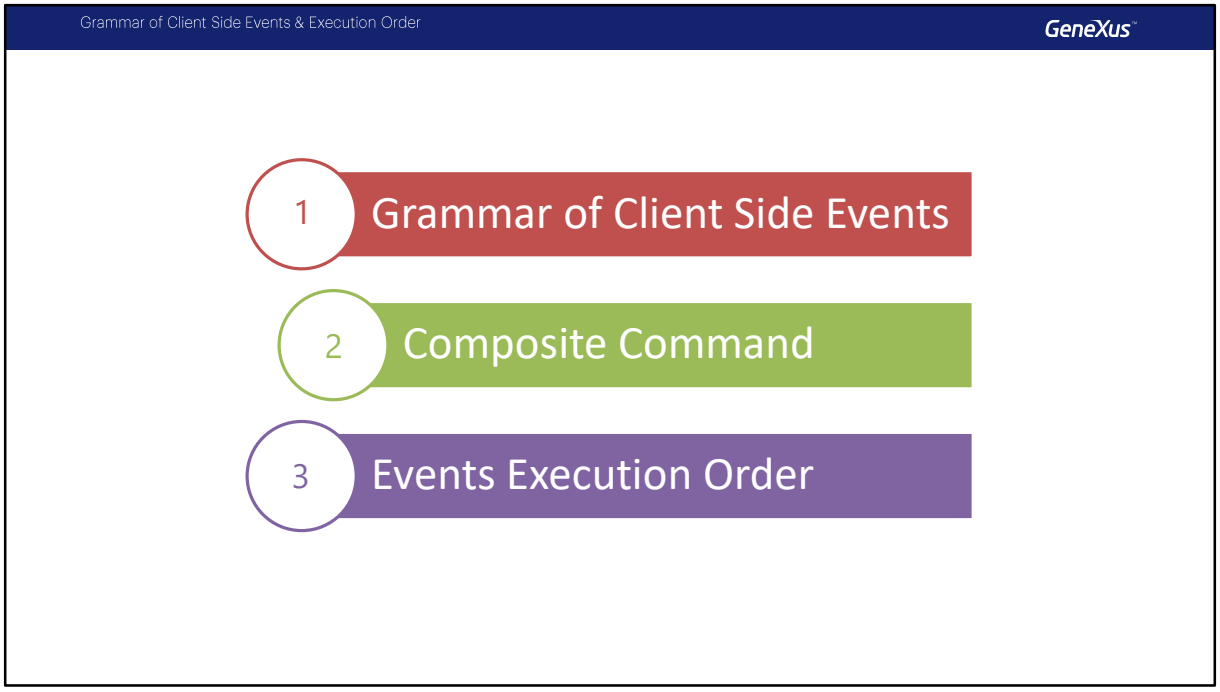

GeneXus™
The power of doing.

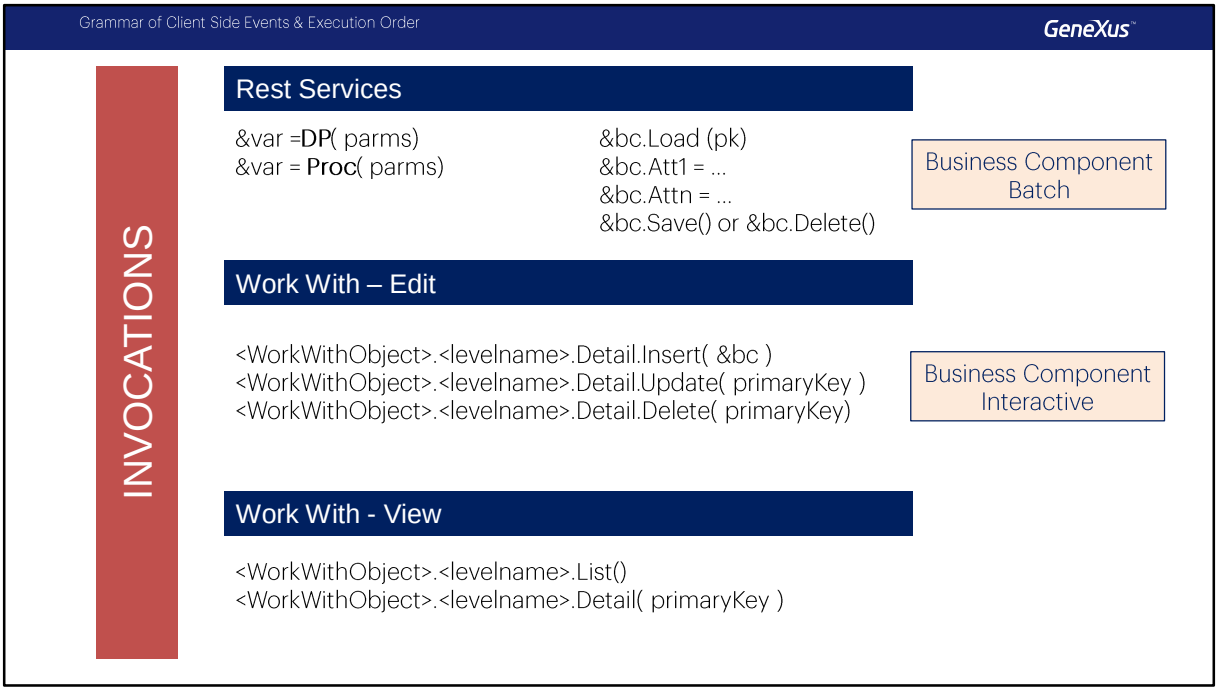
Grammar of Client Side Events & Execution Order

Behavior Development

GeneXus™ 16



Veremos en detalle la gramática de los eventos que se ejecutan en el cliente, esto es, en el dispositivo. Qué se puede hacer, y cómo, luego veremos en detalle el comando Composite y por ultimo estudiaremos el orden de ejecución de los eventos.



Primero que nada, repasaremos las invocaciones que pueden realizarse dentro de un evento del cliente que vimos en el video sobre las invocaciones de objetos en Smart Devices.

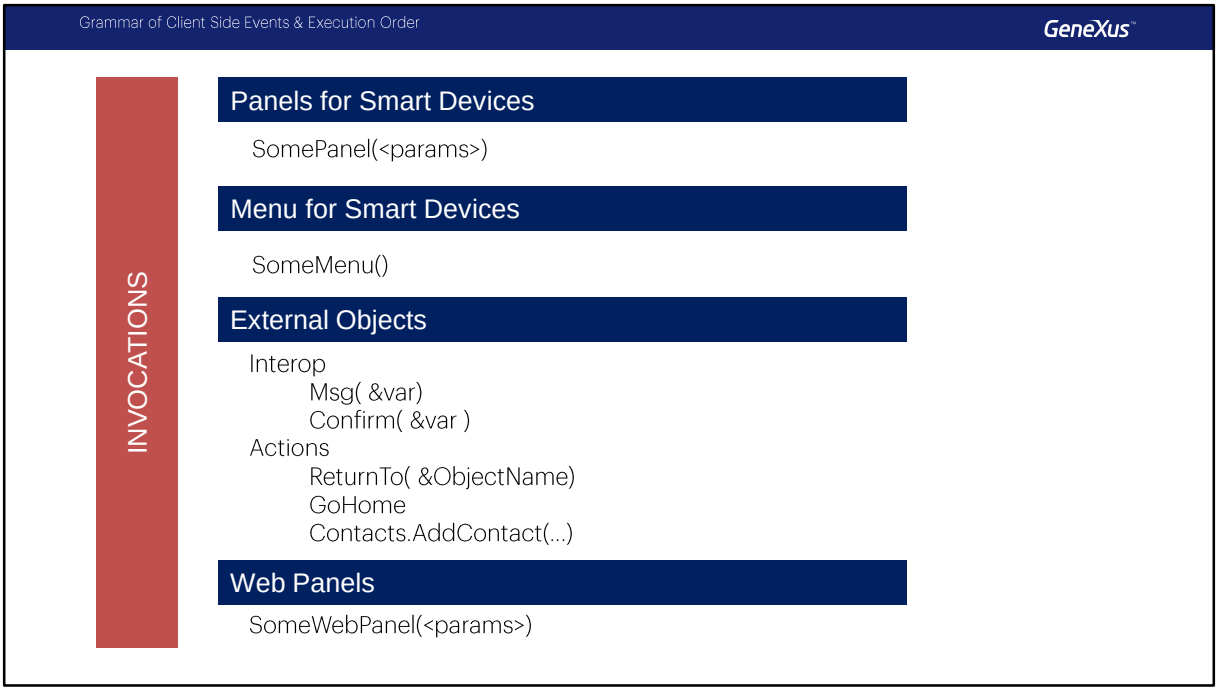
Podemos clasificar las invocaciones en:
Servicios REST del servidor, como Data Providers o procedimientos que nos devuelvan información que cargaremos en una variable en el dispositivo. Necesariamente deben estar expuestos como servicios REST. No podemos llamar a un procedimiento interno desde el dispositivo si estamos en una aplicación de arquitectura online (mas adelante veremos el caso offline).

También podemos querer dentro de un evento del lado del cliente, ingresar un nuevo registro a la base de datos sin tener que pedir información al usuario. Esto se hace como en cualquier otro objeto GeneXus, con los métodos y propiedades del Business Component, a excepción de que aquí también deberá estar expuesto como servicio REST, dado que estamos invocando desde el dispositivo, en una arquitectura online.

Dentro de un evento del lado del cliente, también podemos llamar a la pantalla de Detail del WorkWith, para insertar, actualizar o eliminar (lo que internamente se

traducirá en una invocación al Business Component REST). Aquí, a través de la pantalla, se le piden los datos al usuario y luego se realiza esa operación de manera transparente para el desarrollador.

También podríamos simplemente querer llamar al List o al Detail en modo View.



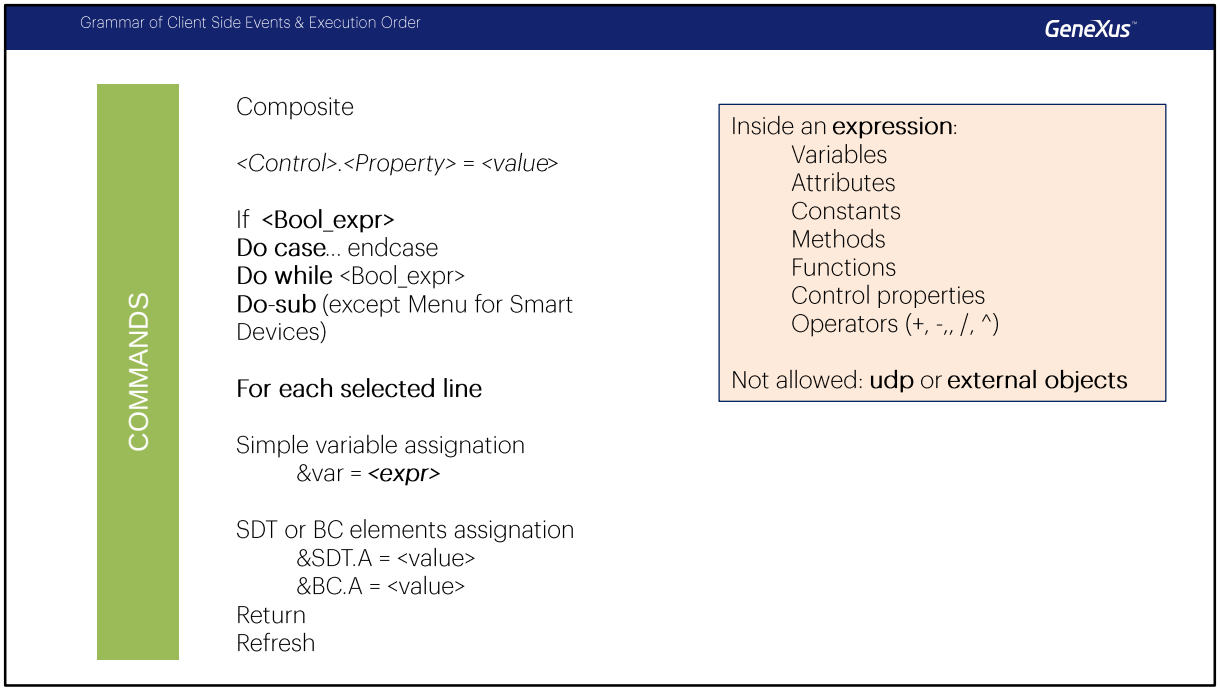
Así como a objetos **Panels for Smart Devices**, que son pantallas un poco más flexibles que las de los work with.

También podemos llamar a un **Menu for Smart Devices**.

…O utilizar algunas de las funcionalidades provistas por las **apis**, como desplegar un mensaje en la pantalla, pedir confirmación al usuario para continuar, retornar al objeto indicado, o al objeto main de la app, agregar un contacto a la app de contactos del dispositivo, etcétera.

También se puede invocar a un **web panel**. Éste se abrirá en el navegador del dispositivo conservando la application y status bar de nuestra app (siempre que el navegador sea Chrome para Android y Safari para iOS. De lo contrario, abrirá el navegador default al que se le quita el marco, para que luzca más parecido al resto de la aplicación).

Todo esto en cuanto a las invocaciones.



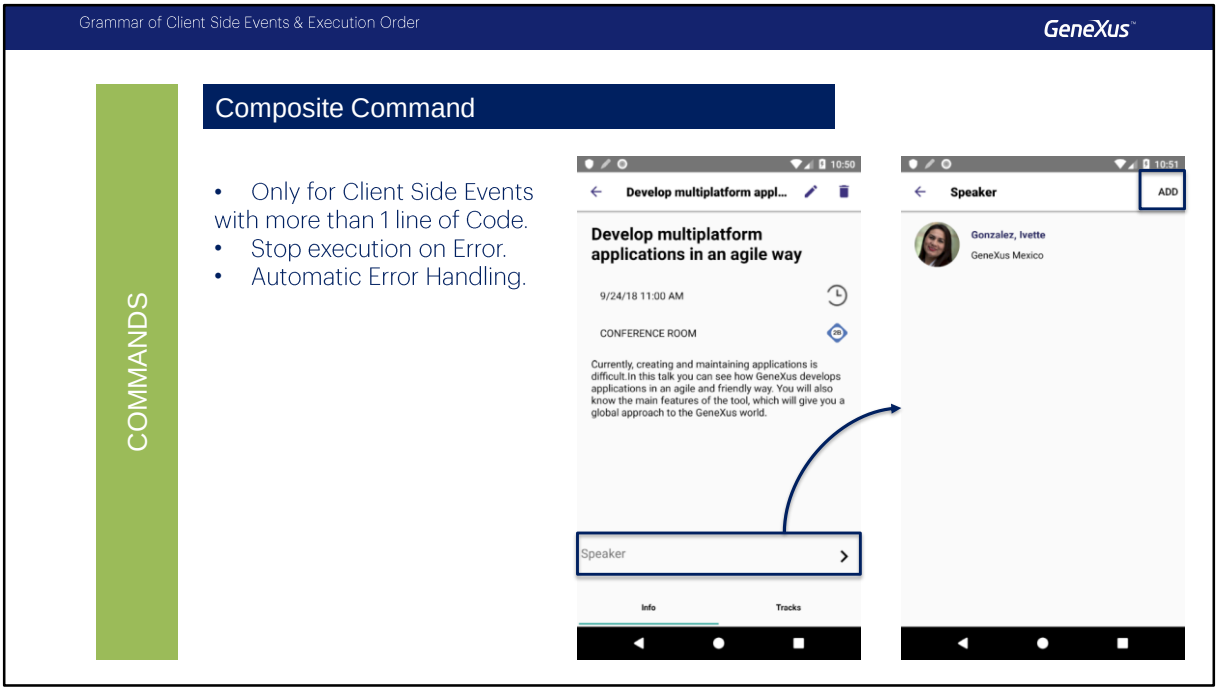
Los **comandos** aceptados por el momento son los que se muestran.

Por ejemplo, puede hacerse visible o invisible un control, se le puede configurar la clase, se pueden utilizar las estructuras de control **if**, **do case**, y **do while** (por ahora no están incluidas las **for in**, ni los métodos, como el **Add**, de los SDTs o BCs).

Estas estructuras aceptan expresiones booleanas que pueden incluir todo lo conocido, excepto invocaciones (udps, o métodos de external objects). También pueden utilizarse invocaciones a subrutinas (do-sub) excepto en objeto Menu for Smart Devices.
En el comando **for each selected line** sólo se puede invocar a un procedimiento (si es online se invoca una vez y es en el server en el que se ejecuta n veces –una vez por cada línea seleccionada- y se devuelve el resultado final todo como un servicio REST transparente para el desarrollador).

En las asignaciones, a variable simple se le puede asignar una expresión, o una invocación a un proc que devuelve el valor, pero a un elemento de SDT o BC sólo se le puede asignar un valor, no una expresión.

También tenemos como comandos el Return y el Refresh.

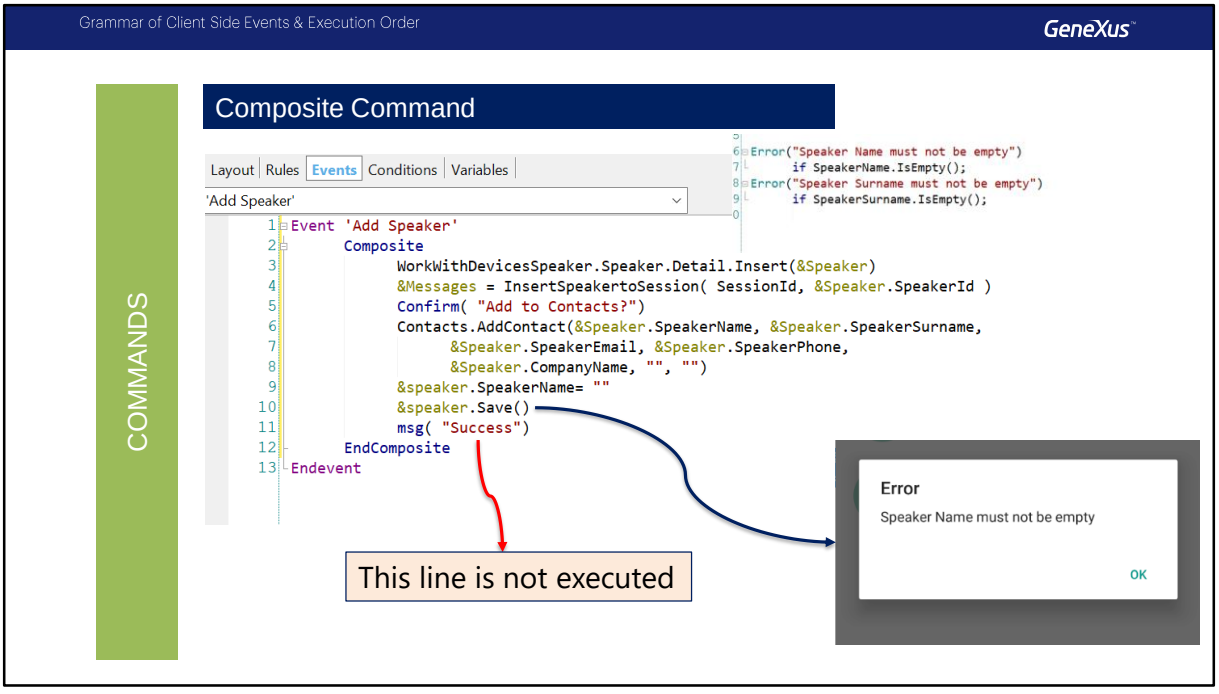


Respecto al Comando Composite, que habíamos visto en varios videos.

Este evento solo se utiliza en eventos del lado del cliente en Smart Devices, cuando deben realizarse un par de invocaciones o más en un evento, y en este caso es obligatorio agrupar el código completo del evento dentro de este comando.
La importancia de este comando es que, cuando ocurre un error en la secuencia de llamadas, la ejecución se detiene y se manejan los errores automáticamente, desplegándolos en la pantalla sin tener que implementar ninguna programación.

Lo veremos a través de un ejemplo.

Supongamos que entre los oradores de una conferencia, queremos permitir agregar uno nuevo, insertándolo en la base de datos y luego asociándoselo a esta conferencia.
Entonces al acceder a la pantalla de oradores de una conferencia (sesión Speaker del WorkWithDevicesSession) agregamos un botón Add para realizar esta operación.



Aquí vemos entonces el evento asociado al botón “Add”, sus comandos los rodeamos con el bloque Composite-EndComposite. ¿Por qué?

Porque tenemos más de un comando dentro del evento. Es más, podemos agregar muchos más comandos dentro de ese mismo evento.

Observemos que es lo que programamos en este bloque Composite-EndComposite.

Lo primero que hacemos es invocar al Detail de oradores, en modo Insert, pasándole como parámetro una variable Business Component de tipo Speaker, para que nos devuelva allí cargados los datos del orador que el usuario acaba de insertar en el Detail, al volver de la invocación.

Luego, llamamos a un procedimiento que, pasándole el id de la conferencia en la que estamos posicionados y el Id del speaker recién insertado, agregue ese orador a esa conferencia (en la tabla correspondiente al nivel Session.Speakers).

El procedimiento tendrá como parámetro de salida una variable del tipo de datos el SDT predefinido Messages (devuelta automáticamente por todo Business Component cuando usamos su método GetMessages()). Dentro del proceso la cargaremos con los mensajes de error o advertencias que nos interesen.

De este modo, a la vuelta se inspecciona automáticamente esa variable y en caso de warnings se despliegan los mensajes correspondientes y se continúa con la ejecución, y en caso de error se despliega también el mensaje pero se detiene la ejecución.

Este procedimiento deberá estar expuesto como servicio Rest, ya que es llamado desde un evento del lado del Cliente.

Si no hubo errores, se ejecuta el siguiente comando.

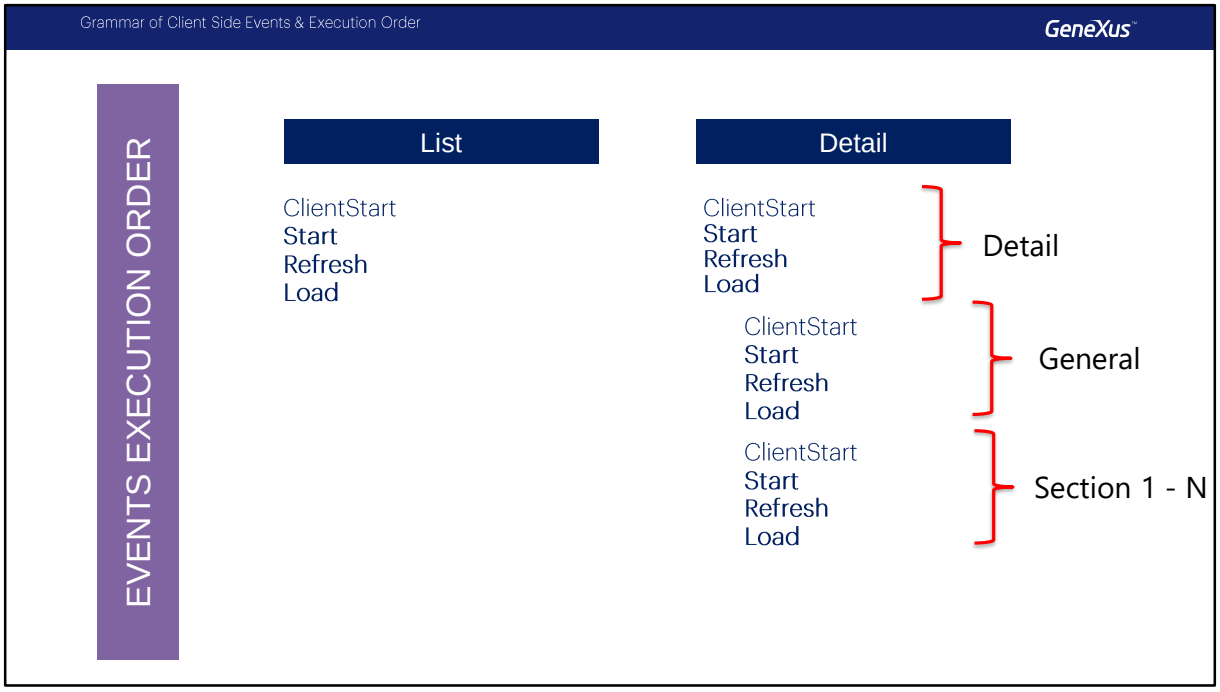
El método Confirm() de la api Interop, despliega al usuario el mensaje especificado y ofrece la posibilidad de continuar con la siguiente ejecución (presionando 'OK') o de detenerse allí (presionando 'Cancel'). Si presiona 'Ok' entonces la siguiente invocación sí se realiza, y se agrega el contacto en la agenda de contactos del dispositivo.

Otra vez, si esta operación es exitosa se pasa a la siguiente sentencia, donde, en este ejemplo, estamos dejando vacío el Speaker Name del business component Speaker e intentamos Salvar.

Esto va a arrojar un error, puesto que en la transacción de Speakers tenemos programada la regla Error para no dejar el nombre del orador vacío, por tanto veremos cómo se desplegará el mensaje de error correspondiente a la regla. O sea, esto lo agregamos para forzar que se ejecute la regla de error del Business Component.

El mensaje Success no se mostrará, puesto que la ejecución se interrumpió aquí.

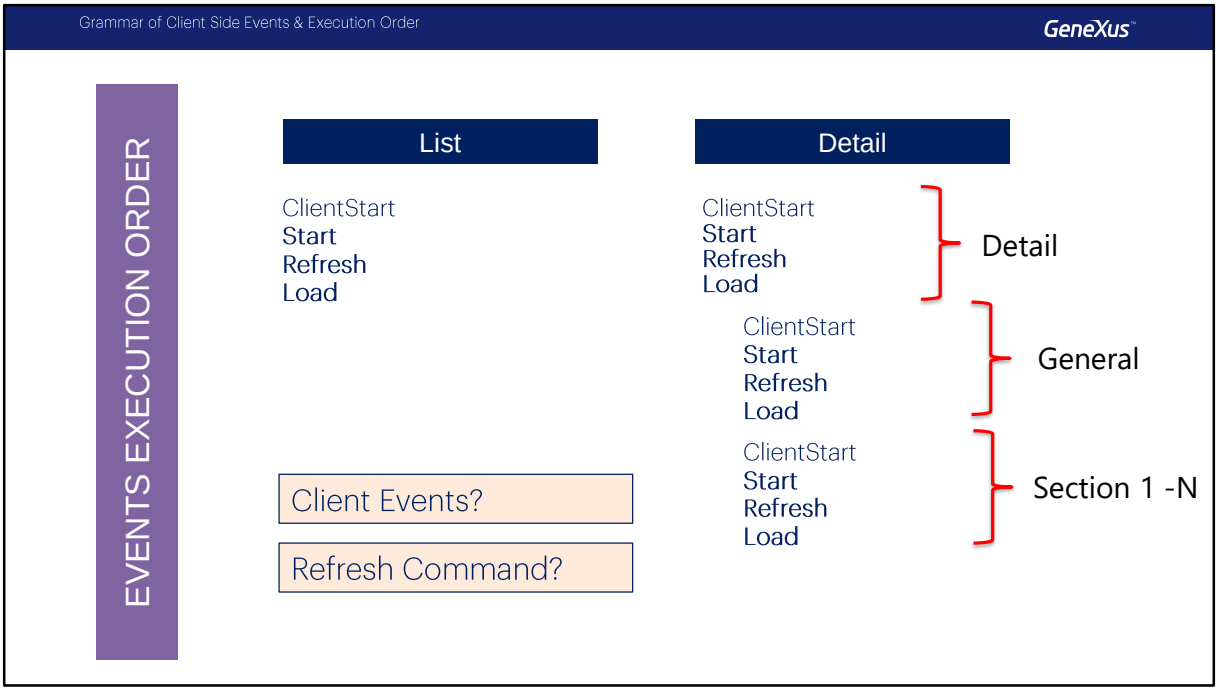
Esta es una gran diferencia con las aplicaciones Web, dado que en éstas cuando dentro de un evento un objeto llamado produce un error, no se interrumpe la ejecución, sigue en la sentencia siguiente y es el desarrollador quien debe encargarse de manejar los errores y programar las acciones a tomar.
El comando Composite, en cambio, es el que hace que cuando ocurra un error en una secuencia de llamadas se detenga la ejecución y se manejen los errores automáticamente, desplegándolos en la pantalla sin tener que implementar ninguna programación.



Pasemos ahora a ver el último punto dentro de los eventos, que es el orden y el momento en que se ejecutan.
Veremos brevemente el orden de ejecución del List y del Detail cuando se cargan por primera vez, y luego veremos qué pasa cuando se dispara un evento del cliente, con estos objetos ya abiertos.
Para los Panels for Smart Devices valdrá lo mismo que para el List. Y para los que tengan SD components, va a valer lo mismo que para el Detail.

- Si estamos abriendo un List, se van a ejecutar en este orden: el evento ClientStart, en el cliente, y luego, los eventos Start, Refresh y Load, en el servidor, si es que estamos dentro de una arquitectura online.
- Si el objeto abierto es el Detail, entonces se van a ejecutar en ese orden: el ClientStart del Detail, seguido del Start, Refresh y Load del propio Detail, y luego, por cada Section, empezando por la General, se van a ejecutar esos mismos eventos.

Lo mismo sucederá si se trata de un panel con SD components.



Luego de cargada la pantalla, sea esta de List o de Detail, el usuario está listo para interactuar.
¿Qué sucede cuando el usuario dispara una acción?

El código asociado es ejecutado del lado del cliente, esto es, en el dispositivo. Si se encuentra una llamada a un procedimiento rest, data provider o business component, se realiza la llamada y se queda a la espera del resultado, para seguir ejecutando el resto del código en el cliente. Es decir, se hace esa invocación al servicio REST, y cuando el servicio responde con el JSON se ejecuta lo que sigue.

Obsérvese que en cualquier caso los eventos del sistema: Start, Refresh, Load no son ejecutados.
A menos que se encuentre un comando Refresh dentro del propio código del evento de usuario.
¿Qué pasa en este caso con el Comando Refresh?

Se va al Server, si no hay cambios en los datos no se trae nada al cliente. De lo contrario, los eventos Refresh y Load de todo grid no basado en un SDT son ejecutados.

Con esto terminamos el tema.

GeneXus™

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