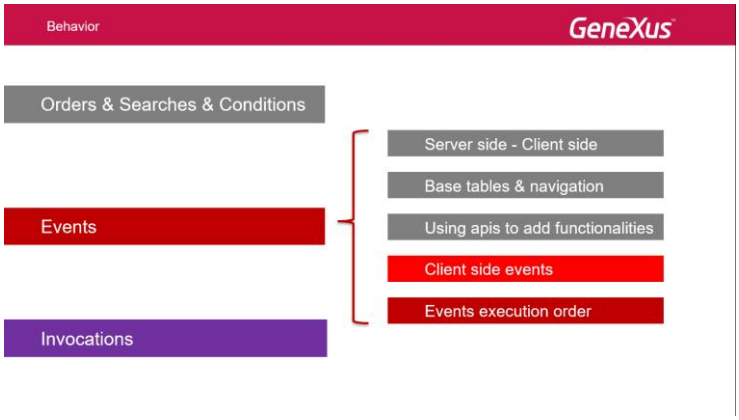
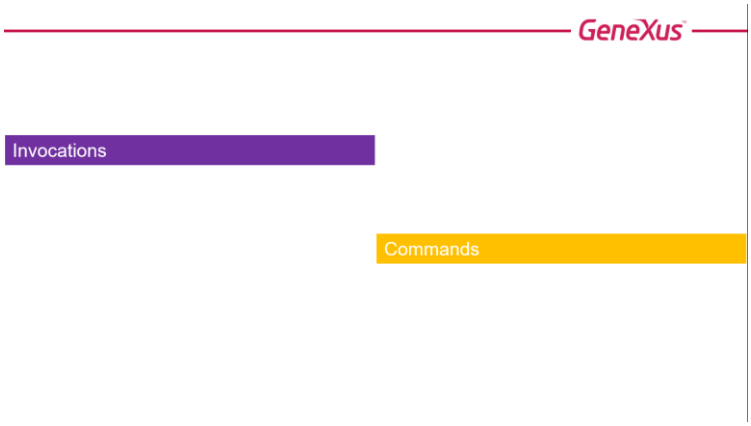
Grammar of Client Side Events



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.

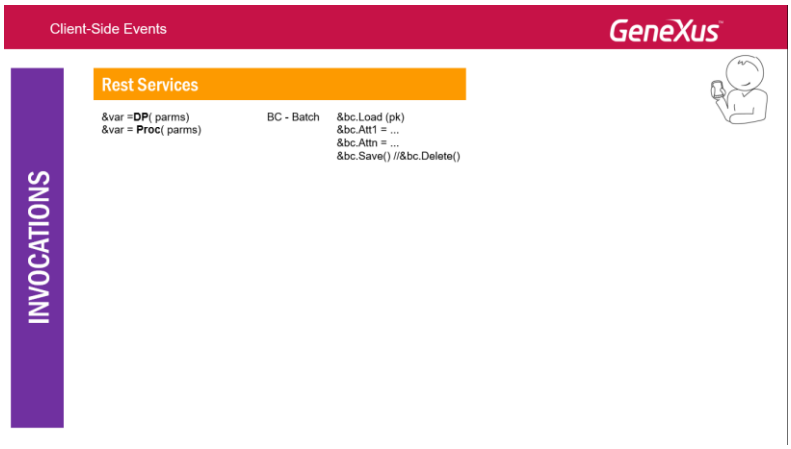
Primero que nada, repasaremos las invocaciones que pueden realizarse dentro de un evento del cliente y luego veremos los otros comandos, recordando lo que habíamos dicho antes: que la gramática de estos eventos es reducida respecto a los eventos que se ejecutan en el Server (que eran, recordemos, el Start, Refresh y Load).

Sobre las invocaciones, volveremos luego para estudiar las CallOptions que permiten indicar algunos aspectos de la invocación a ser ejecutada inmediatamente.



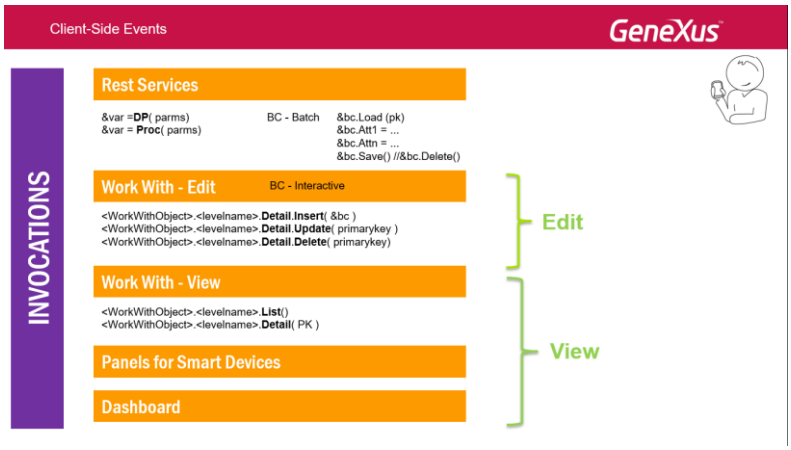
Podemos clasificar las invocaciones en: **Servicios rest** del servidor, como data providers o procedimientos que nos devuelvan información que cargaremos en una variable en el dispositivo. Necesariamente deben estar expuestos como servicios rest. No podemos llamar a un procedimiento interno desde el dispositivo si estamos en una aplicación de arquitectura online (al final del curso 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.



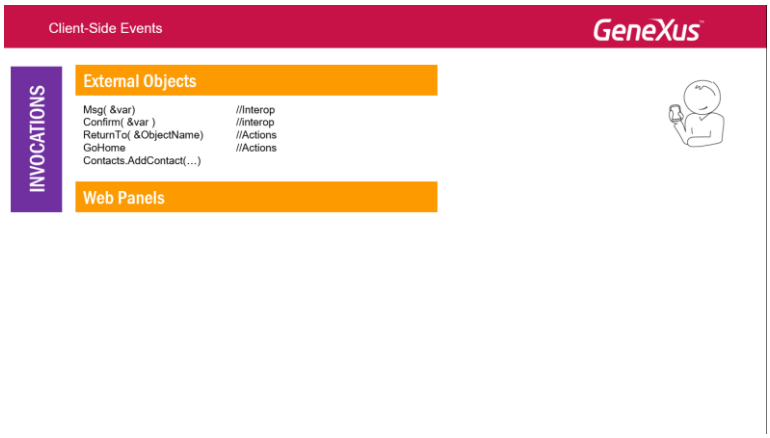
Dentro de un evento del lado del cliente, también podemos llamar a la pantalla de Detail del WorkWith, para insertar, actualizar o eliminar (lo que internamente se traducirá en una invocación al BC rest). Aquí, a través de la pantalla, se le piden los datos al usuario y luego se realiza esa operación 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, y también podemos llamar a un **Dashboard**.



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 chorme 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).

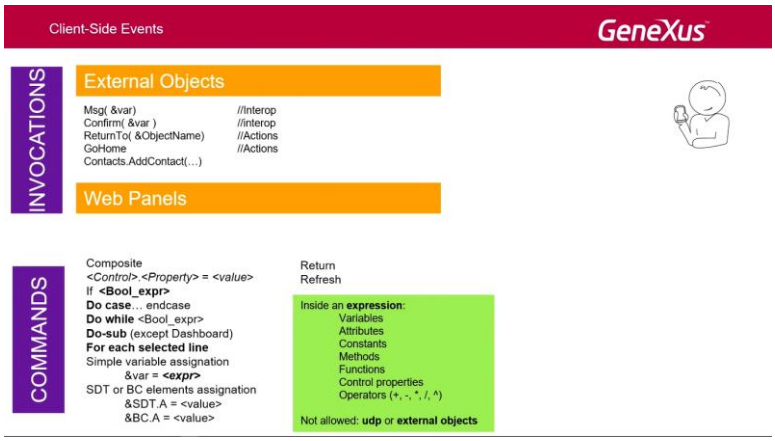


Eso en cuanto a las invocaciones. Los **comandos** aceptados por el momento son los que se muestran.

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, esos no son aceptados). También pueden utilizarse invocaciones a subrutinas (do-sub) excepto en el objeto dashboard.

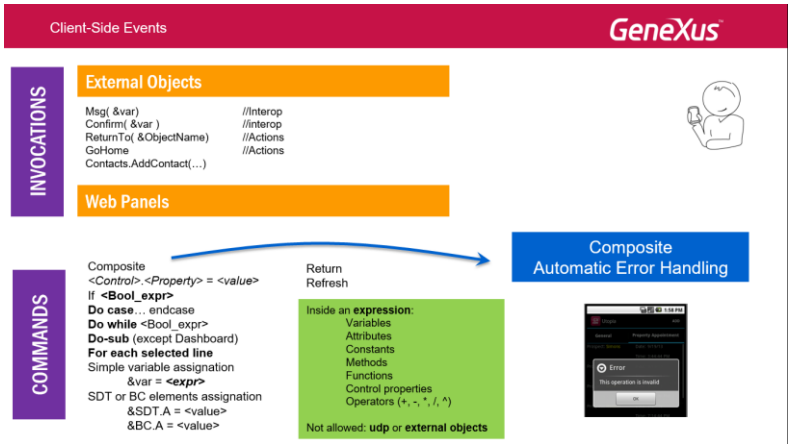
En el comando **for each selected line** sólo se puede invocar a un proc (si es online se invoca una vez y es en el server en el que se ejecuta n veces –una vez por cada línea seleccionada- y se devuelve el resultado final en un json).

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**, recordemos lo que ya habíamos adelantado en otros videos: cuando deben realizarse un par de invocaciones o más en un evento, es obligatorio agrupar el código completo del evento dentro de este comando. De este modo, 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

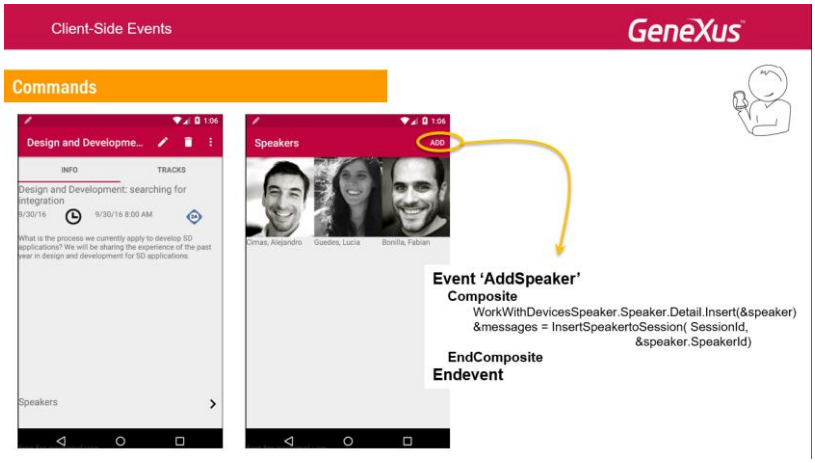
programación. Este comando está implementado sólo en Smart Devices y es obligatorio en éstos.



Vamos a ver un ejemplo, donde codificamos un evento del cliente que requerirá utilizar este comando.

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. 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.

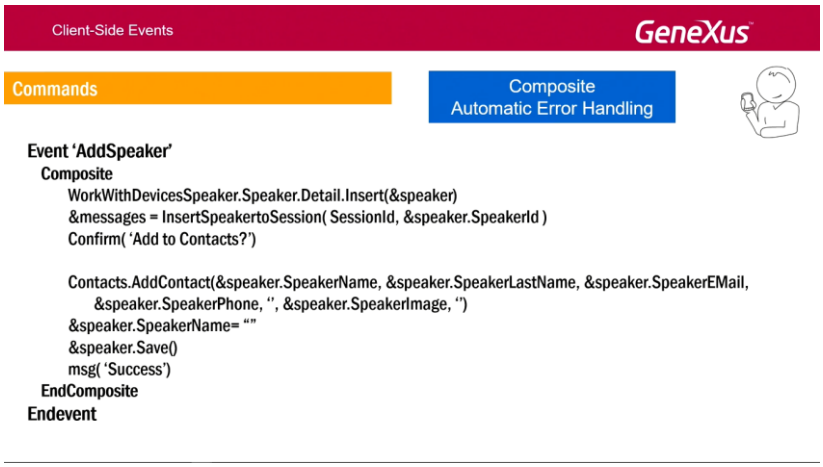
Es más, podemos agregar muchos más comandos dentro de ese mismo evento.



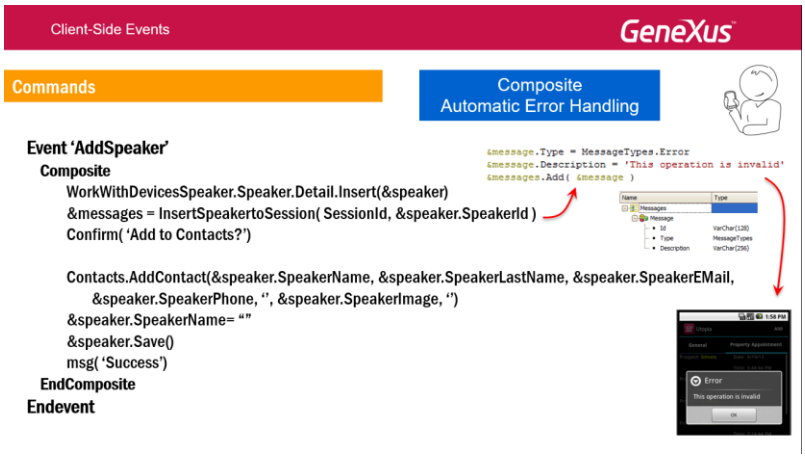
Observemos que es lo que programamos en este bloque Composite-EndComposite.

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).



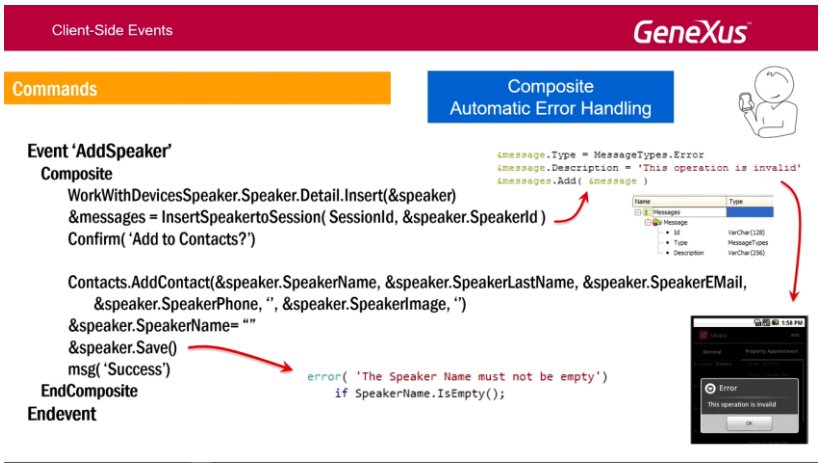
El procedimiento tendrá como parámetro de salida una variable del tipo de datos del SDT predefinido Messages (que recordemos devuelta automáticamente por todo BC cuando usamos su método GetMessages()). Dentro del procedimiento la cargaremos con los mensajes de error o advertencias que nos interesan. 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, y no se sigue ejecutando lo que viene debajo. Este procedimiento deberá estar expuesto como servicio Rest. Piensen porqué.



Si no hubo errores, se ejecuta el siguiente comando. El método Confirm de la API Interop (recordemos que no es necesario en este caso escribir Interop.Confirm()). Este despliega al usuario el mensaje especificado entre paréntesis 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 si vamos a la transacción de Speakers vemos que teníamos 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. El mensaje

Success, en ese caso, no se mostrará, puesto que la ejecución se interrumpió aquí, antes, en el Save. Si no existiera esa regla de error, entonces sí se salvaría con éxito, y se desplegaría el mensaje Success.



Podemos ver, en definitiva, la gran diferencia con las aplicaciones Web, ya que en este tipo de aplicaciones, las web, cuando dentro de un evento un objeto llamado producía un error, no se interrumpía la ejecución, seguía en la sentencia siguiente y era el desarrollador quien debía encargarse de manejar los errores y programar las acciones a tomar en ese caso.

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, nada en absoluto.

Pasemos ahora a ver el último punto dentro de los eventos, que es el orden y el momento en que se ejecutan.