GeneXus X Evolution 3

Web user experience

Diálogo a pantalla completa (full screen)

En este tipo de diálogo no se realizan validaciones de los datos, controles de integridad referencial (IR), inferencias de la tabla extendida, ni disparos de reglas y fórmulas a medida que los datos van siendo ingresados por el usuario en la pantalla.

Las validaciones, reglas y fórmulas se disparan solamente en dos momentos:

1. Al ingresar a la transacción (reglas que no dependen de nada, no tienen condiciones de disparo).
2. Al confirmar los datos.

La **confirmación** de los datos determina la **grabación de la pantalla completa**, disparando previamente todas las reglas, validaciones y haciendo la carga de los atributos de la tabla extendida.

Está en la naturaleza de los generadores Java y .Net trabajar con este tipo de diálogo. Sin embargo existe la posibilidad de modificar este comportamiento, para que el usuario final pueda tener mayor interacción con la aplicación de manera tal que no deba esperar a confirmar la transacción para ver los atributos inferidos, así como las validaciones y disparo de reglas y fórmulas. En lo que sigue explicaremos el diálogo a pantalla completa con Validación a nivel del Cliente.

Diálogo a pantalla completa con Validación a nivel del Cliente (Client Side Validation)

La base de este diálogo es a pantalla completa, pero el usuario tendrá mayor interacción

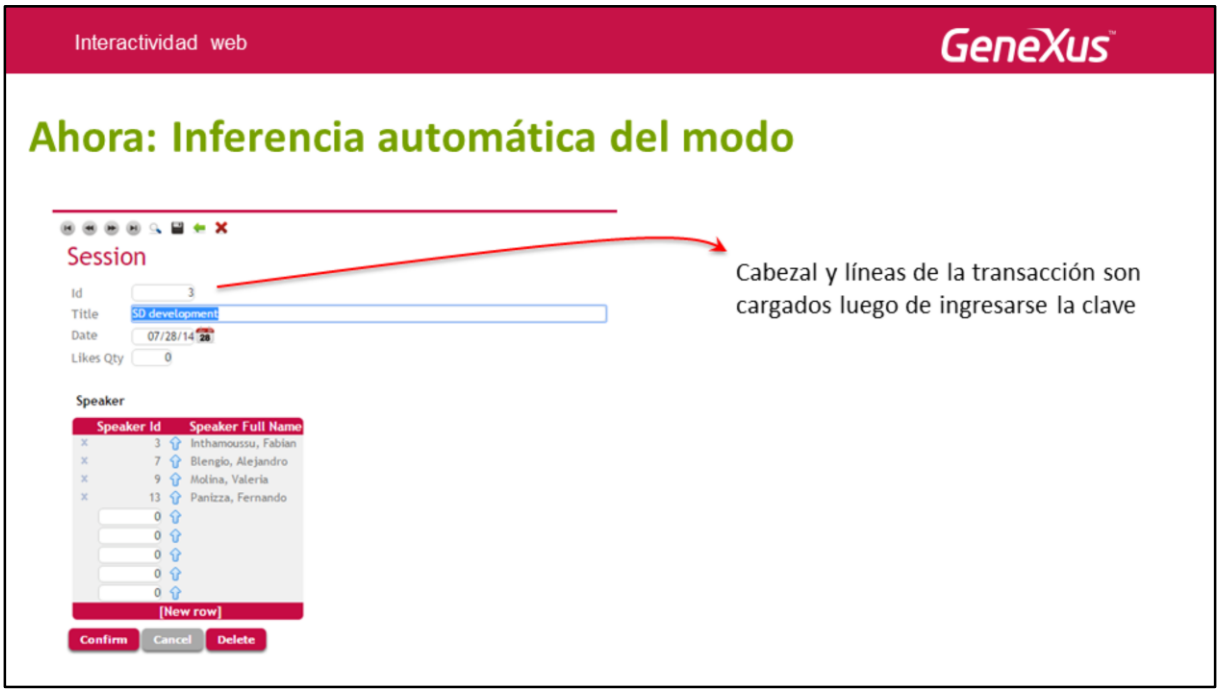
con la aplicación.

Tanto las validaciones de datos, como los controles de IR y disparo de algunas reglas y fórmulas, además de realizarse cuando se confirma la transacción (como en el diálogo full screen clásico), también se realizarán antes, en forma interactiva a medida que el usuario vaya ingresando y abandonando los campos de la pantalla.

Las grabaciones de los registros sí se realizarán a pantalla completa, tras la confirmación.

De modo que se trata de un híbrido entre los diálogos campo a campo y a pantalla completa.

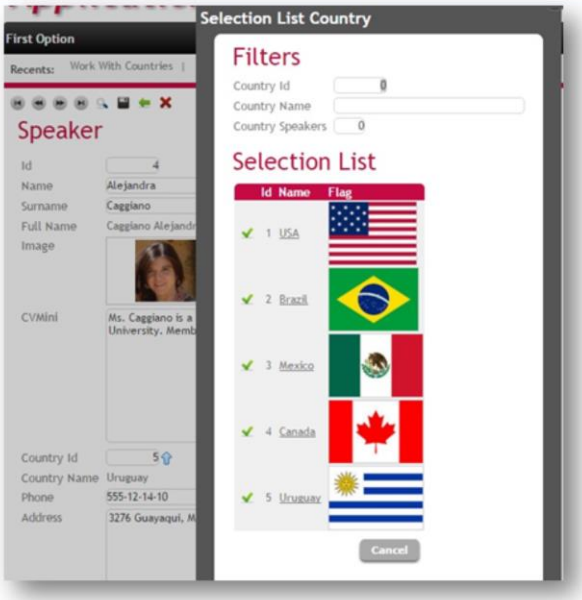
Nota: Si bien la naturaleza de internet no posibilitaba la validación a nivel del cliente (es decir, enviar partes de la lógica de las transacciones al browser), con el advenimiento de Ajax esto es posible.



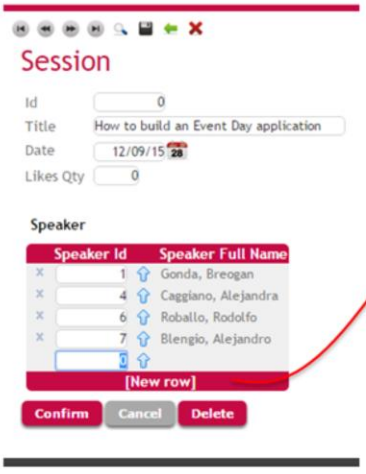
Obvervar que ya no aparece más botón ‘Get’ para lograr POST al server y cargar el form de la transacción con la información correspondiente, así como obtener el modo. Ahora es realizado automáticamente mediante un call javascript que realiza un request al server (Ajax). Con esto se logra mayor interactividad con el usuario.

Prompts

Los prompts se visualizan como ventanas emergentes modales.

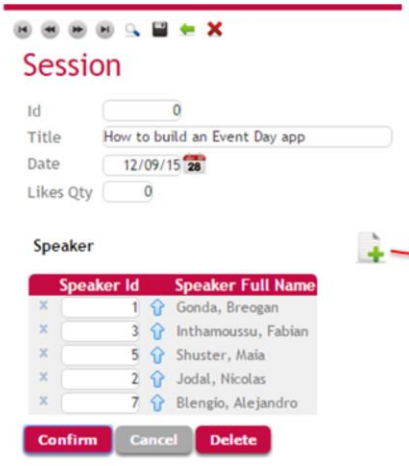


Agregar líneas vacías



Opción [New Row] o cuando se sale del último campo de la última fila → se agrega línea vacía.

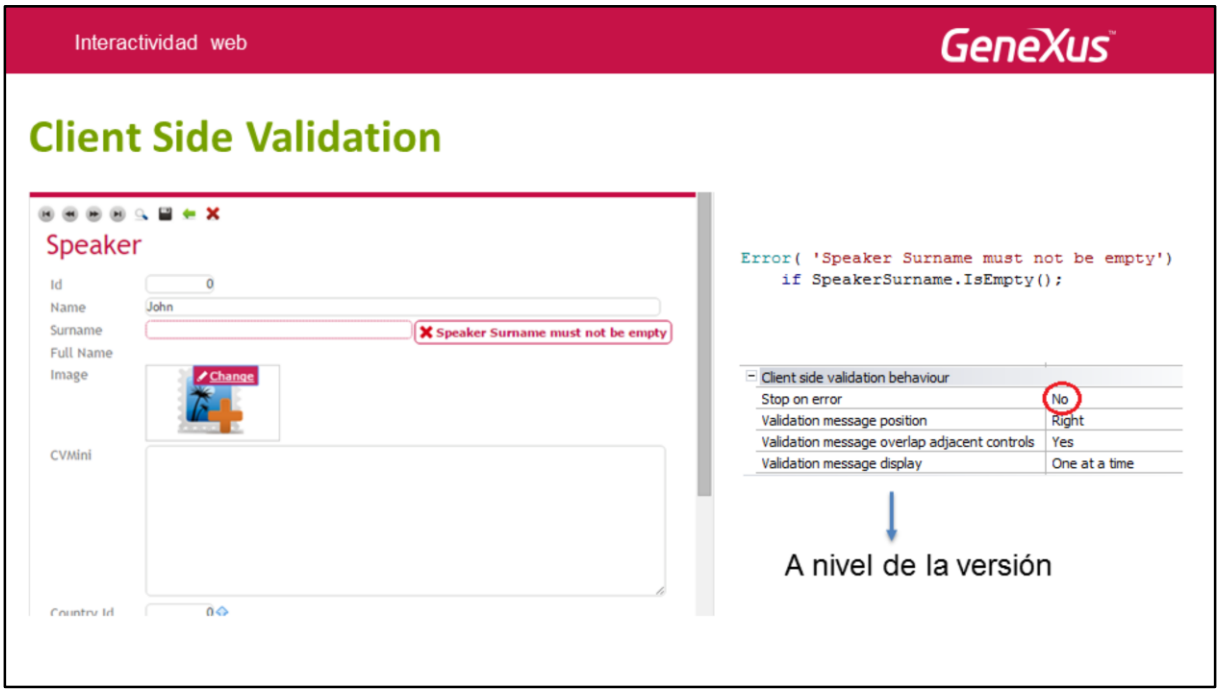
Agregar líneas vacías



Si el método **AddLines** del grid es utilizado, opción [New Row] desaparece.

Ej:

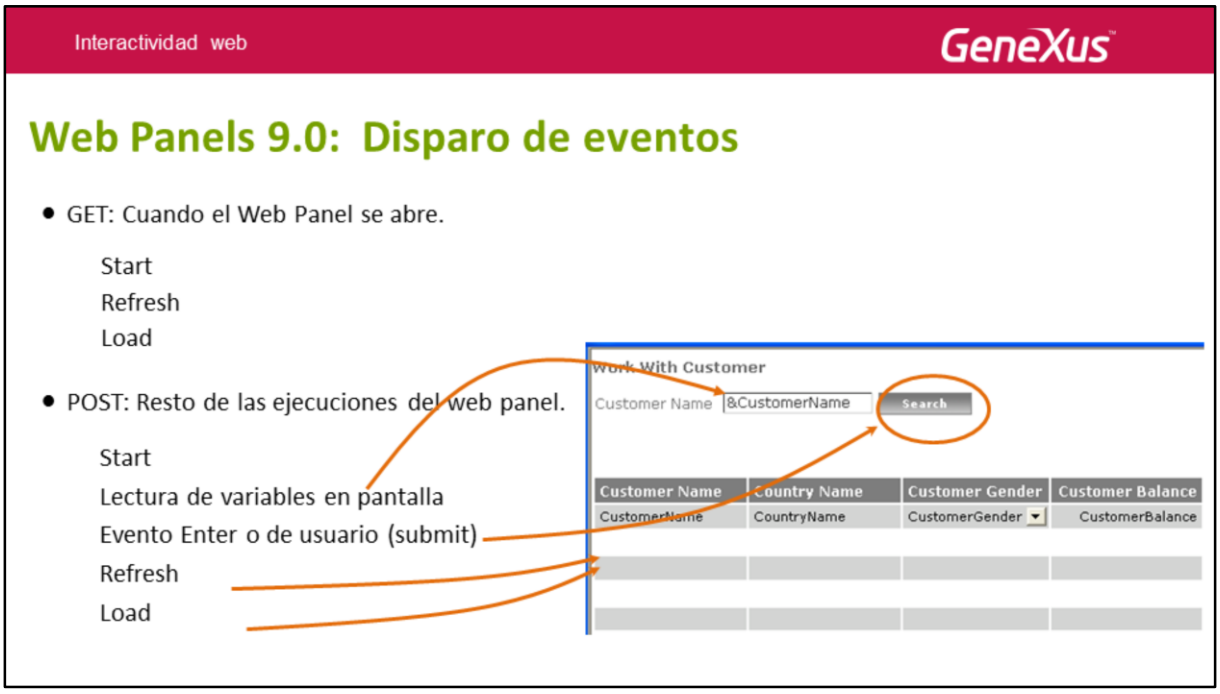
```
Event InsertImg.Click
    GridSession_Speaker.AddLines(1)
endevent
```



Entre las propiedades de la versión tenemos las que especifican el comportamiento de la validación a nivel del cliente.

Obsérvese que por defecto se ha cambiado el comportamiento frente a los errores. En el ejemplo se ha especificado una regla de error en caso de encontrarse el apellido del orador vacío. Si en ejecución se abandona el campo sin completarlo, se dispara el error que se muestra en pantalla, pero se le permite al usuario ingresar el siguiente campo. Por supuesto, esto es sólo a la hora de interactuar con la pantalla. Si el usuario completa los campos pero deja el Surname vacío, al intentar grabar la regla de error se ejecutará del lado del servidor y no permitirá ingresar el registro en la base de datos.

Vea que también aparecen propiedades para configurar dónde se desea que aparezcan en la pantalla los mensajes y errores, etcétera.



Los web panels en la versión 9.0 de GeneXus seguían, salvo excepciones, el esquema clásico de trabajo en Internet. En el mismo, cuando el usuario accede a una página del servidor Web para visualizarla, el Browser baja la página al cliente. Por lo tanto, (sin ajax) no existe forma de detectar lo que realiza el usuario: el servidor Web volverá a tener el control cuando se dispare el evento ENTER o algún evento de usuario o click. En ese momento se envía (se somete) el resultado al servidor para continuar con su procesamiento.

Es decir, una vez que el objeto web finaliza la ejecución en el servidor no queda en memoria. Como consecuencia, la forma en que programábamos este tipo de aplicaciones presentaba algunas diferencias con respecto a lo acostumbrado en ambientes no web.
Es por esta razón que era importante destacar el orden en que se disparan los eventos y el momento en que las variables adquieren el valor ingresado por el usuario.

El orden de ejecución de los eventos en web panels 9.0 es diferente si se trata de la primera llamada al mismo (GET) o si se disparó algún evento de usuario, enter o click (POST).

Recordemos cómo era este mecanismo…

La **primera vez que se ejecuta el web panel** (se conoce también como el momento en que se hace el **"GET"** de la página) los eventos que se disparan son los siguientes y en el siguiente orden:

1. Start
2. Refresh
3. Load

Luego de esto, cuando el usuario haga clic sobre un control que tenga asociado el evento Enter o uno de usuario o click, se ejecutará nuevamente el web panel y el orden de disparo

de los eventos será diferente, como se indica.

En el resto de las ejecuciones del web panel, que ocurren cuando se presiona un botón, o se fuerza la ejecución del evento asociado a una imagen, text block, etc. (haciendo clic sobre el control que tiene asociado el evento de usuario, Enter o click) momento que se conoce también como el **"POST"** de la página, los eventos se dispararán en el siguiente orden:

1. Start (nuevamente se dispara este evento)

2. Lectura de las variables de la pantalla. Esto se realiza porque el usuario puede haberlas modificado (por ejemplo las variables de la parte fija del web panel que están involucradas en las conditions, como en el ejemplo que se presenta arriba, donde se quieren cargar en el grid los clientes cuyo nombre contenga el string cargado por el usuario en la variable *&CustomerName*)

3. Evento Enter o click o evento de usuario (código correspondiente al evento asociado al control que se presionó y produjo el POST).

4. Refresh (genérico y luego el del grid)

5. Load (del grid)

En el ejemplo no necesitamos codificar nada en el evento asociado al botón Search. Solo lo pusimos para poder enviar al servidor la variable con el valor que ingresó el usuario y que la página se refresque cargando en el grid los clientes que cumplan con el filtro que el usuario estableció mediante esa variable. Sin embargo esto ahora ya no es necesario, como veremos a continuación con respecto a las nuevas funcionalidades a nivel del control Grid.

Web Panels ahora

- Nuevas funcionalidades en grids
- Algunos **eventos** se ejecutarán del lado del Cliente.
- Smooth web user experience

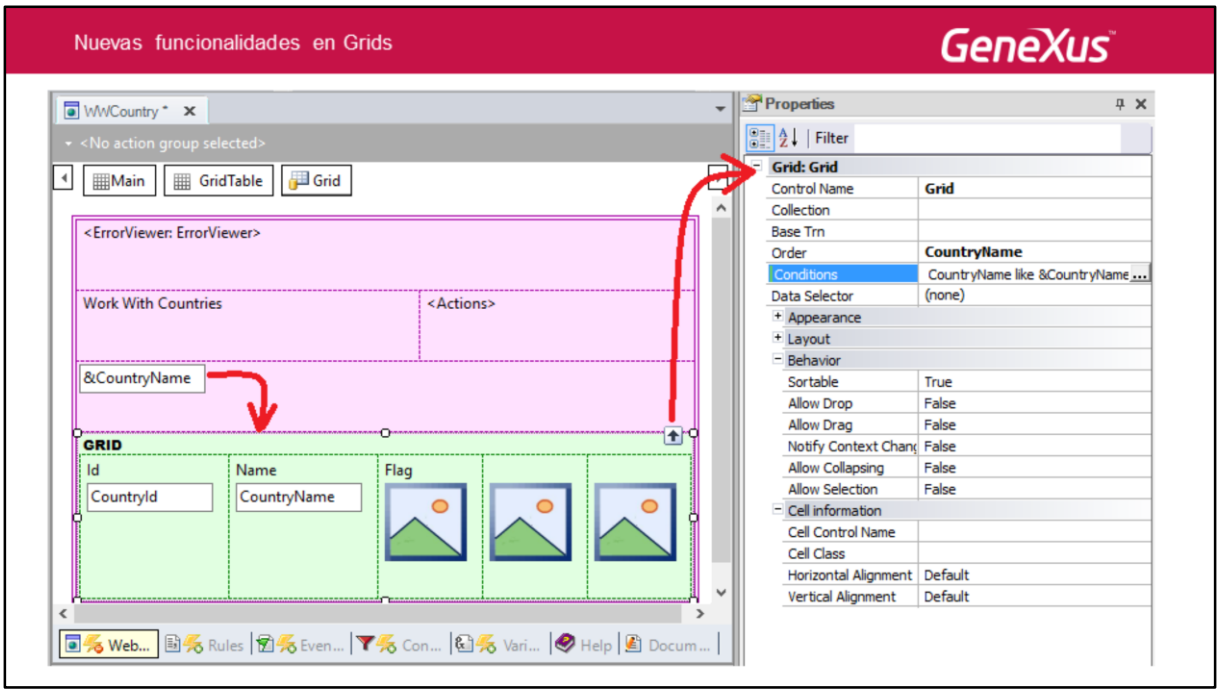
Veremos en lo que sigue estos tres puntos.

Nuevas funcionalidades en Grids

Generalidades

Los grids en Web Panels cuentan con las siguientes mejoras:

- Refresh automático
- Ordenamiento automático de las columnas
- Paginado automático



Primero que nada, veamos en este ejemplo que el grid tiene dentro de sus propiedades las conocidas Order, Conditions y Collection (que permitirá que un grid basado en una variable simple se pueda colocar como colección), así como las nuevas Base Trn y Data Selector que se estudiarán más adelante.

Este Web panel es el que creó el pattern Work With aplicado a la transacción Country. Vea que aparece una variable &CountryName en el form, para que el usuario pueda filtrar los países que se muestran en el grid. Obsere que no se ha colocado un botón para conseguir el refresh.

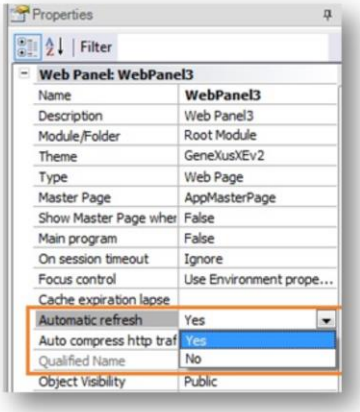
GeneXus

Refresh automático

Un Grid (ya sea con o sin tabla base) puede cargarse en forma automática una vez que los filtros relacionados son ejecutados, dependiendo del valor de la propiedad Automatic Refresh.

Valores posibles:

- Yes (Default value)
- No



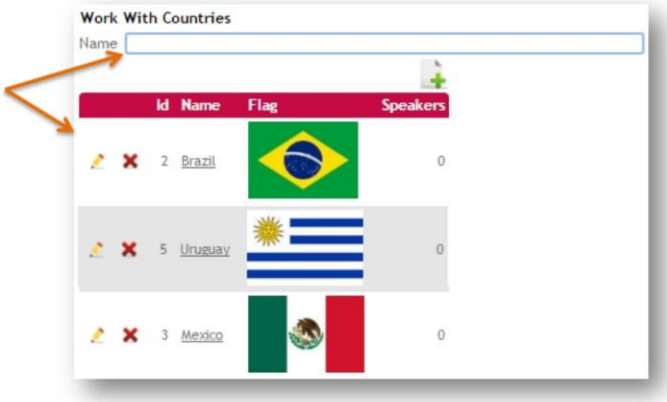
Refresh automático: Yes

Luego que el usuario ingresa los filtros, la grilla carga automáticamente sin necesidad de hacer un clic.

Ejemplo: Grilla de países

Cuando el usuario digita en el filtro, la grilla se filtra automáticamente.

Dependiendo del control y del tipo de datos, el filtro se hace mientras se escribe o cuando se sale del control.



Dependiendo del tipo de datos del filtro, y del control web utilizado para filtrar, la condición será aplicada cuando se está ingresando 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, 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.

GeneXus

Paginado automático

GeneXus realiza un paginado automático si la propiedad Rows del grid tiene un valor distinto de cero.

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

GeneXus

Ordenamiento automático de las columnas

Durante la ejecución de la aplicación, las columnas pueden ser ordenadas sin necesidad de programar ningún código adicional, con sólo hacer clic sobre el título de la columna:

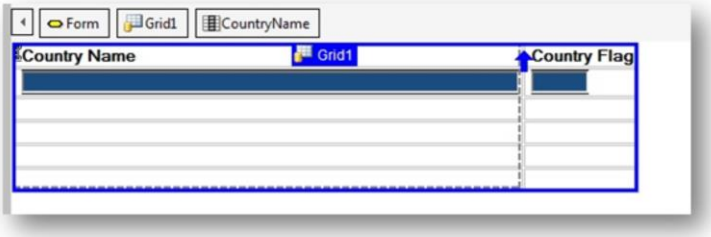
Se ordena la página actual cargada → no compite con la propiedad **Order** especificada a nivel del control grid.

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

GeneXus

Drag & Drop de las columnas

Durante el diseño de la transacción o del web panel, las columnas pueden ser intercambiadas con solamente hacer Drag & Drop



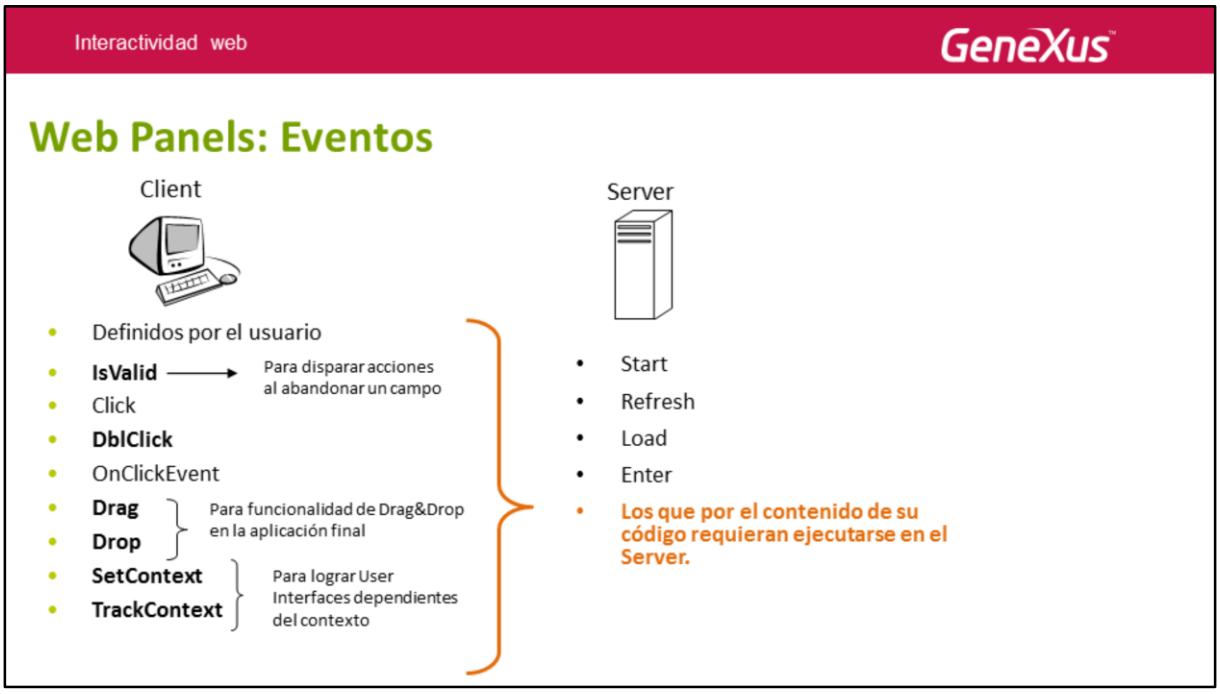
Al seleccionar la columna a mover, se muestra una flecha azul indicando los posibles lugares donde se puede colocar dicha columna. Se posiciona el mouse en uno de esos lugares (Drag) y allí se “suelta” la columna (Drop).

Eventos ejecutados en el Cliente

- Algunos **eventos** se ejecutarán del lado del Cliente.
  - Evitando roundtrips innecesarios al servidor, y reduciendo la cantidad de datos que viajan de un lado a otro.
  - Algunos **eventos de usuario** deben ejecutarse en el Server pero otros no → se ejecutarán solo en el Cliente.
    - Ejemplo: un evento que cambia el estado de un control no necesita ejecutarse en el Server.
  - GeneXus tiene la inteligencia para determinar estos casos.

```
Event 'X'
  control.visible  = 0
EndEvent
```

Esto valdrá tanto si la Web User Experience elegida es smooth (la veremos al final) o no (y el comportamiento es como hasta ahora).



Los resaltados en negro son nuevos eventos (el DblClick existía antes pero no funcionaba).

Cualquier evento cuyo código pueda ser resuelto enteramente ejecutando javascript en el Browser lo será, de modo tal de evitar el viaje de información al Server.

Llamaremos eventos de usuario a todos los que define el usuario y asocia luego a controles, así como los listados arriba (IsValid, Click, DblClick, OnClickEvent, Drag, Drop, SetContext, TrackContext). Los eventos Start, Refresh, Load y Enter, que son del sistema; serán eventos ejecutados **necesariamente en el Server**.

Los eventos indicados arriba como de ejecución en el cliente no necesariamente serán ejecutados allí. Si, debido al código que contiene uno de estos eventos, no puede ser resuelto en el cliente (porque, por ejemplo, modifica el valor de una condition que es utilizada en la carga de un grid, es decir, obliga un Refresh), entonces será ejecutado en el Server. En otras palabras, a simple vista no sabemos si uno de estos eventos será resuelto en el cliente o en el servidor. Esto lo determina GeneXus en función de la interrelación de todos ellos. Hablaremos de esto en lo que sigue. De todos modos el lugar donde GeneXus ejecuta el código de un evento será transparente para el programador que sólo debe preocuparse de la lógica. Si GeneXus puede optimizar ejecutando en el cliente, lo hará. Si no puede, ejecutará en el Server.

Evento IsValid

- Útil para disparar acciones luego de abandonar un campo del form.
  - Se dispara, si el control es:
    - Edit → cuando se abandona el campo.
    - Checkbox, listbox, combo, radiobutton → cuando se selecciona un valor en el control.
  - En web form solo se dispara el evento si el valor del campo cambia.
  - Los 'messages' dentro del evento serán desplegados en el control ErrorViewer.

Ejemplo

&CountryId ⟶ En el form de un Web Panel

```
Event &CountryId.IsValid
      WebComp1.Object = CountryInfo.Create( &CountryId )
Endevent
```

- Luego de ingresar un valor para la variable del campo, se desea crear un web component mostrando la información del país.
- Si en otra ejecución el valor de esa variable no se modifica, no volverá a dispararse el IsValid. Solo se volverá a disparar una vez que cambie.

Recuerde que en Web solo dispara el evento IsValid si detecta un cambio en el valor del campo.

Drag & Drop en la aplicación final

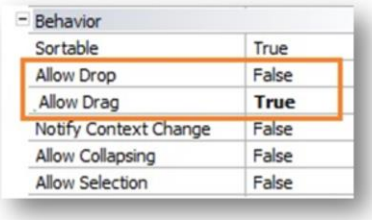
Drag & Drop

Arrastrar *información* desde un lugar a otro, con motivo de tomar ciertas acciones.

Drag & Drop

Propiedades de **Grid**

- AllowDrag: permite que contenido del grid sea arrastrado.
- AllowDrop: permite que se le agregue al grid contenido que se haya arrastrado de otro lado. Se resuelve en el Cliente.



Eventos Drag & Drop

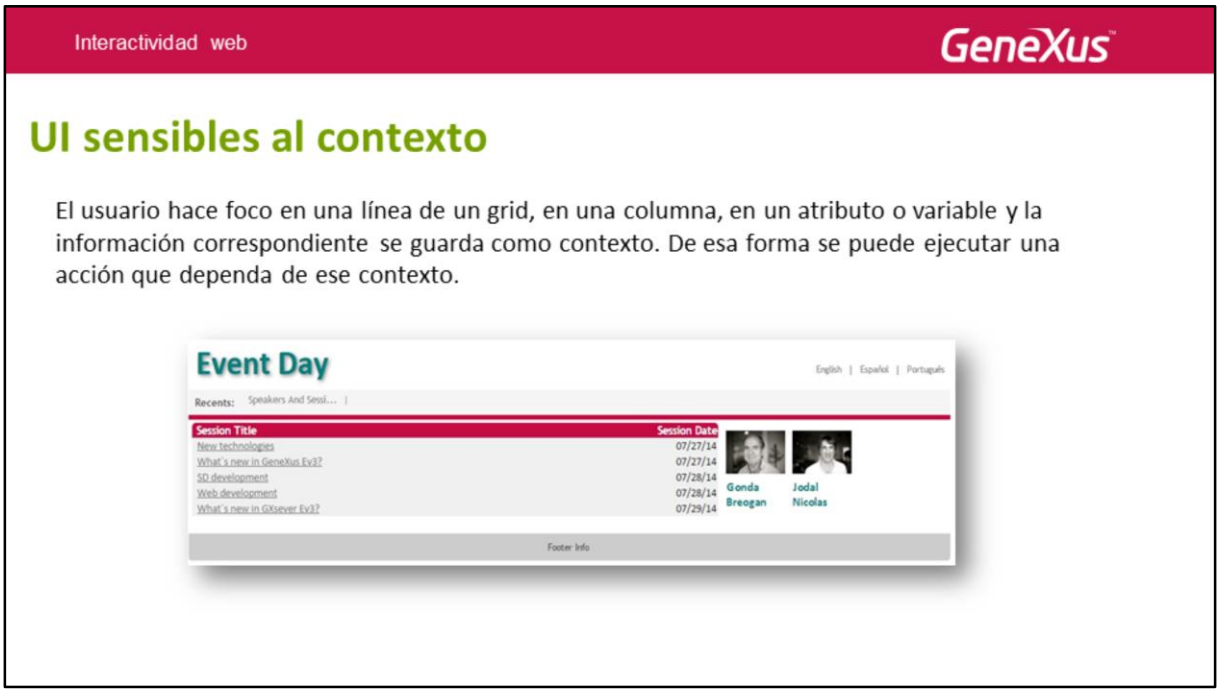
- Aplica a controles:
  - Image
  - TextBlock
  - Table
  - (Freestyle) Grid
  - Web Component
  - Button

Drop Event

control.**Drop**(&p_1, ..., &p_N)

Los parámetros solo pueden ser variables

Aplicaciones sensibles al contexto

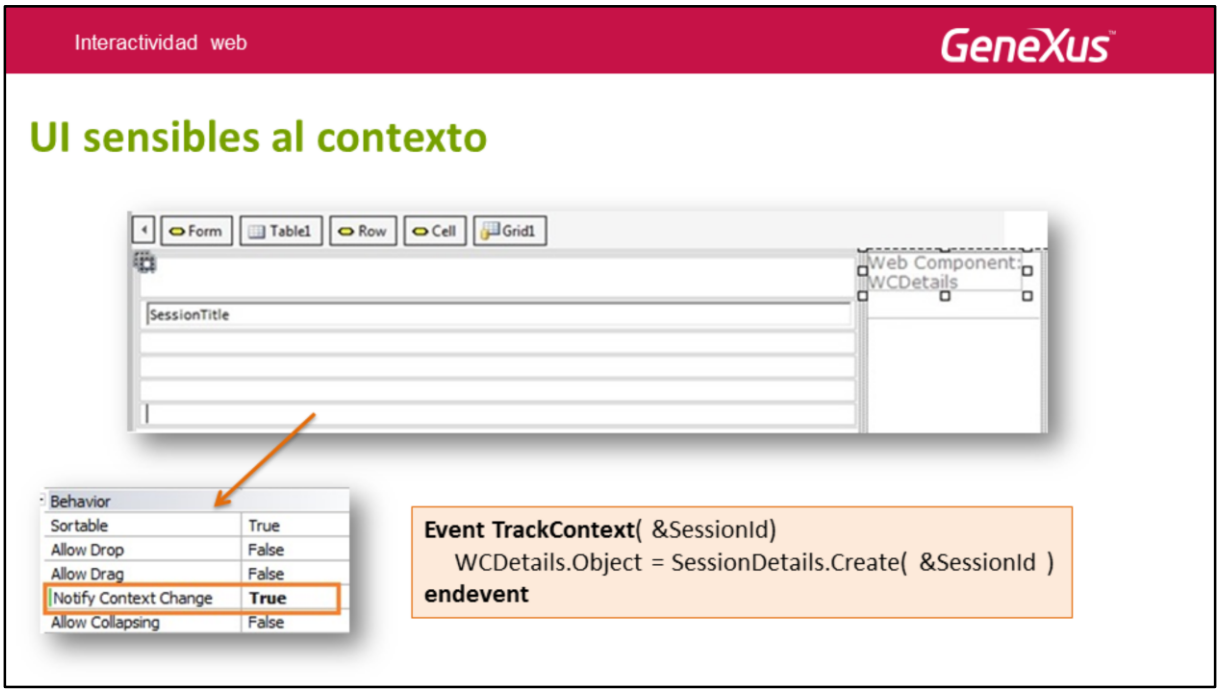


La tendencia actual es implementar interfaces orientas a la intención, que pueden ser abordadas llevando cuenta del contexto de la aplicación y tomando las acciones pertinentes.

La mejor introducción es un ejemplo: el usuario elige una línea del grid de la izquierda, que muestra todas las conferencias. Al hacerlo la aplicación detecta un cambio en el contexto, y dispara una acción, en la cuál se programa la visualización a la derecha, de los datos de los oradores.

¿Qué entendemos por contexto? Cuando hablamos de contexto, nos referimos al contexto de nuestra aplicación en un form. Cuando accedemos a un textbox de SpeakerId, nuestro contexto es Speakerid. Cuando accedemos al grid correspondiente a las conferencias, nuestro contexto es una línea de ese grid, y así.

Cuando nos movemos dentro de la pantalla, estamos cambiando el contexto de atributos y variables. Esta información acerca de los cambios en el contexto es esencial para crear aplicaciones con interfaz orientada a la intención.

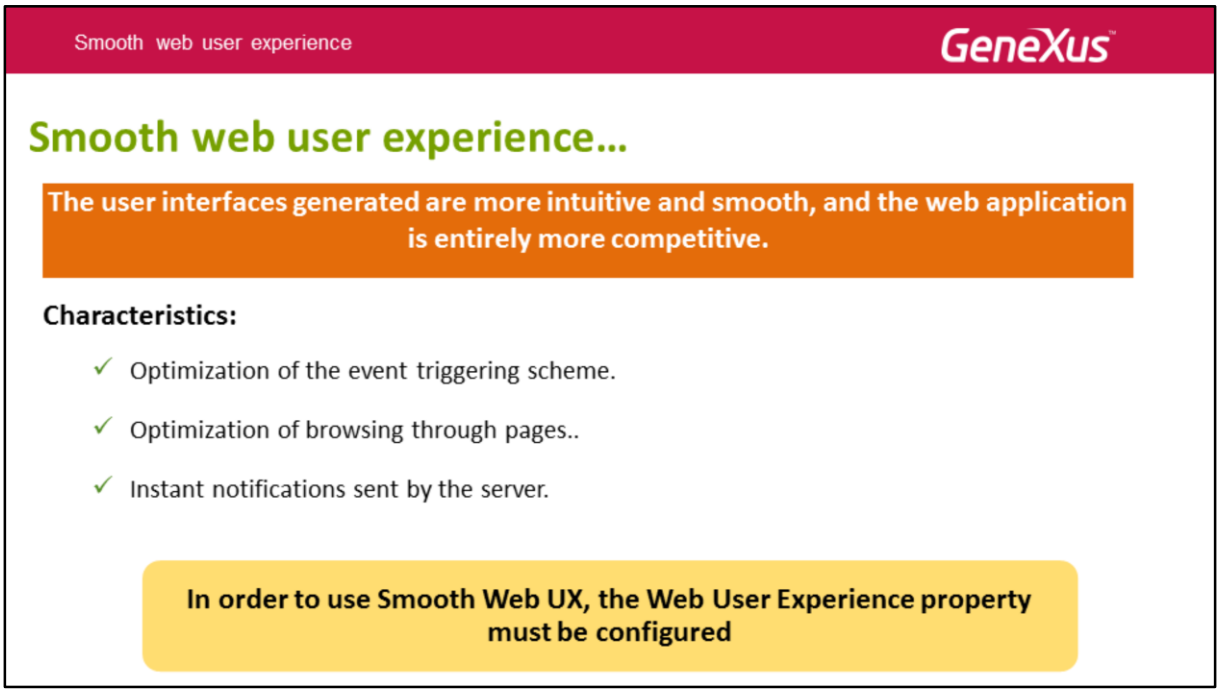


El grid de nombre Grid1 (que muestra las conferencias), tiene una propiedad **Notify Context Change** configurada en True. ¿Qué significa esto? Que habilita a que se haga un seguimiento de cambio del contexto relacionado al grid, es decir, si el usuario selecciona una línea, la aplicación lo detectará, guardando esta línea seleccionada como contexto. En nuestro caso tenemos oculto el atributo *SessionId*.

Ahora habrá que programar la carga del web component WCDetails con los datos de la conferencia elegida en el grid. Para ello se programa el evento TrackContext, que recibe como parámetro el &SessionId de la línea elegida (es decir, la variable guardada en el contexto, correspondiente a la columna de la línea elegida, que es el atributo *SessionId* hidden).

Este evento se disparará cuando el contexto cambia. De esta manera, cuando el usuario elige una línea, se detecta, y se dispara el evento, creándose entonces en el control Web component de la derecha, el web component 'SessionDetails' mostrado.

Smooth
web user experience



Las interfaces de usuario generadas son más intuitivas y "smooth", y la aplicación web en su totalidad es más performante debido al mecanismo utilizado para resolver el diálogo cliente-servidor.

Las características asociadas a la experiencia de usuario Smooth son:

- Optimización del esquema de disparo de eventos: La ejecución de eventos es óptima en el sentido que solamente se ejecutan los eventos necesarios. Esto implica que la página se "refrescará" menos, el usuario recibirá feedback rápidamente y solamente la mínima información necesaria es enviada al servidor.

- Óptima navegación entre páginas: La navegación entre páginas se resuelve utilizando AJAX, y es más intuitiva para el usuario debido al "esfumado" que brinda la transición y que puede ser diseñada de la forma que mejor quede a la aplicación.

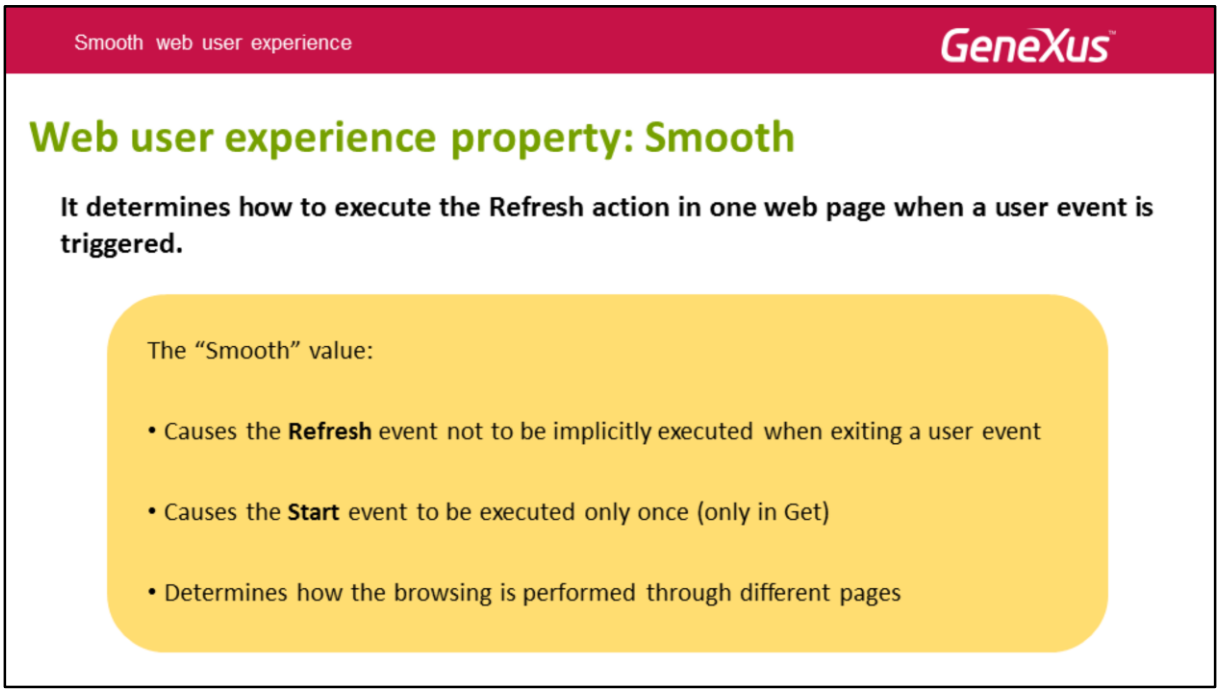
- Notificaciones al instante: A través de las notificaciones web el usuario puede ser avisado de cualquier evento sin la necesidad de refrescar la página web, ya que es el server quien envía las notificaciones.

Smooth web user experience property...

Property at the version level and at object level:

Possible values:

- Smooth – The mechanism is triggered. It is the default value when creating a new KB.
- Previous versions compatible – The mechanism is disabled. Its performance will be the same as in previous versions



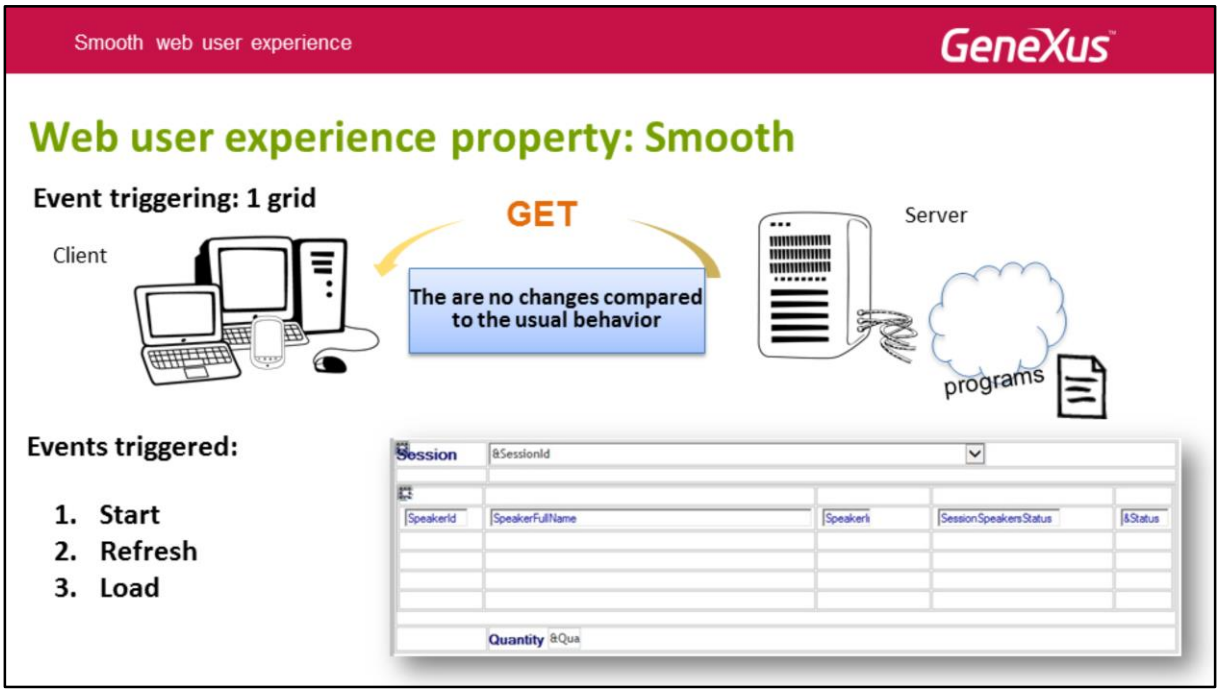
El valor Smooth en la propiedad Web User Experience determina cómo se ejecuta la acción Refresh en una página web cuando se dispara un evento de usuario dentro de la propia página.

También determina cómo se realiza la navegación entre distintas páginas.

El valor "Smooth" hace que el evento **Refresh** no se ejecute en forma implícita cuando se sale de un evento de usuario, y que el evento **Start** se ejecute una sola vez. Además cada web component es independiente de la página principal que lo contiene. Esto significa que el Refresh de un web component **no afecta al resto de la página**.

El valor Smooth de la propiedad Web User Experience también habilita "Single Page Applications", donde navegar hacia una página web que está contenida en la misma Master Page no provoca el refresh de la página completa en el browser. Solamente actualiza la página contenida en el contentplaceholder

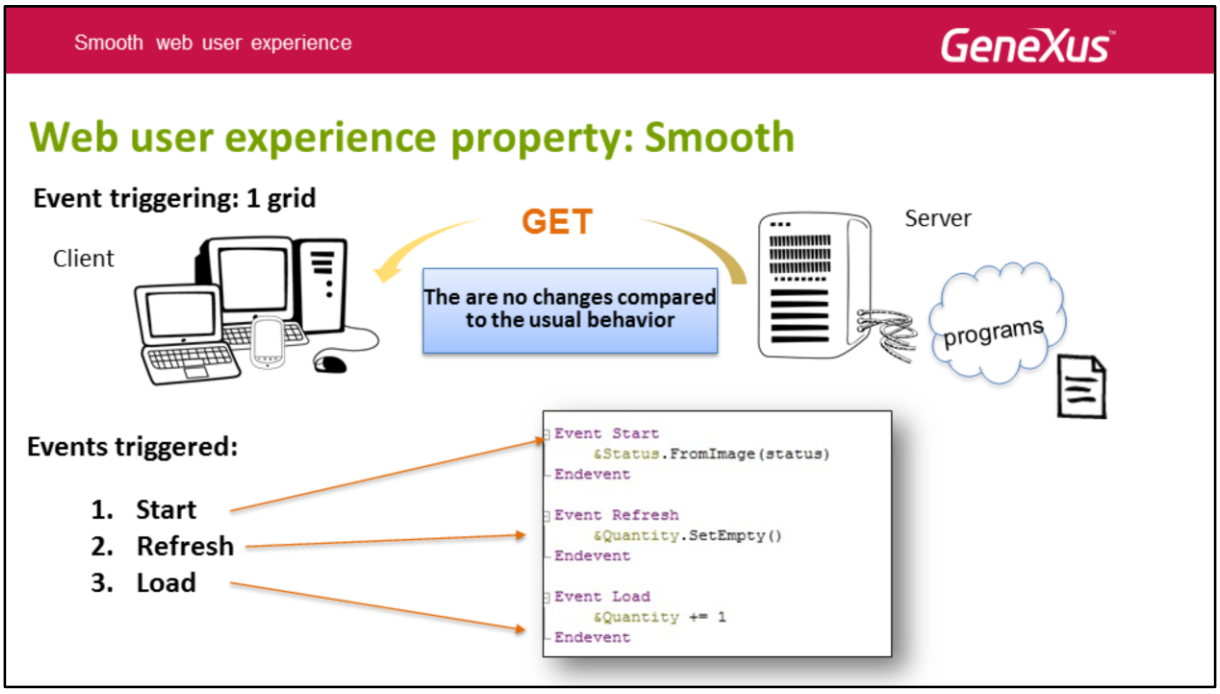
Otro beneficio de esta propiedad es la forma en la cual se ejecuta el comando Load dentro de un evento de usuario. En este caso el comando **Load** no fuerza el refresh del grid completo, sino solamente de la línea en cuestión.



Este web panel permite seleccionar una conferencia (session) de todas las registradas, mediante la variable &SessionId presentada con el control combo dinámico (habiendo configurado las propiedades asociadas a dicho control). Luego de elegida la sesión, se muestran en el grid todos los oradores registrados para dicha sesión, cada uno con su status actual (Confirmado o Cancelado).

Abajo, se desea ver en una variable la cantidad total de oradores registrados para la sesión seleccionada.

¿Qué sucede cuando invocamos al web panel desde el navegador, por primera vez? Lo que se conoce como hacer un **GET**. El cliente le va a pedir al servidor que ejecute el web panel y como respuesta el servidor le envía un archivo html que le indica al navegador cómo dibujar la pantalla (con qué datos, con qué formato, etc.).



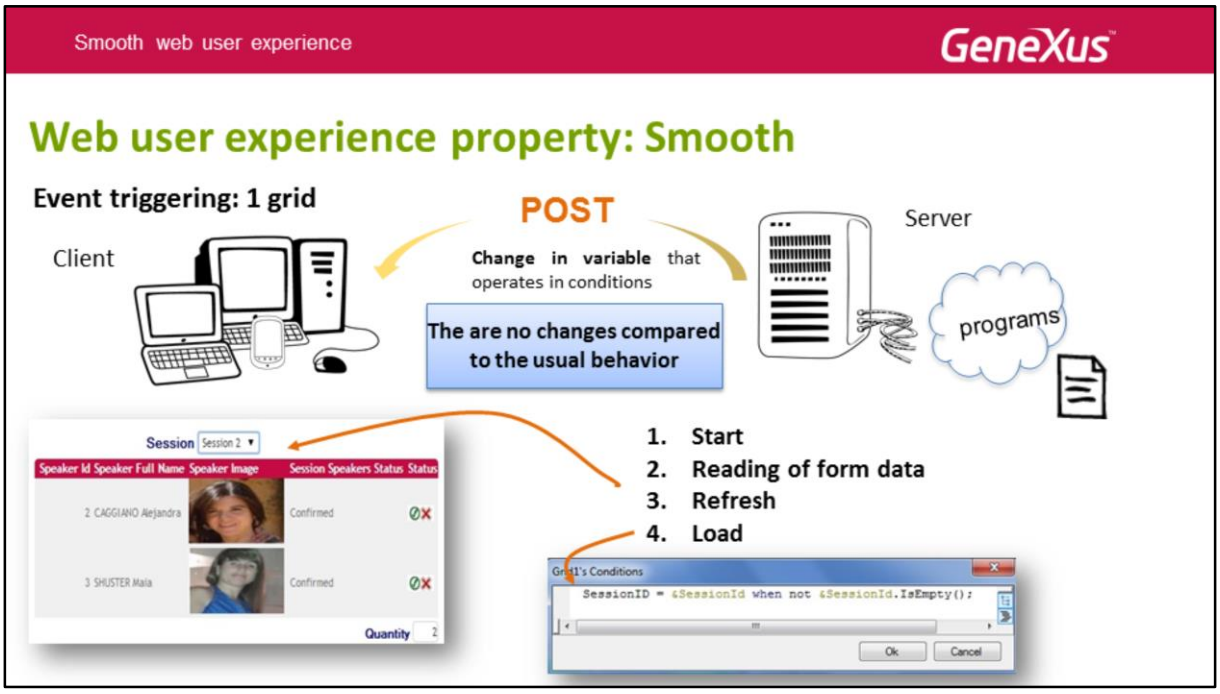
Veamos qué eventos se disparan en el web panel y en qué orden…

Primero se dispara el evento Start, ejecutándose el código definido en dicho evento. Por lo que hemos incluido dentro del mismo, se carga la imagen del símbolo Confirmado/Cancelado en la variable &Status.

En segundo lugar se dispara el evento Refresh, que se ejecutará 1 vez, provocando el acceso a la tabla física correspondiente, e inmediatamente después vendrá la ejecución del evento Load, una vez **por cada** registro navegado, ya que el web panel del ejemplo tiene tabla base por tener atributos presentes en el grid.

Como consecuencia, los datos de cada registro se agregarán como línea del grid, en el html que se enviará y verá al cliente. Es decir, si hay 10 registros en la tabla base de la consulta, el evento Load se ejecutará 10 veces seguidas: una vez por cada registro.

Observemos que en el evento Refresh, hemos inicializado la variable &Quantity (porque este evento, como dijimos, se ejecuta 1 vez justo al comenzar a realizarse la consulta aplicando el filtro, y si no inicializamos dicha variable en ese momento, la cantidad de oradores se seguiría acumulando, sin reinicializarse, cuando el usuario cambie la elección de la session en el dynamic combo para ejecutar la consulta de nuevo).



Analicemos ahora qué sucede cuando el usuario ingresa (o selecciona en este caso) un nuevo valor en la variable que se usa para filtrar.

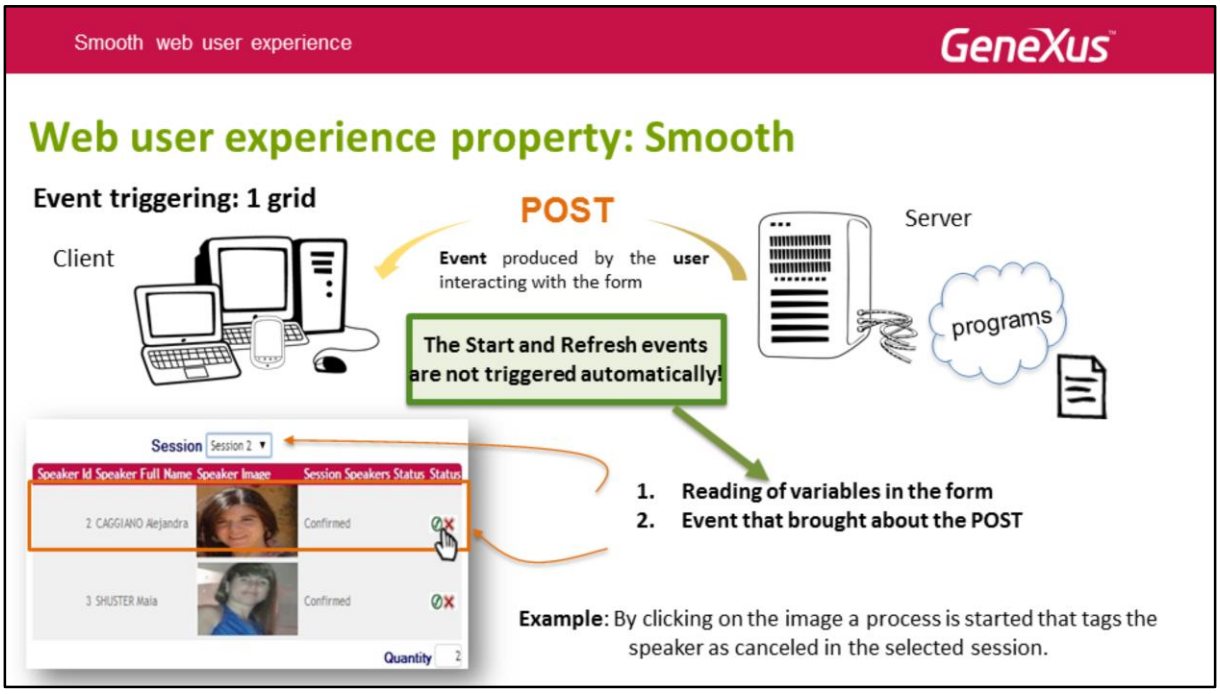
Por defecto, las variables que intervienen en las Conditions, provocan un Refresh automático. Es decir, la consulta vuelve a ejecutarse y la pantalla vuelve a cargarse. O sea que el programa asociado al web panel debe volver a ejecutarse en el servidor.

Pero, ¿qué se ejecuta en el servidor y en qué orden?

Primero el evento Start En este caso, se carga la imagen del símbolo de Confirmado/Cancelado.

Segundo la lectura de los datos de pantalla (que le fueron enviadas en el post). Allí se obtiene el valor que el usuario dio a la variable &SessionId.

Tercero el Refresh, e inmediatamente los Loads que correspondan. Para cada registro de la tabla base, Speaker, se estará verificando si cumple la condición, es decir, si su orador tiene el valor de SessionId indicado en la variable &SessionId, y en caso afirmativo, se ejecuta el Load, y se agrega como línea del grid, al archivo con el resultado. Terminado el proceso, se devuelve el archivo al navegador del cliente, que dibuja la pantalla con esos datos.



En forma general, un **POST** se produce cada vez que se efectúa alguna acción en el cliente, que requiere volver al servidor a ejecutar (por ejemplo, porque como consecuencia, deben refrescarse los datos).

Acciones de este tipo pueden ser presionar la tecla Enter o algún botón o control asociado a un evento de usuario.

Si bien los eventos de usuario **no disparan automáticamente el evento Refresh**, el desarrollador podrá ejecutar un Refresh utilizando el comando Refresh, cuando sea necesario.

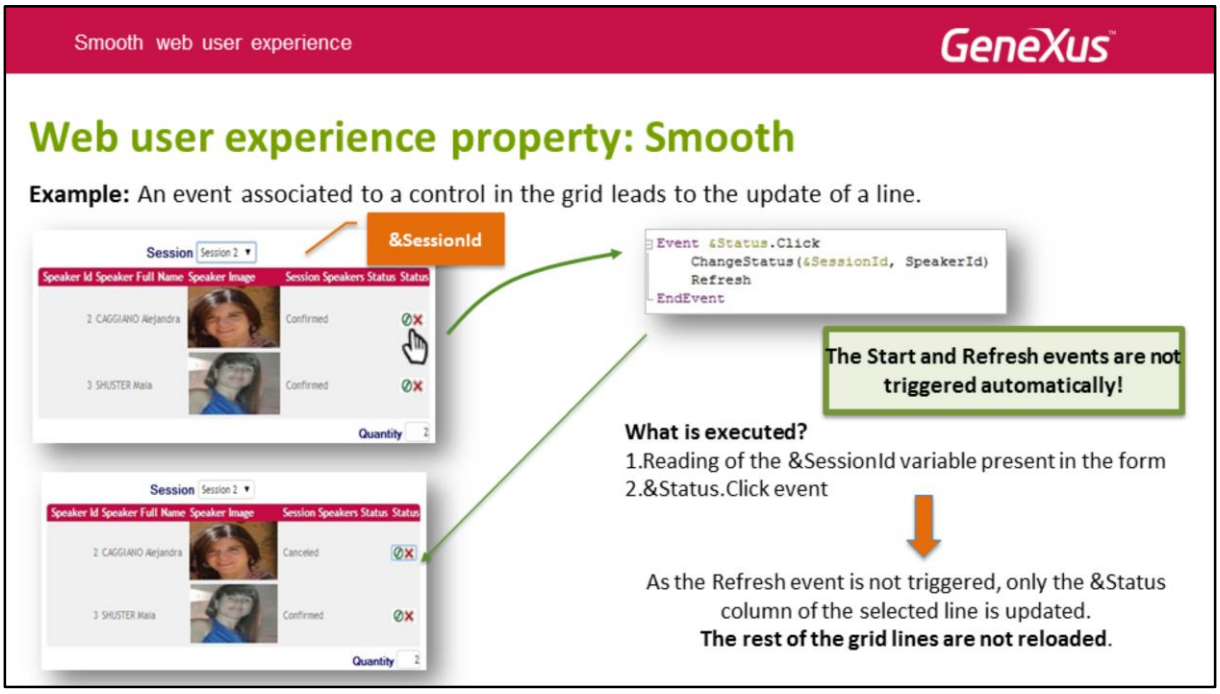
Entonces, cuando se ejecuta un evento de usuario, sucede lo siguiente:

- Se leen la variables en pantalla
- Se ejecuta el evento de usuario que provocó el Post.

Tanto las variables del form como los parámetros del web panel están dentro del alcance del evento de usuario.

Como consecuencia de esto, cuando un evento de usuario se dispara para una línea del grid, solamente afecta a esa línea porque el grid no se carga nuevamente

Veamos un ejemplo…

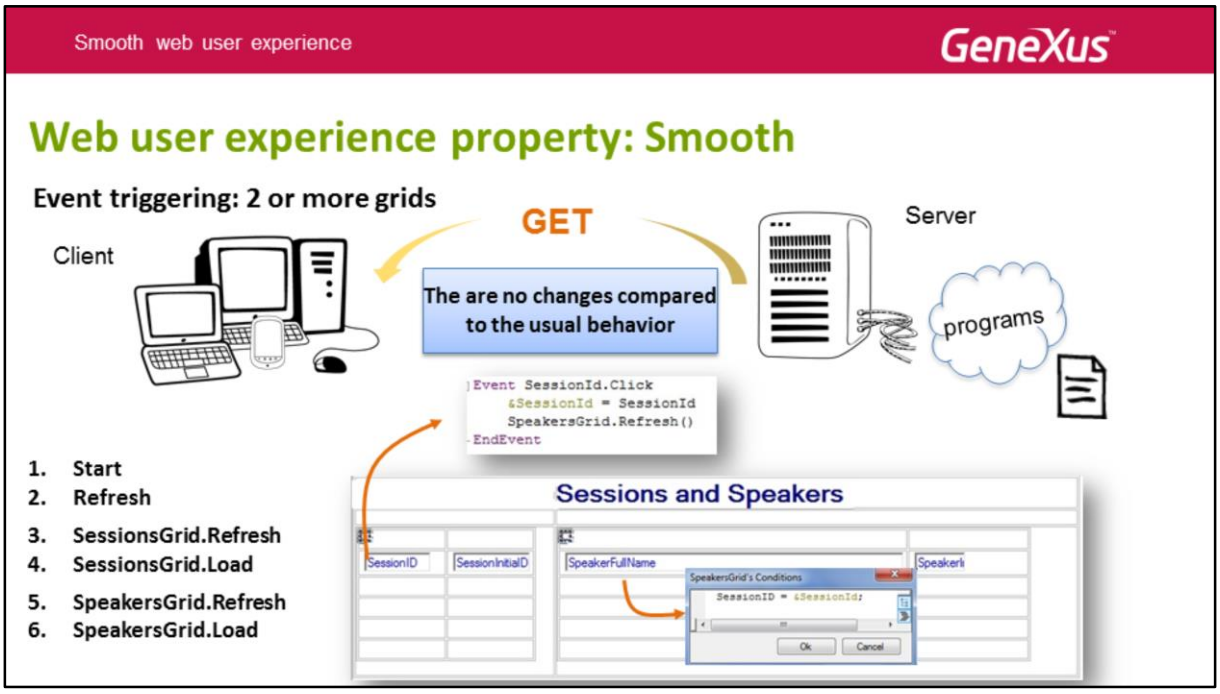


Tenemos un web panel con un grid, que permite seleccionar una sesión (Session) a través de la variable &SessionId definida como un combo dinámico, y muestra los oradores registrados (Speakers).

Si bien la sesión tiene inicialmente confirmados a todos sus oradores, es posible que en un determinado momento alguno sea cancelado
Por lo tanto haciendo clic en la imagen del Status de un orador en la grilla , el mismo será marcado como Cancelado.

Se definió entonces el evento &Status.Click que llama al procedimiento ChangeStatus que se encarga de realizar el cambio de estado para el orador seleccionado.

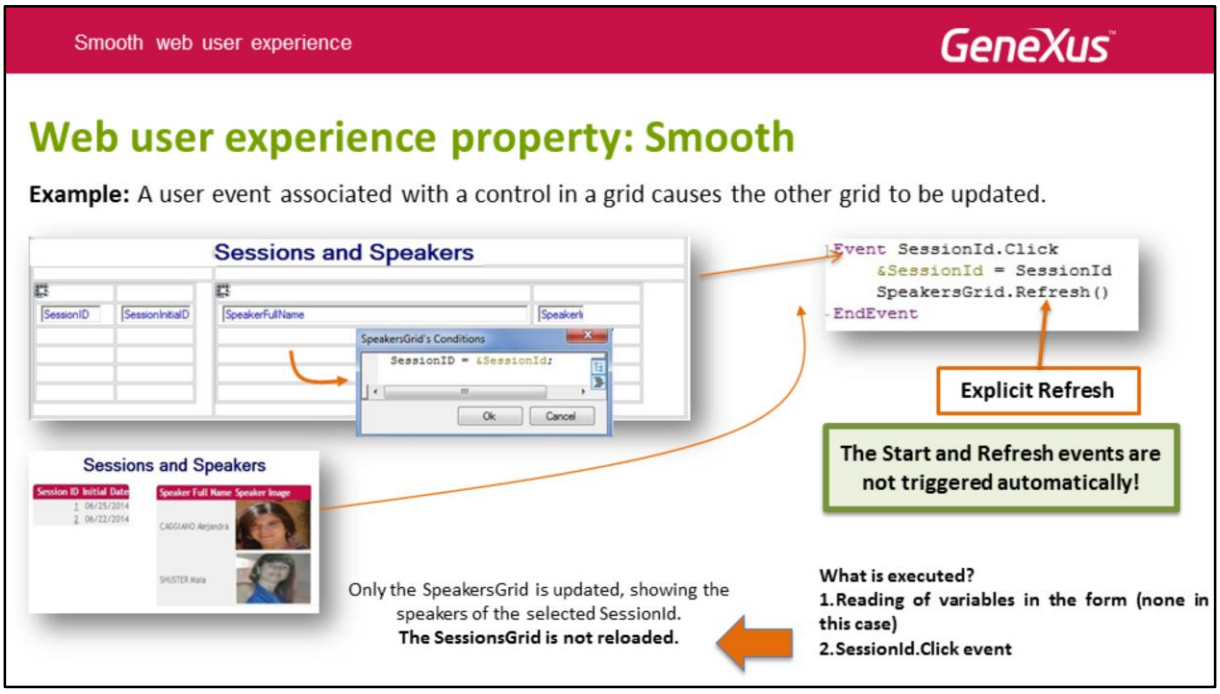
Al no dispararse el evento Refresh, solamente se actualiza la columna involucrada, de la línea seleccionada, las restantes líneas del grid no se vuelven a cargar.



Al llamar a este web panel desde el cliente se ejecutarán, en orden:

1. El evento Start
2. El evento Refresh, general, que ejecuta su código (de haberse programado) y llama al refresh y load de cada grid. Así, llama a:
3. Evento Refresh del primer grid del form, y éste al
4. Load de ese grid, una vez por cada registro . Y luego
5. Evento Refresh del segundo grid del form, y éste al
6. Evento Load de ese grid, una vez por cada registro que cumpla las conditions. Es por eso que en el GET, como la variable &SessionId está vacía, pues no se ejecutó aún el evento Click que la carga, para el segundo grid no se cargará ninguna línea.

Veamos ahora qué sucede cuando el usuario hace click sobre el identificador de una sesión (SessionId), o sea, cuando se realiza un POST.



En el web panel anterior hemos definido el evento de usuario SessionId.Click.

De esta forma, al hacer click sobre SessionId se realiza la lectura de variables en pantalla (en este ejemplo no hay ninguna), y luego se ejecuta el evento definido.
En este evento se guarda en la variable &SessionId el identificador de la sesión seleccionada, y se provoca luego explícitamente el Refresh del grid que muestra los oradores (SpeakersGrid).

Es necesario provocar este Refresh en forma explícita ya que de lo contrario no se va a realizar (ya que el evento Refresh no se disparará automáticamente) y entonces el SpeakersGrid no mostraría ningún resultado

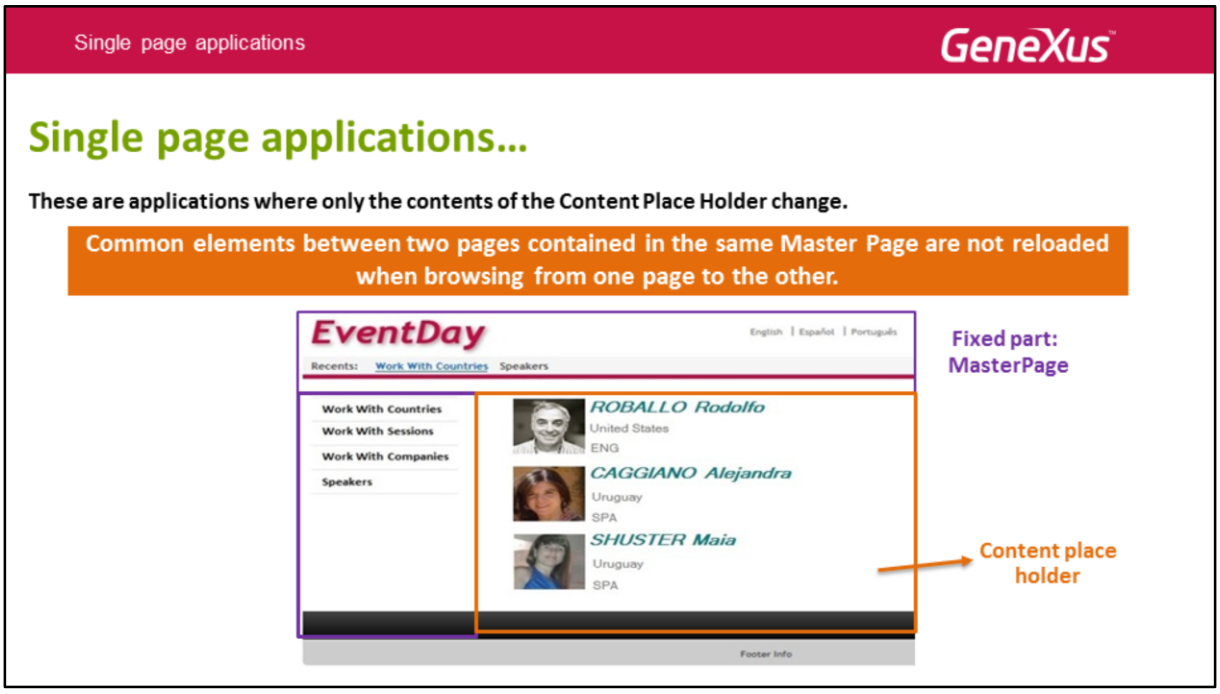
Observar que el SpeakersGrid tiene definida una condición que indica que se filtre por la excursión almacenada en la variable &SessionId.

Una vez ejecutado entonces explícitamente el Refresh del SpeakersGrid, se disparará a continuación el evento Load del SpeakersGrid y por lo tanto se visualizarán los oradores registrados en la sesión seleccionada en el SessionsGrid.

El SpeakersGrid se actualiza, mientras que el SessionsGrid no se vuelve a cargar.

En cuanto al evento Start, solamente se ejecutó una vez, al efectuarse el GET del web panel.

Single page applications



Se le llama aplicaciones de una sola página, a las aplicaciones donde lo que cambia es solamente el contenido del Content place holder.

Navegar hacia una página que es sustancialmente similar a la página actual no debería forzar a refrescar la página entera en el browser sino solamente actualizar lo necesario.

En GeneXus, los comandos Call y Link se utilizan para navegar entre páginas. Ambos utilizan Ajax y esto hace posible llamar a la nueva página sin la necesidad de refrescar la página en su totalidad. El resultado es que los elementos que ya se encuentran presentes en la página no serán recargados.

Generalmente las aplicaciones GeneXus utilizan una Master Page donde el desarrollador establece los componentes que son comunes a todas las páginas de la aplicación (cabezales, menúes, etc)
En esta ocasión cuando el usuario navegue entre las páginas contenidas en la misma Mastar Page, no se recargarán los elementos en común.

O sea, navegar desde la página A hasta la página B, cuando ambas comparten la misma Mastar Page, solamente actualizará el Content Placeholder sin provocar la recarga de la página completa.

Además:

- La Master Page actualizará solo los elementos necesarios (como los RecentLinks del pattern Work With).
- La URL en el browser también será actualizada, y los botones Back y Forward podrán ser utilizados en el esquema tradicional.


GeneXus