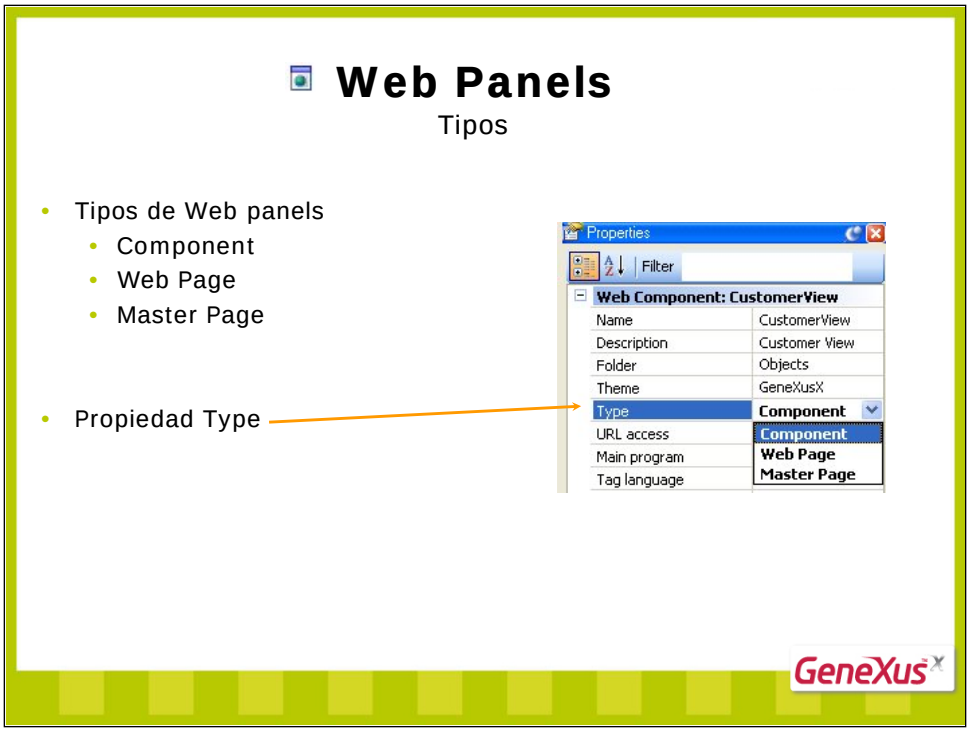
Web Panels

Grid Free Style

(Ejemplo: Continuación)



Propiedades
Grid Free Style



Los objetos web pueden ser definidos con tres tipos diferentes, configurable en la **propiedad Type del objeto**. Para un web panel podrá tomar uno de los valores:

• Component: (transacción ó web panel, que a partir de aquí podrá ser incluido en otro web object – transacción o web panel)

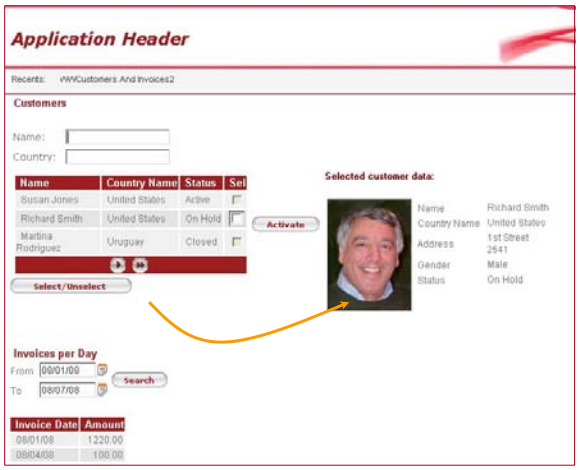
• Web Page (es decir, el objeto será una transacción ó web panel tal como hemos trabajado hasta el momento)

• Master Page

A continuación introduciremos el Web Panel tipo: “Component” y luego volveremos sobre el “Master Page” pero no lo veremos en profundidad en este curso.

Web Panel

Web Component



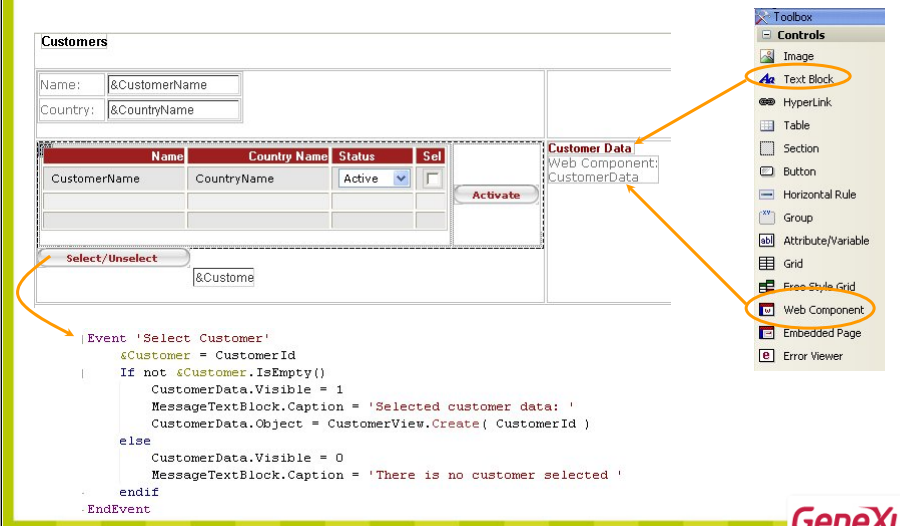
Ya teníamos programado un web panel 'CustomerView' que mostraba los datos del cliente → Reutilicémoslo

Un **component** es un web panel pero que se va a ejecutar dentro de otro.

GeneXus

Web Panels

Web Component




```
Event 'Select Customer'
    &Customer = CustomerId
    If not &Customer.IsEmpty()
        CustomerData.Visible = 1
        MessageTextBlock.Caption = 'Selected customer data: '
        CustomerData.Object = CustomerView.Create( CustomerId )
    else
        CustomerData.Visible = 0
        MessageTextBlock.Caption = 'There is no customer selected '
    endif
EndEvent
```

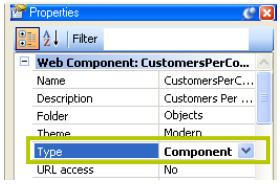
GeneXus

Web Panels

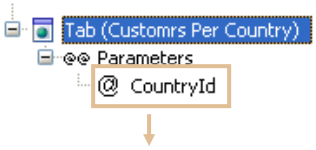
Web Components

Ejemplo: Crear un nuevo tab en la instancia de Pattern Work 'With Countries, mostrando los clientes por cada país.

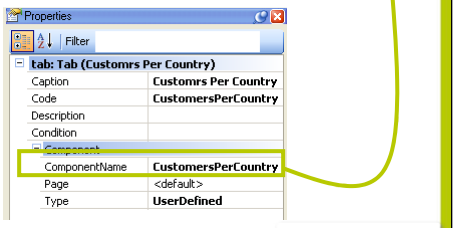
1) Definir el Web Panel CustomersPerCountry como Component



2) Definir el nuevo tab CustomersPerCountry en la instancia de Pattern Work With Countries



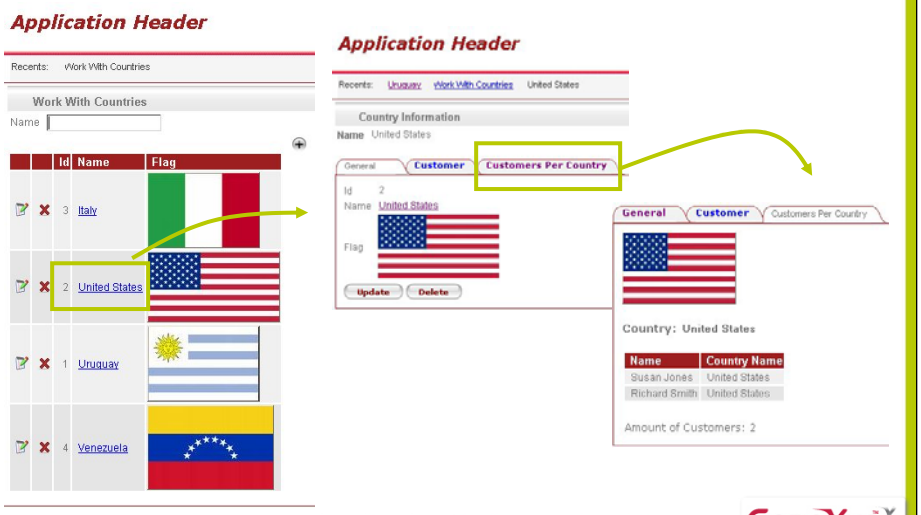
El Web Panel CustomersPerCountry recibe CountryId como parámetro.

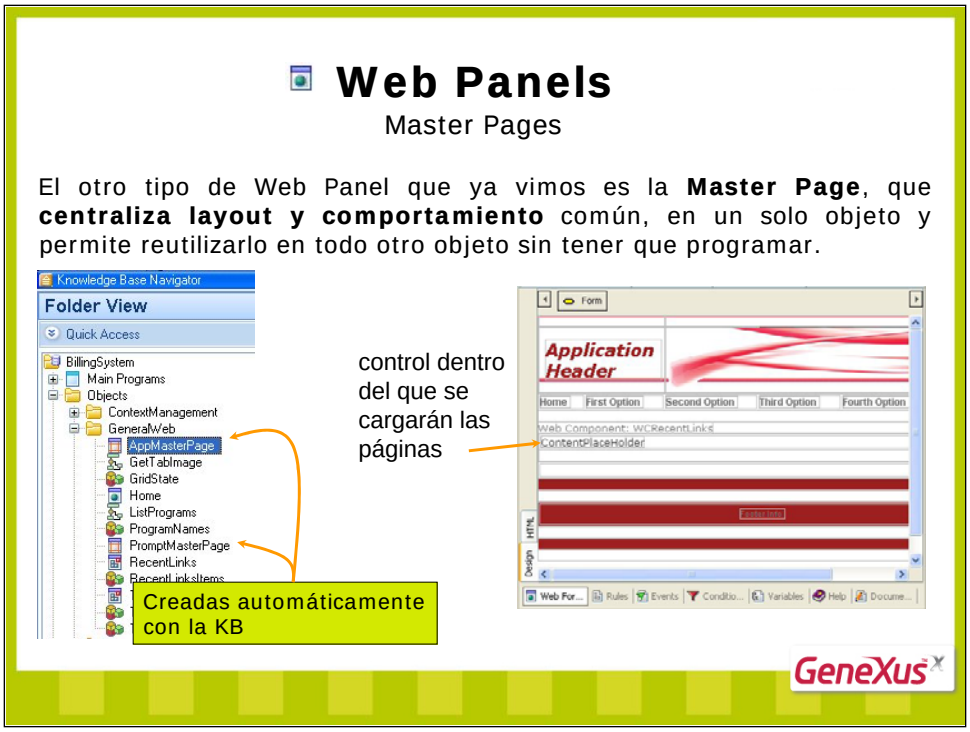


GeneXus

Web Panels

Web Components





Tener un look & feel consistente es hoy en día un deber de toda aplicación Web.

Crear y mantener cada página de una aplicación Web asegurando la consistencia con el resto del sitio toma gran tiempo de programación.

Al crear una base de conocimiento, GeneXus X creará también dos objetos de tipo Master Page:

- ApplMasterPage: Para la aplicación.
- PromptMasterPage: Para los prompts.

Se creará un web panel, ApplMasterPage categorizado como “Master Page” con todo lo que sea el Layout y comportamiento común a todas las páginas del sitio, y en el mismo se dejará un espacio para cargar en cada oportunidad la página que corresponda (el contenido variable del sitio). Corresponde al control especial ContentPlaceholder. Las páginas web que implementan el contenido variable, se implementan como Web Panels o Web Transactions comunes y corrientes (es decir de tipo “Web Page”), y se asocian a la Master Page (a través de la propiedad de igual nombre), de manera que cada vez que se ejecuten, se carguen con ese “contexto”. Abra cualquier objeto GeneXus con form (transacción o web panel) creado en una KB, y vea el valor de la propiedad Master Page.

Las Master Pages proveen una forma de **centralizar el layout y el comportamiento** común en un solo objeto y reutilizarlo en todo otro objeto sin tener que programar. Esto significa que la modificación de alguna parte del layout o del comportamiento común es tan fácil como modificarla en un único objeto y ¡listo!.

En una misma base de conocimiento se pueden definir tantas Master Pages como se desee.