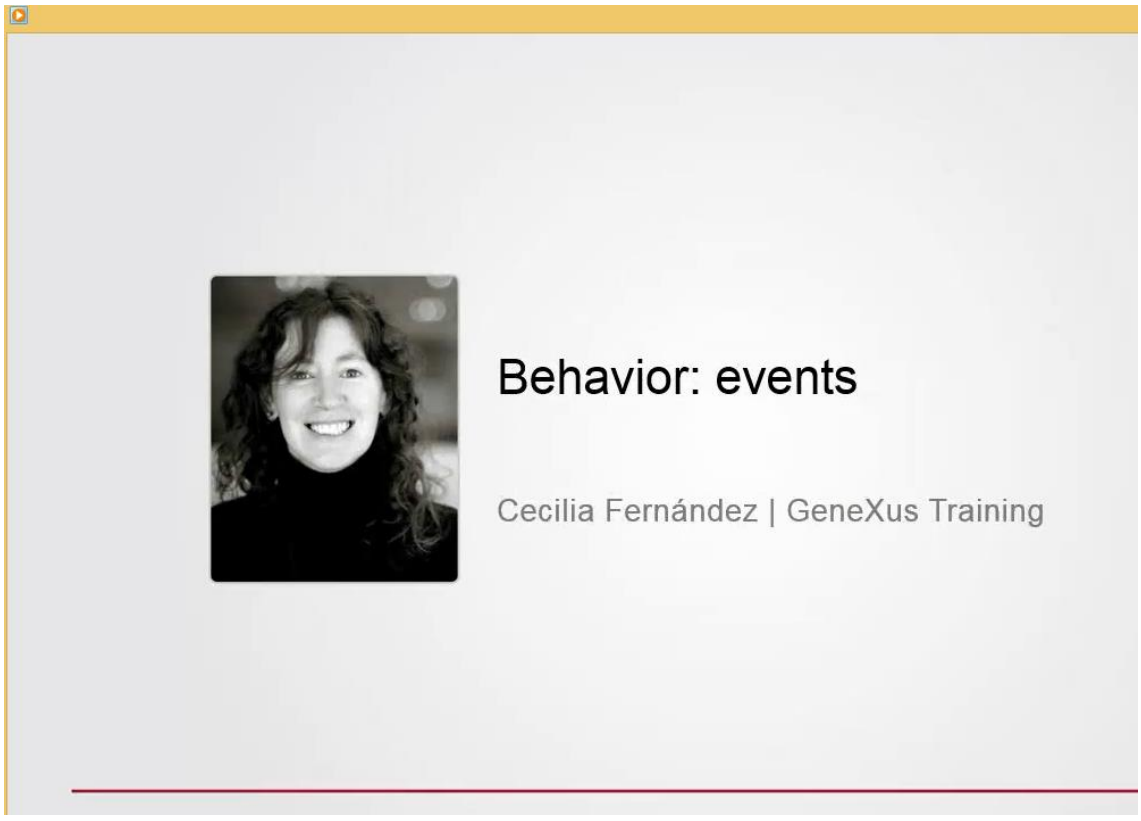
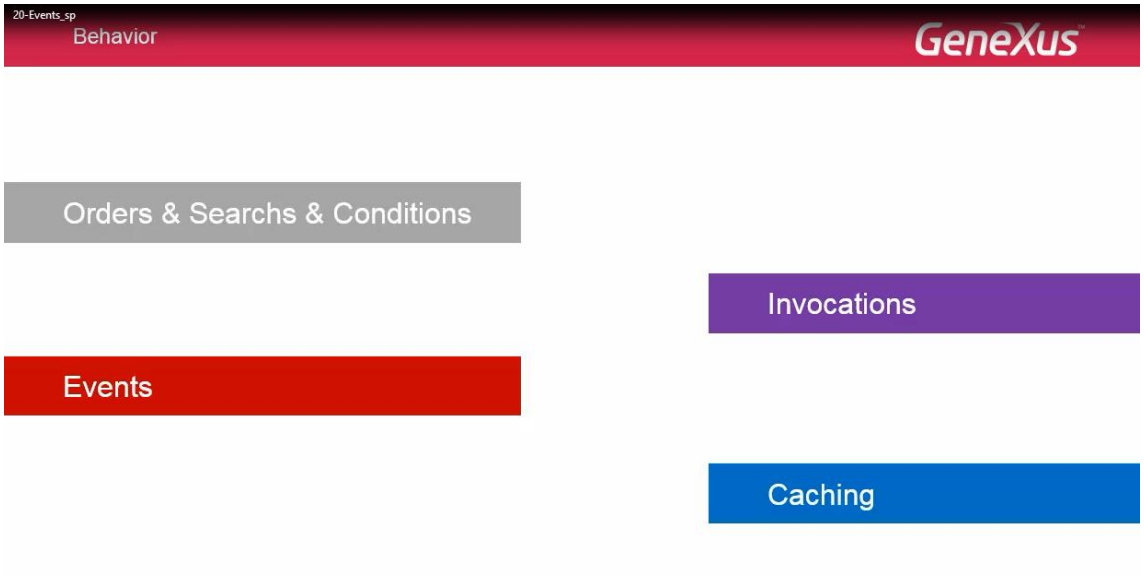
Events

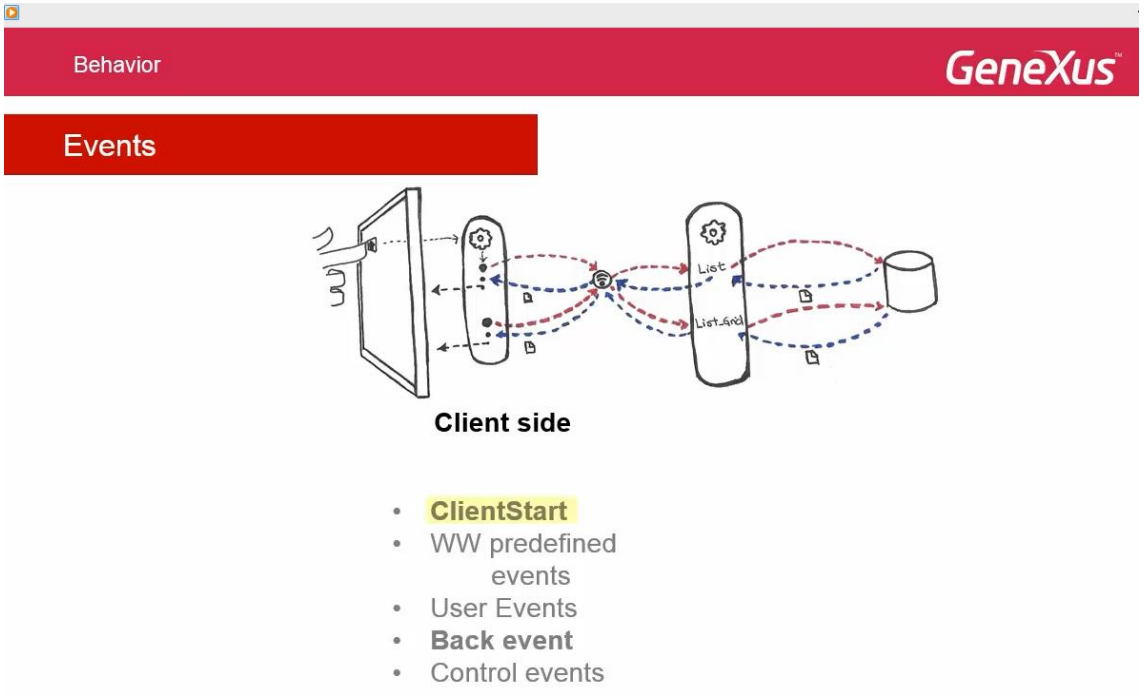

GeneXus™
TRAINING PROGRAM


Behavior: events
Cecilia Fernández | GeneXus Training

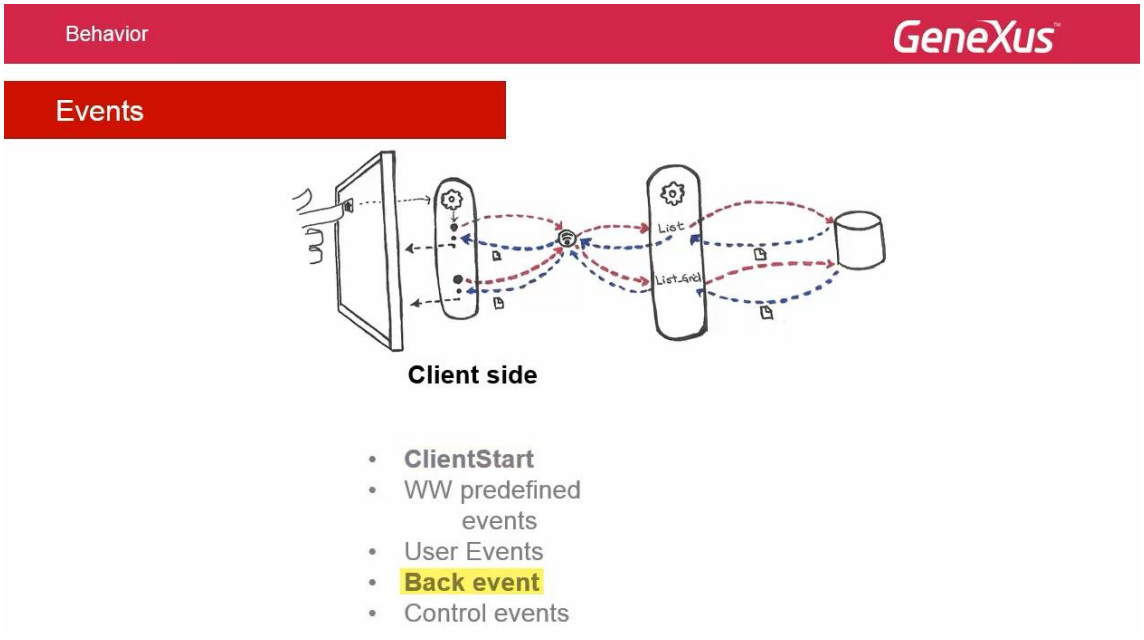


En este video empezaremos a estudiar los eventos que podemos definir a nivel de los objetos Smart Devices, en el contexto de una aplicación online, es decir, que requiere conectividad.

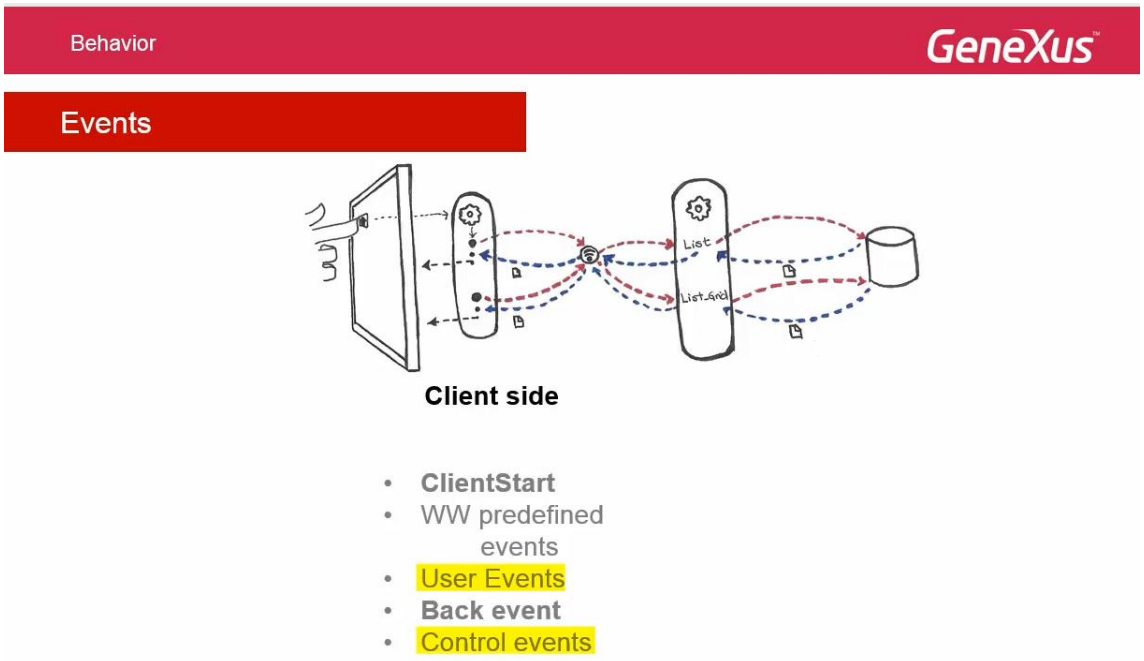
Se mencionarán no obstante, a modo de adelanto, los puntos en los que una aplicación offline difiere de lo aquí estudiado.



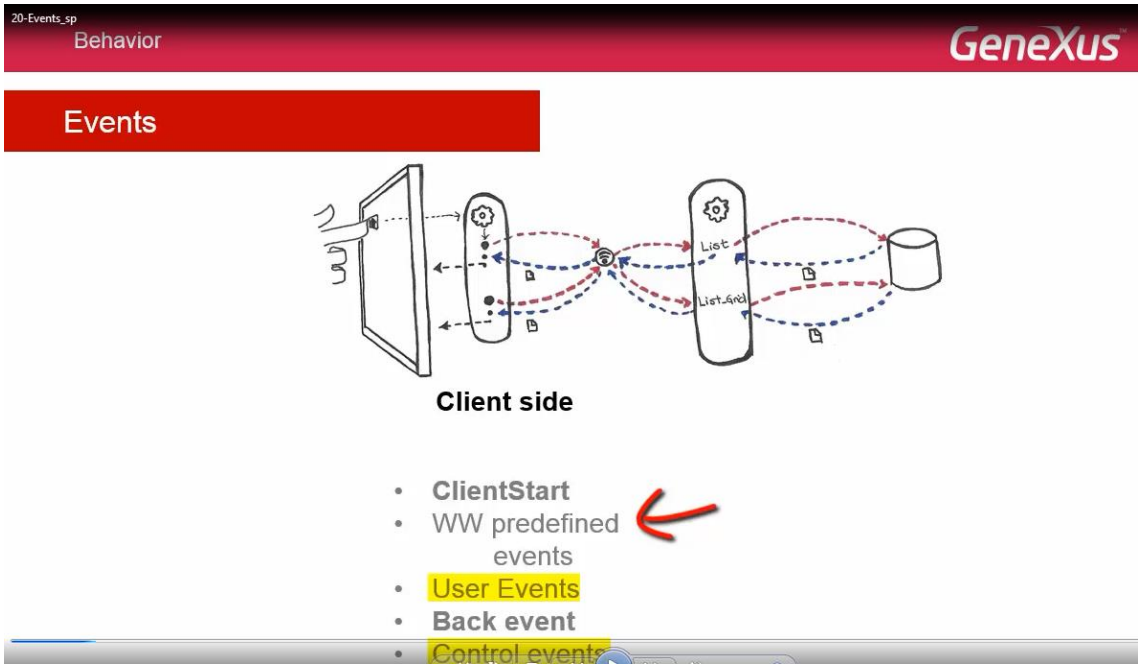
Tenemos eventos que se ejecutarán en el cliente, eventos del sistema, como el nuevo ClientStart o el Back



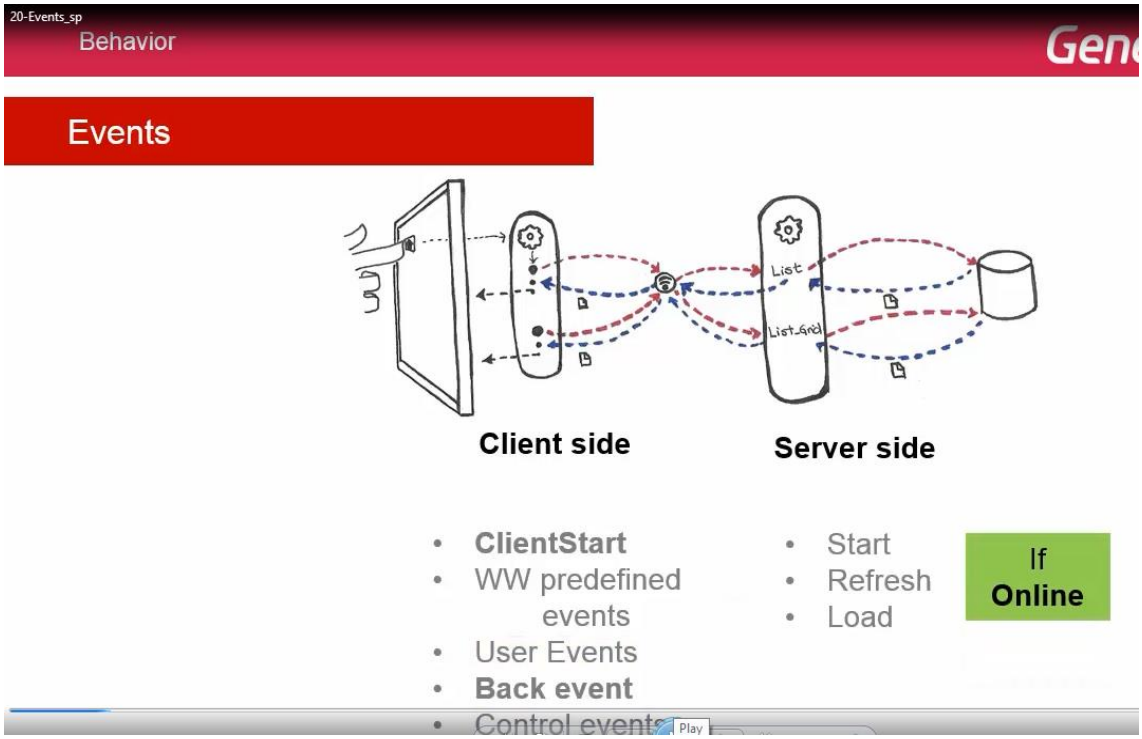
Y de los otros: los eventos de usuario y asociados a controles



entre los que se encuentran los predefinidos por el pattern work with



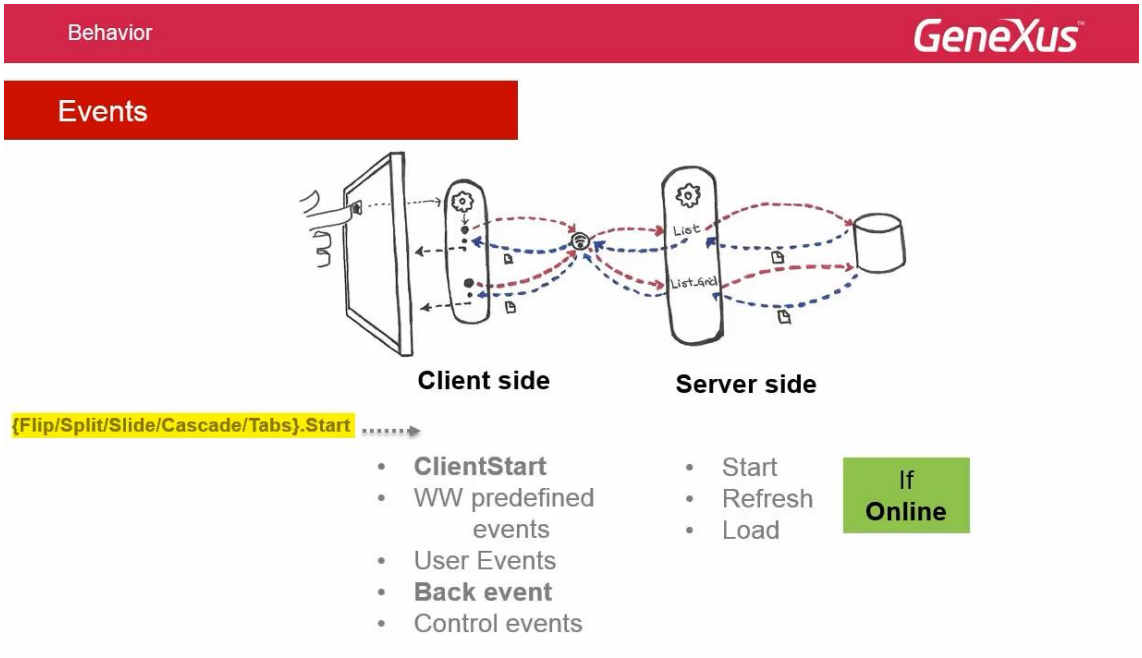
Y tenemos los eventos del sistema ya conocidos: Start, Refresh y Load, que se ejecutarán en el servidor



siempre y cuando se trate de un objeto que requiera conexión olnline.

Luego veremos qué sucede con estos eventos en el caso de arquitectura offline… pero podemos adelantar que se ejecutarán en el cliente.

Cuando se trate del objeto main de la aplicación, también tendremos los nuevos eventos



Flip.Start

Split.Start

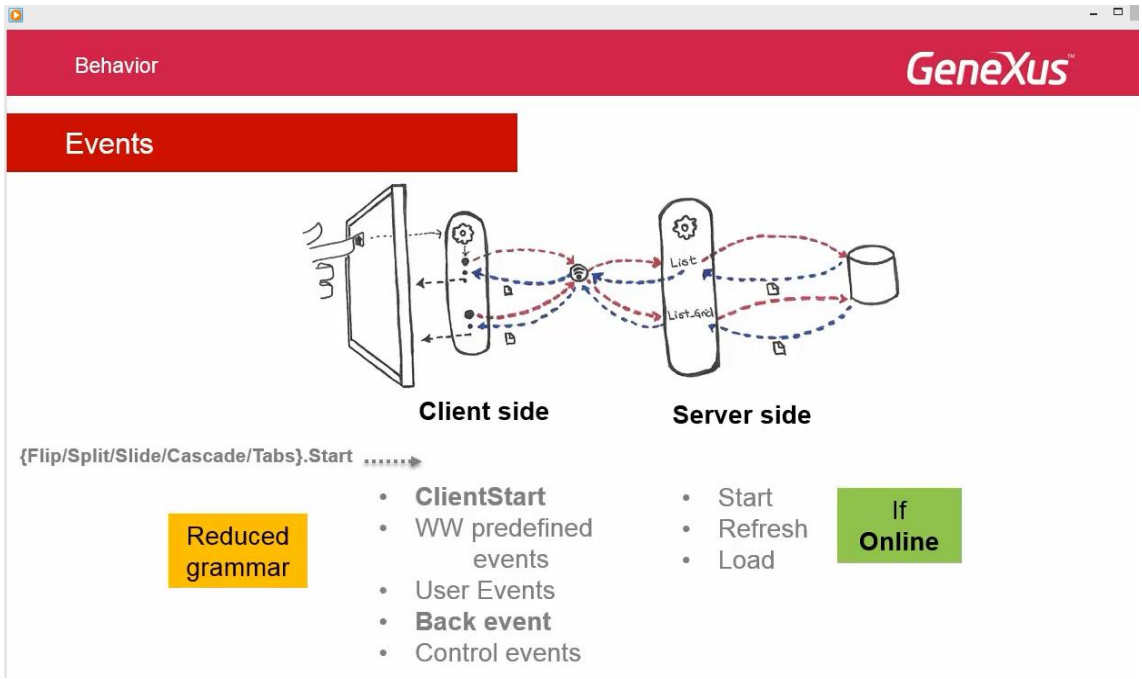
Slide.Start

Cascade.Start

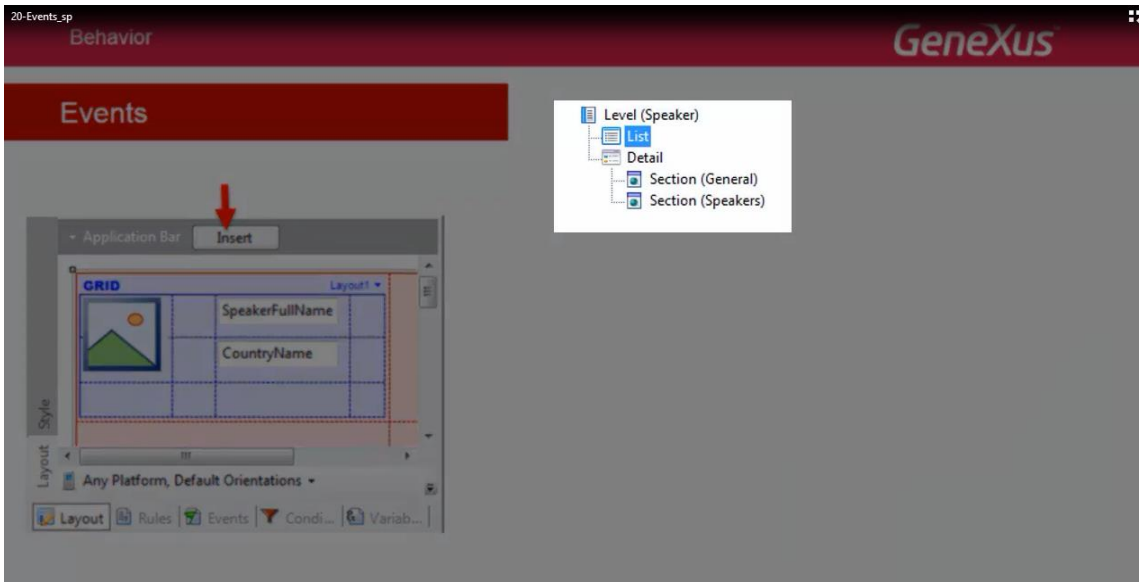
Y Tabs.Start

que vimos cuando estudiamos el navigation style de la aplicación. Sólo son válidos para el objeto principal, sea este un Dashboard, o no.

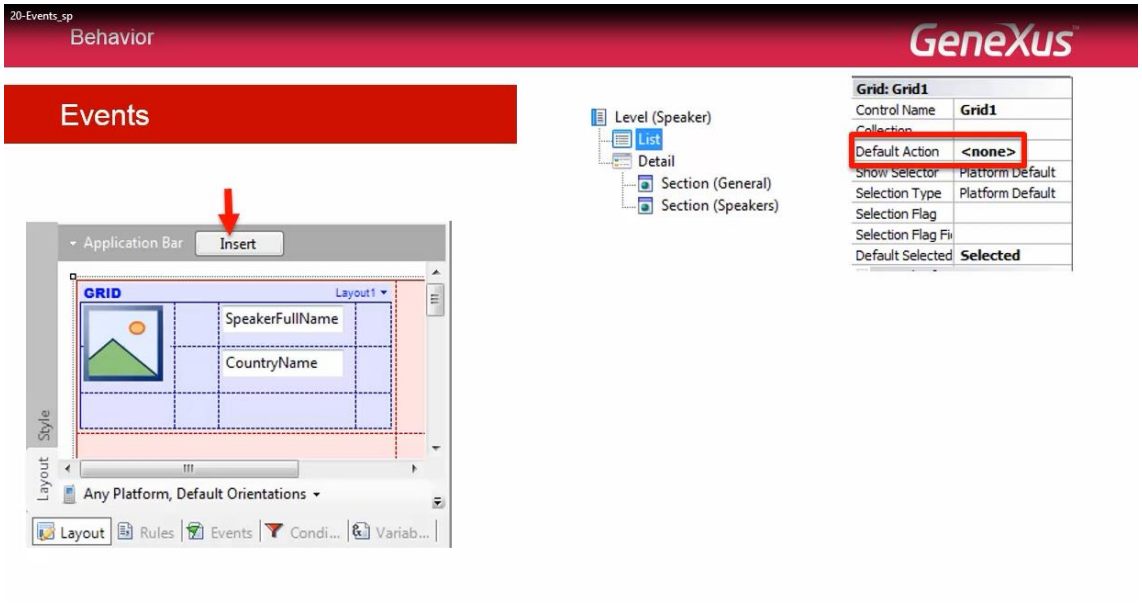
Los eventos ejecutados en el cliente, tendrán una gramática reducida respecto a la de los eventos en el server.. gramática que veremos más adelante en otro video



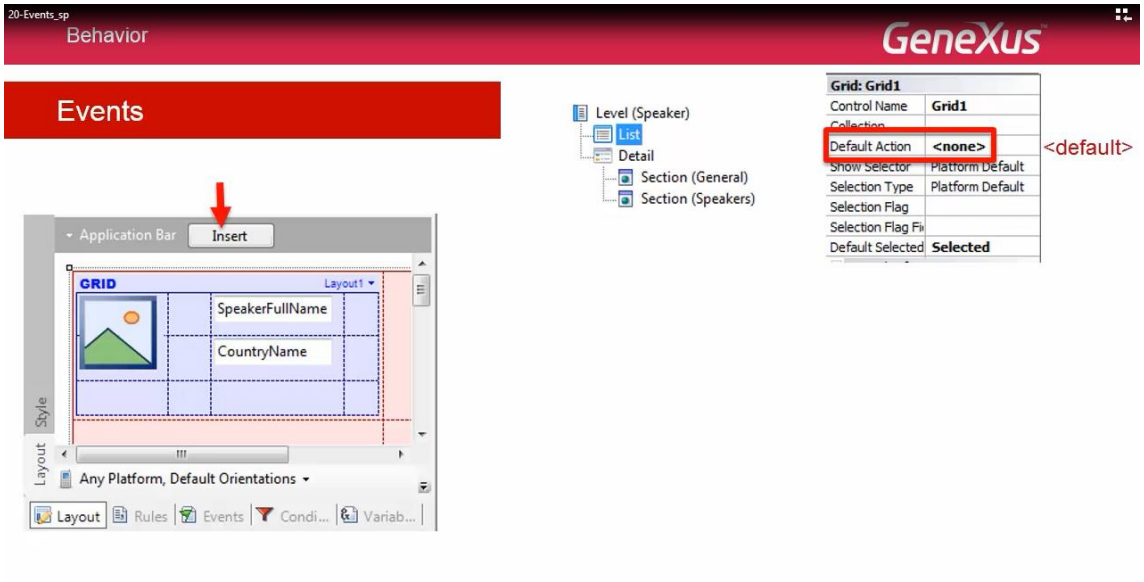
En el ejemplo, el evento Insert en el List



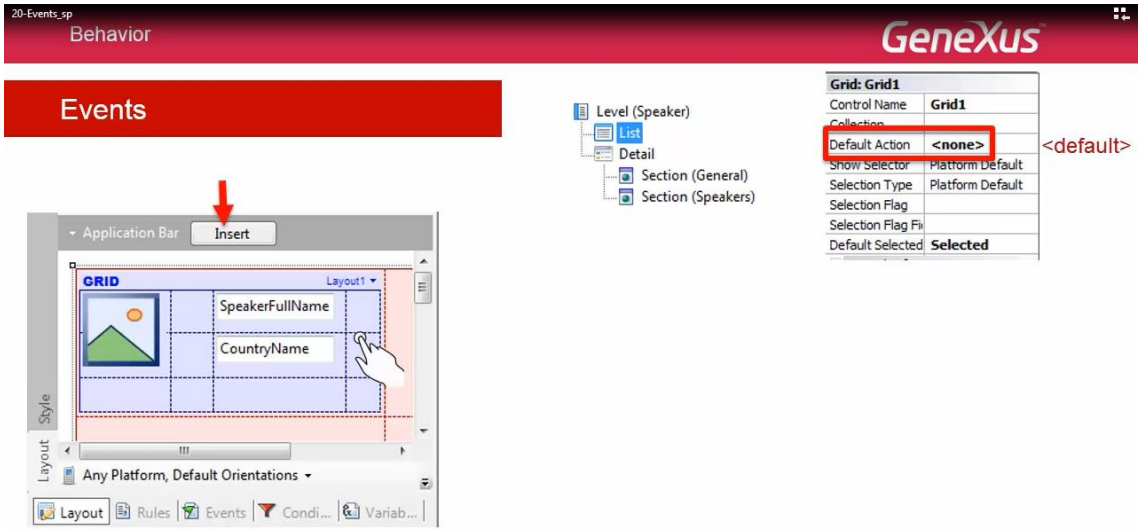
es un evento creado por el pattern



También por cuenta del pattern, corría asociar a la propiedad Default Action, el valor: default

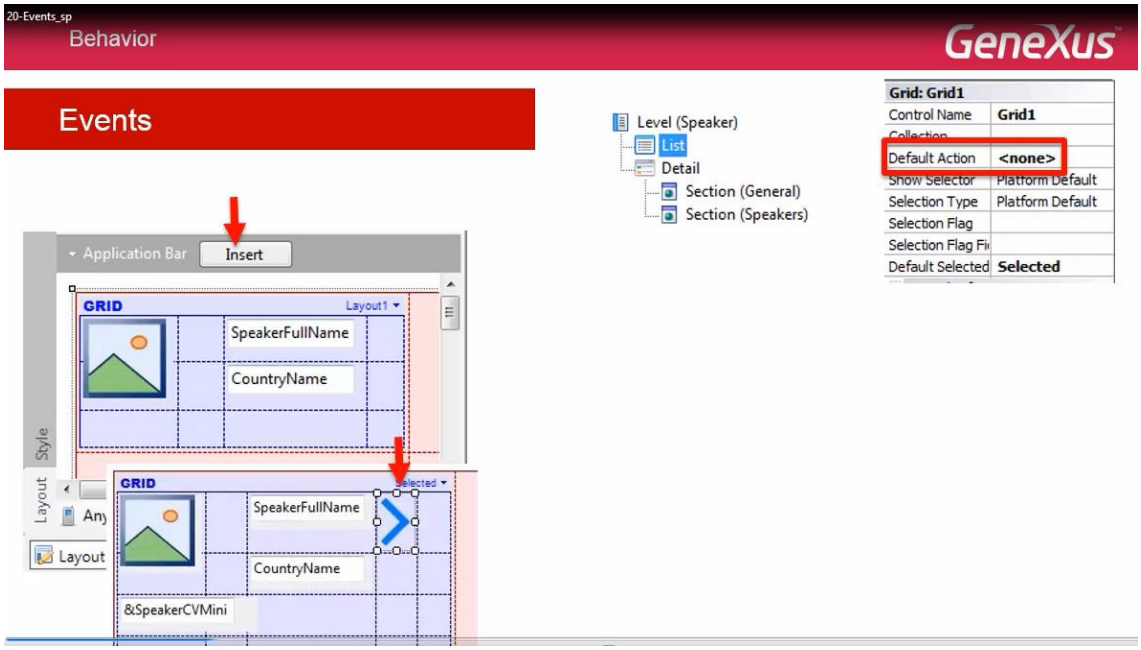


que indicaba aunque no viéramos un evento asociado, que se llamaba al Detail en modo View cuando el usuario hacía TAP

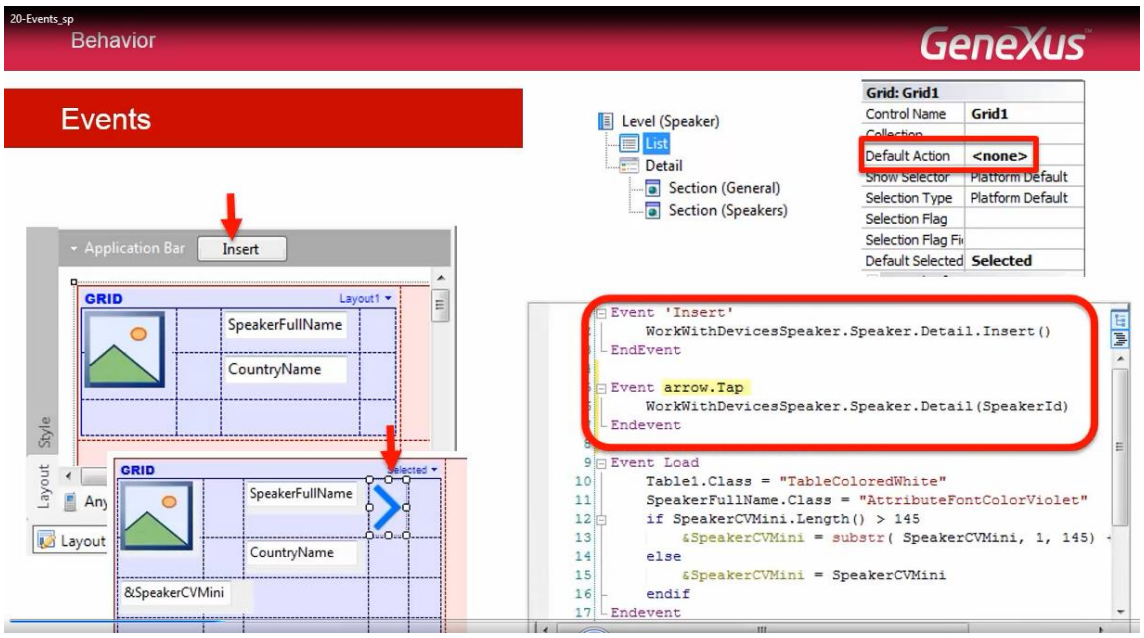


sobre una línea del grid.

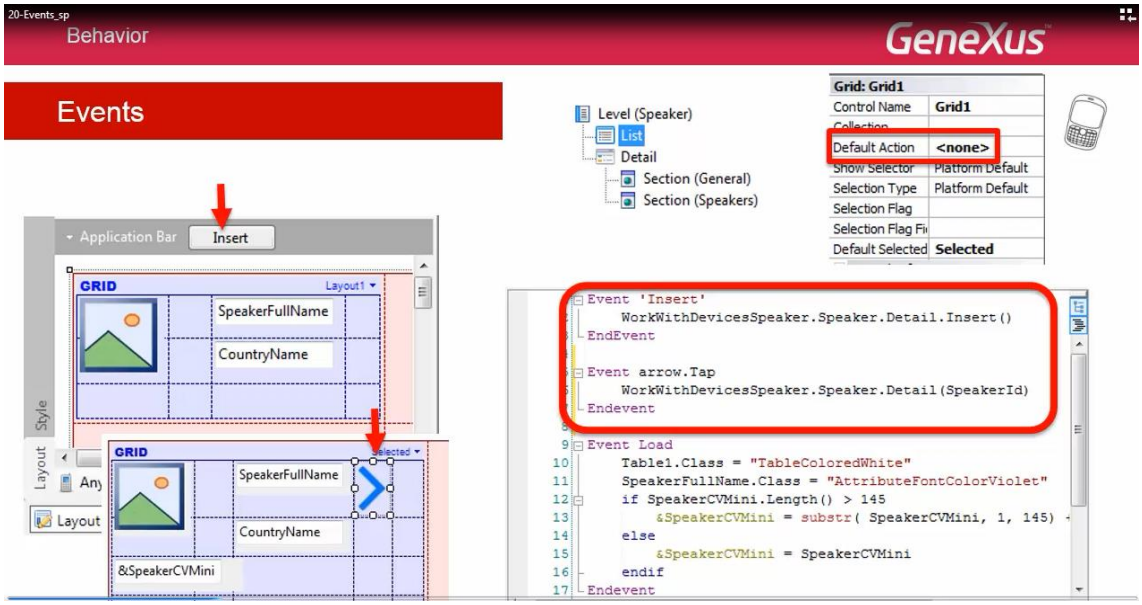
Aquí cambiamos este comportamiento pues lo programamos nosotros a través de uno de los eventos de los controles



que tienen que ver con la lógica “touch screen”

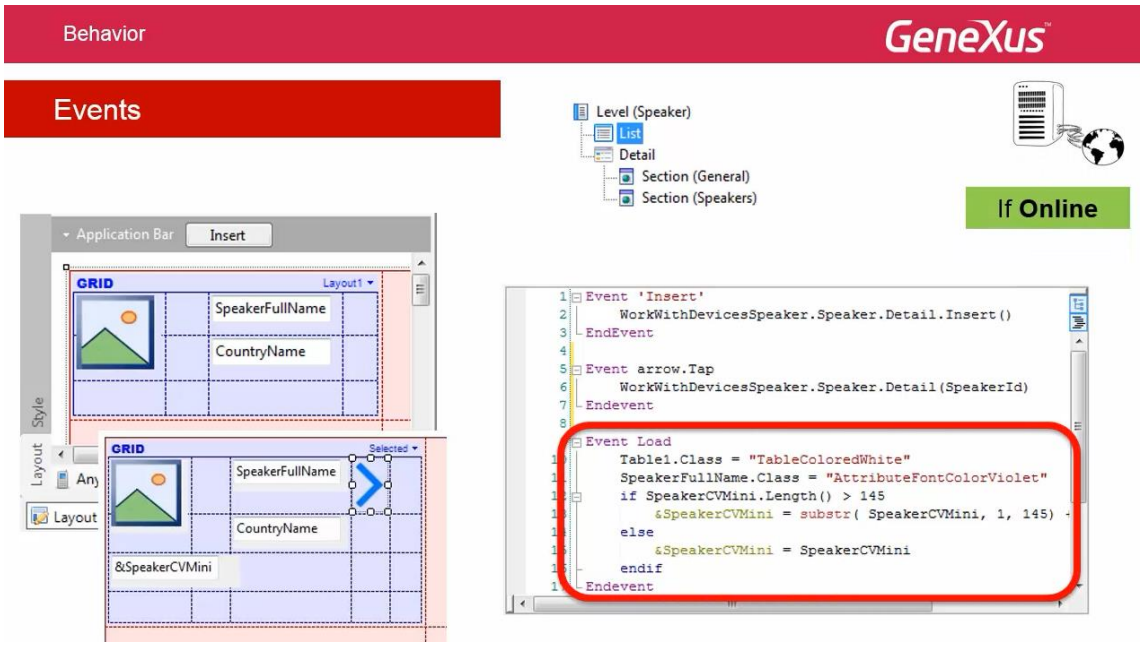


el evento Tap sobre la imagen de nombre arrow

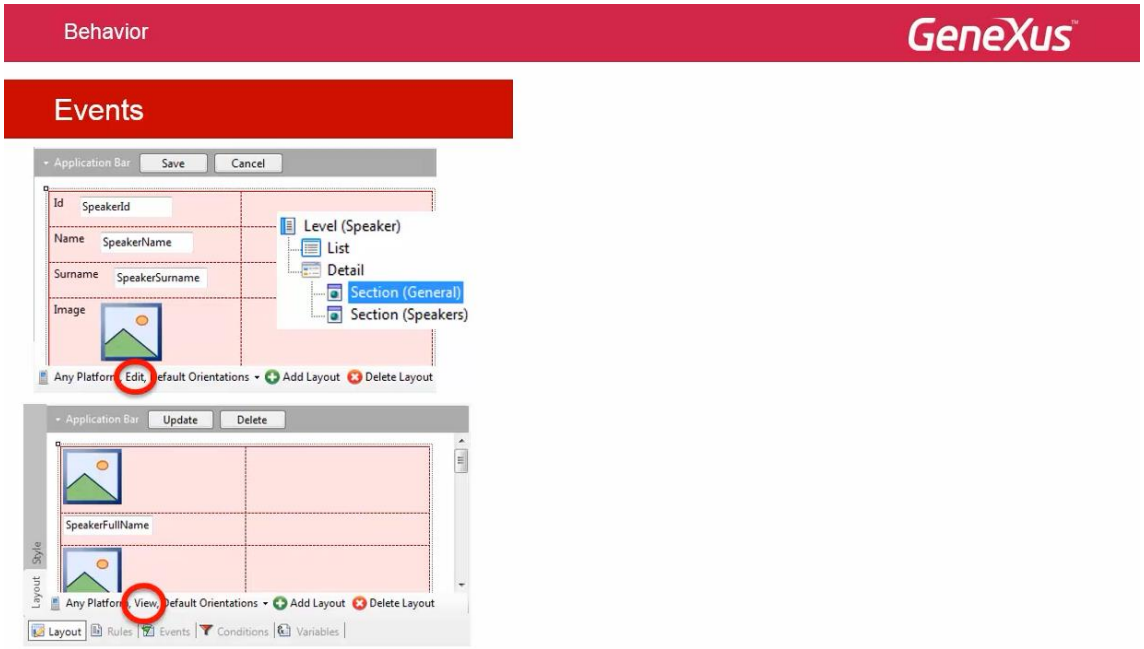


Todos estos eventos son del cliente, aunque invoquen a objetos que deban llamar a servicios -data providers-, del server para devolver los datos o grabarlos.

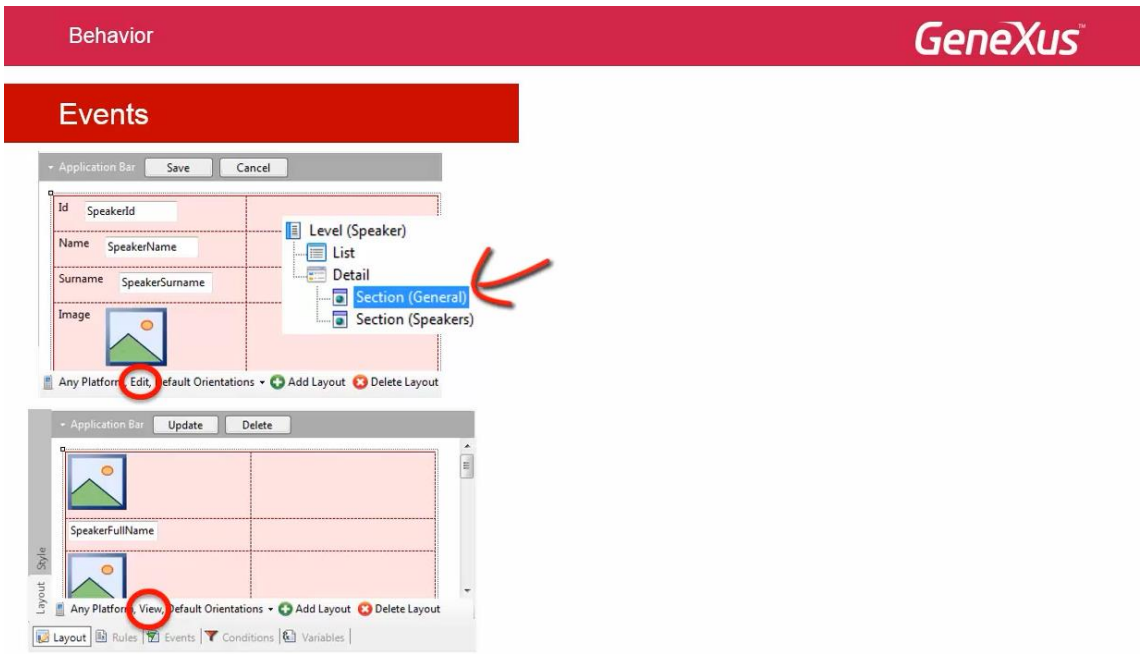
En cambio, el evento Load para cargar las líneas del grid



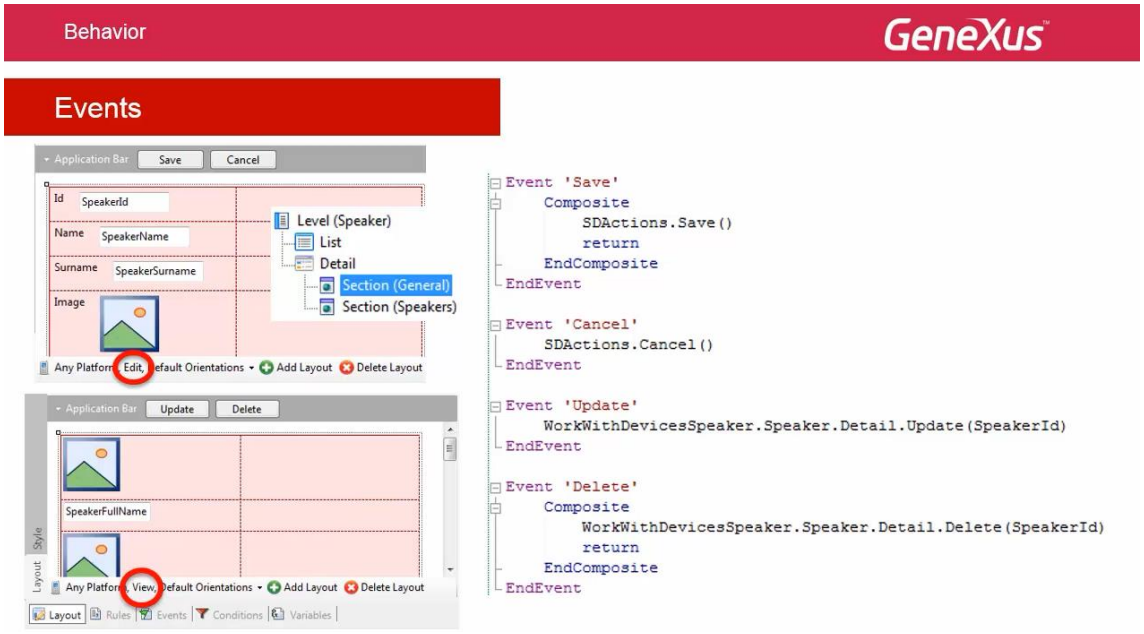
será ejecutado en el server si el objeto se ejecuta online.



Aquí vemos los layouts que creó el pattern para la section: General

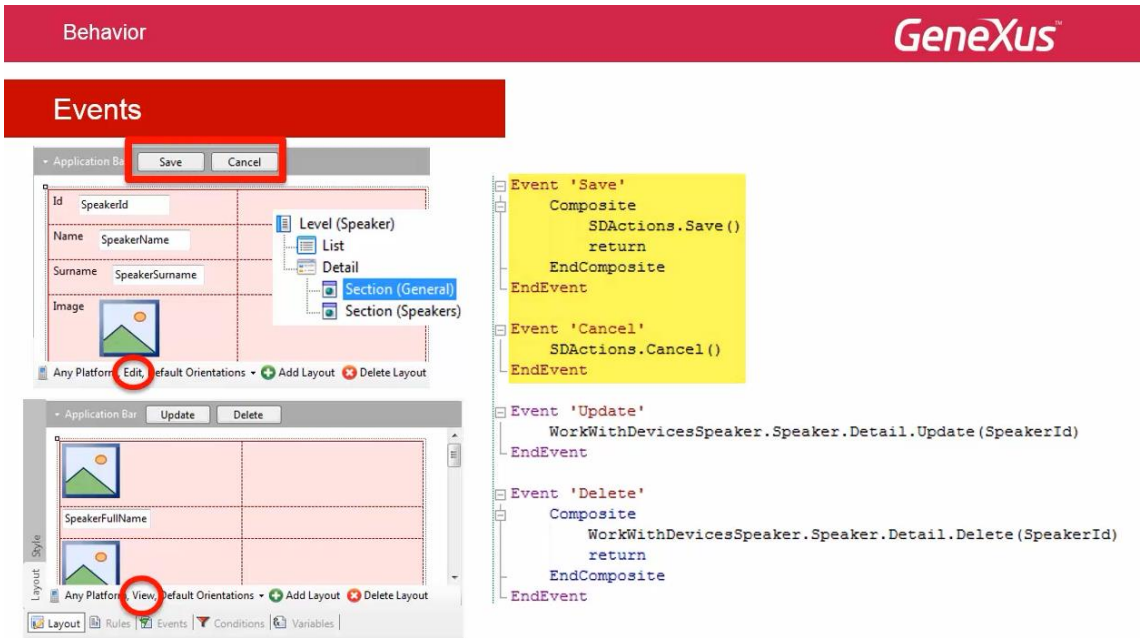


y los eventos Default

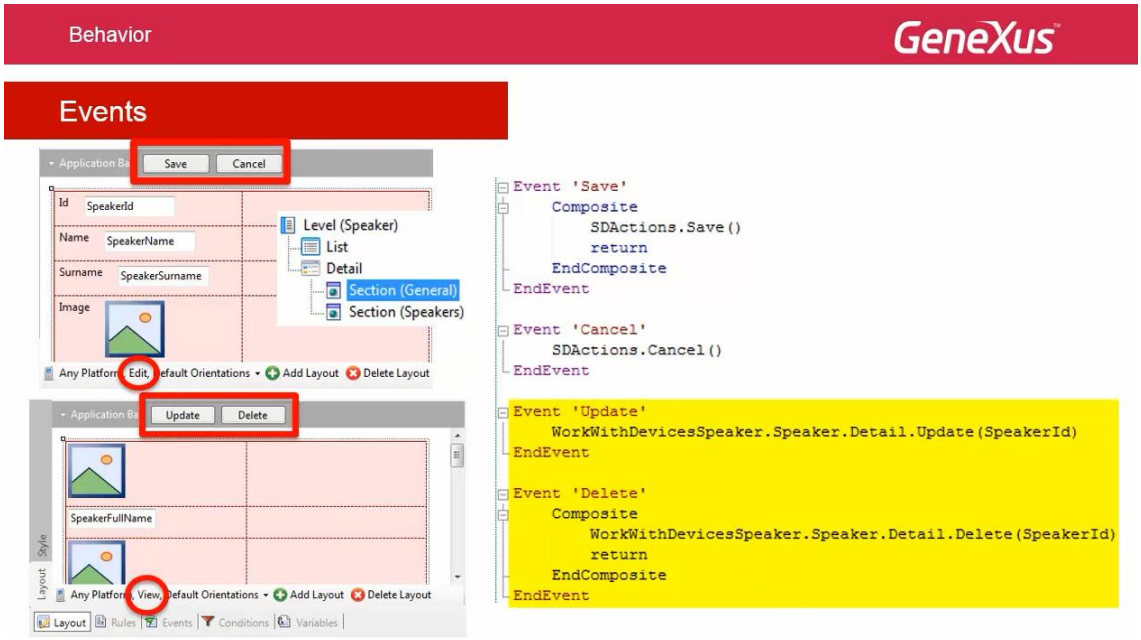


Son cuatro.

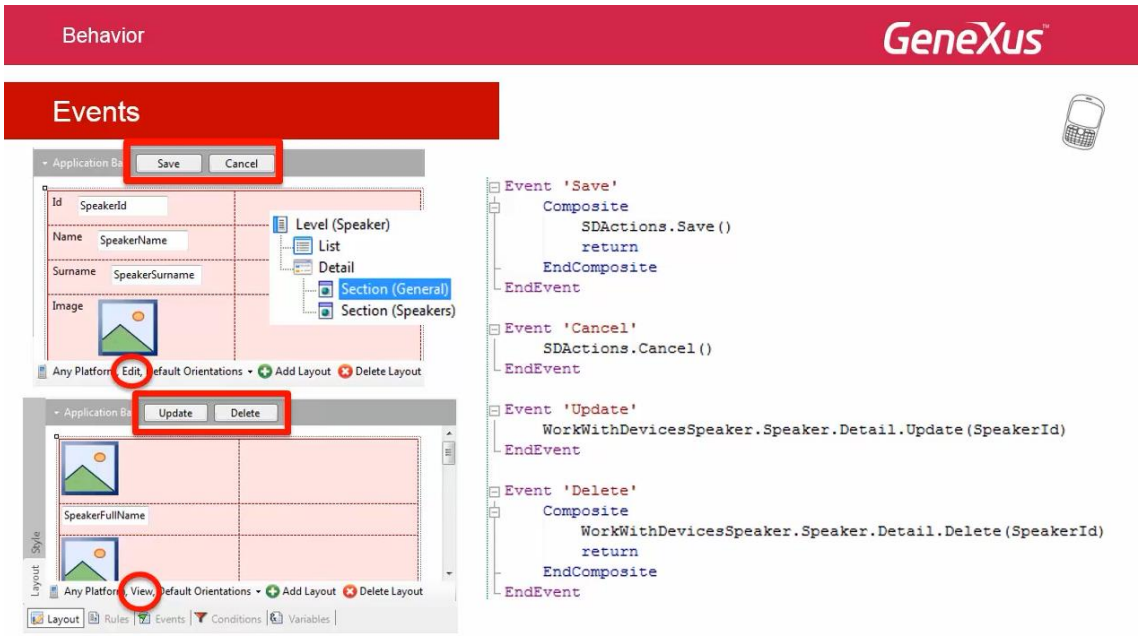
Dos aplican al modo Edit



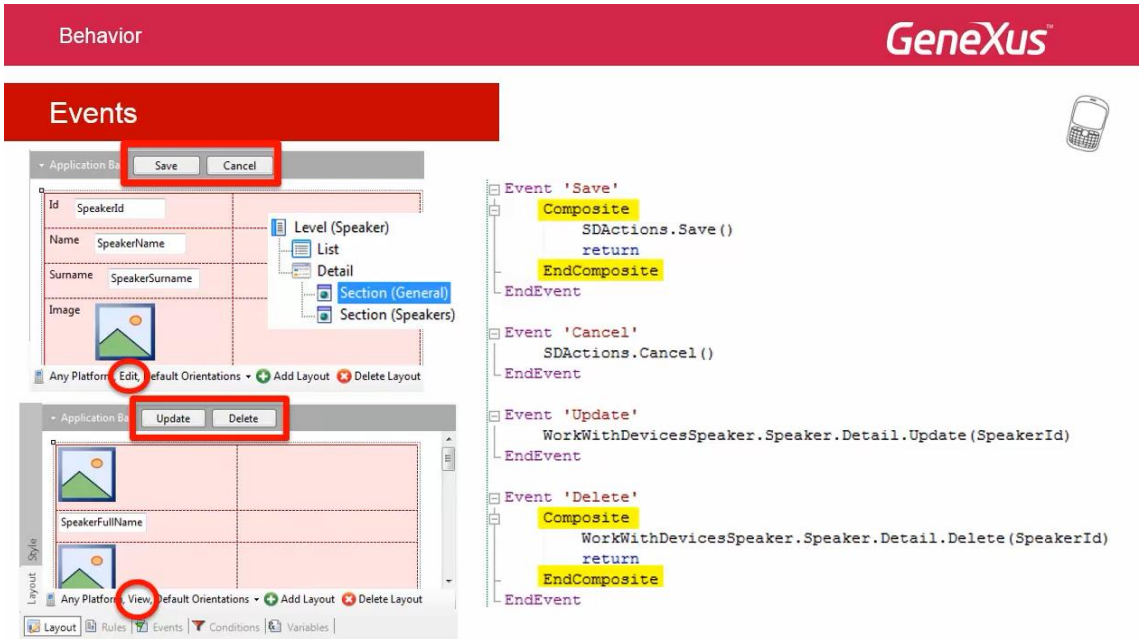
Y los otros dos al modo View..



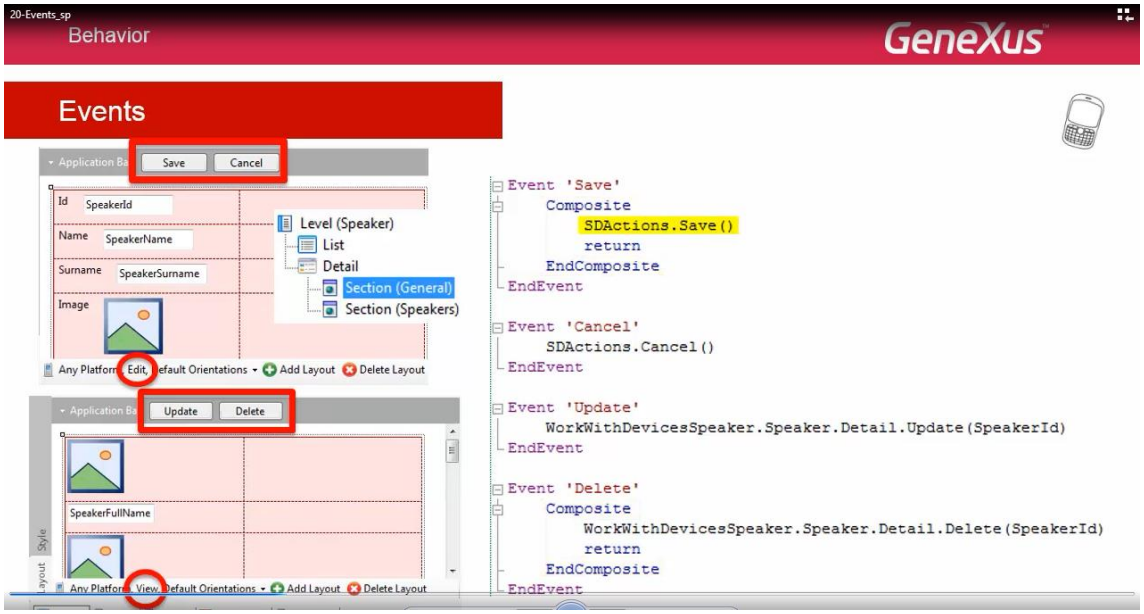
Estos también serán eventos del cliente



Reparemos en que aparece el comando Composit

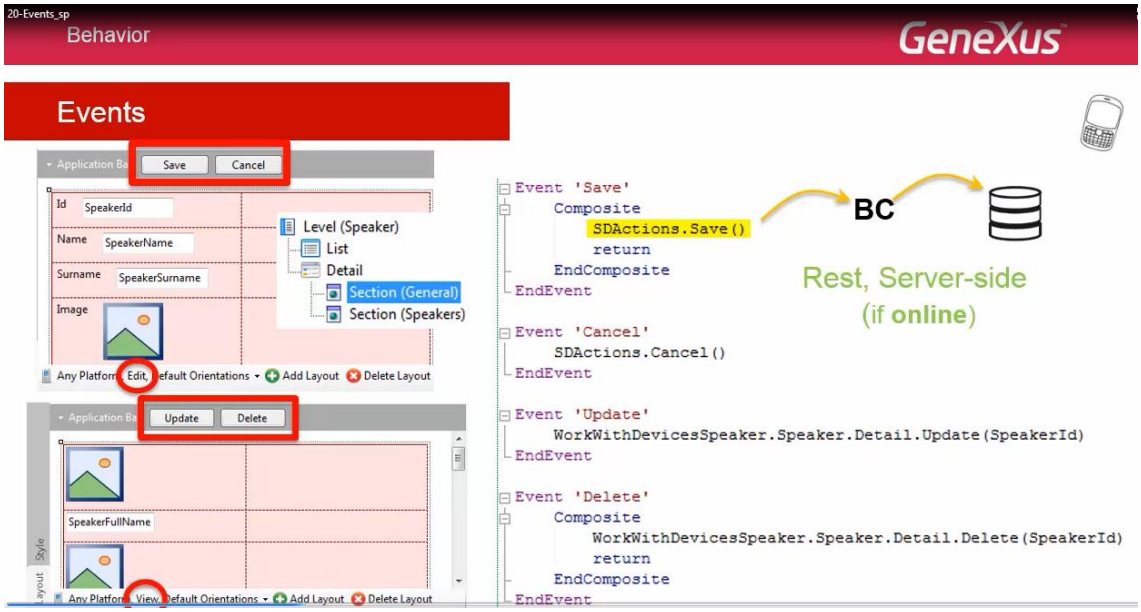


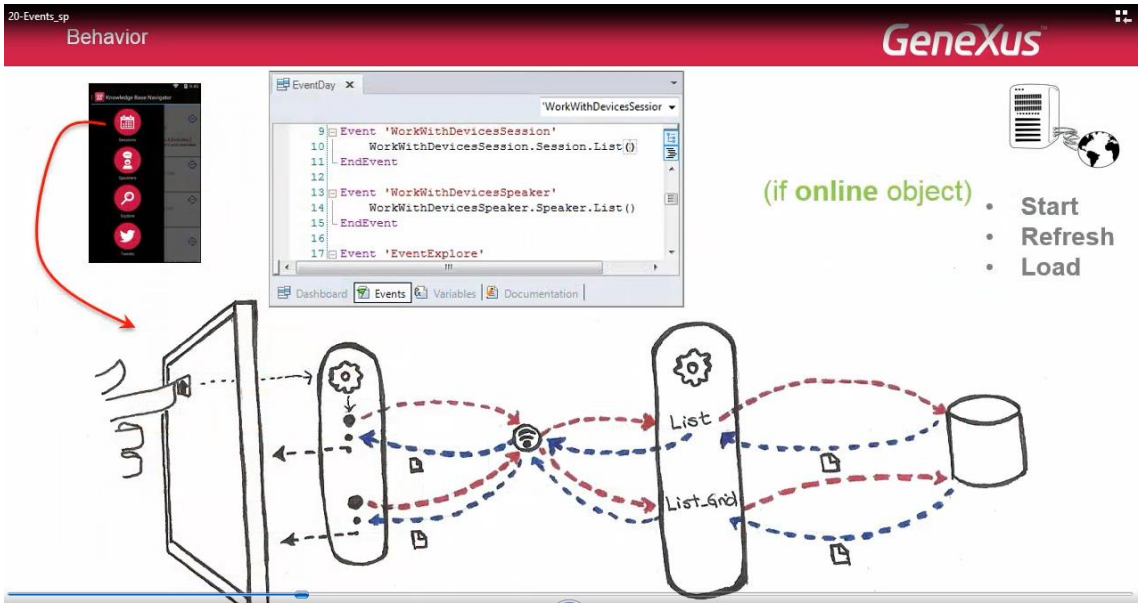
que estudiaremos luego.. y en el evento Save podemos ver el objeto externo SDActions:



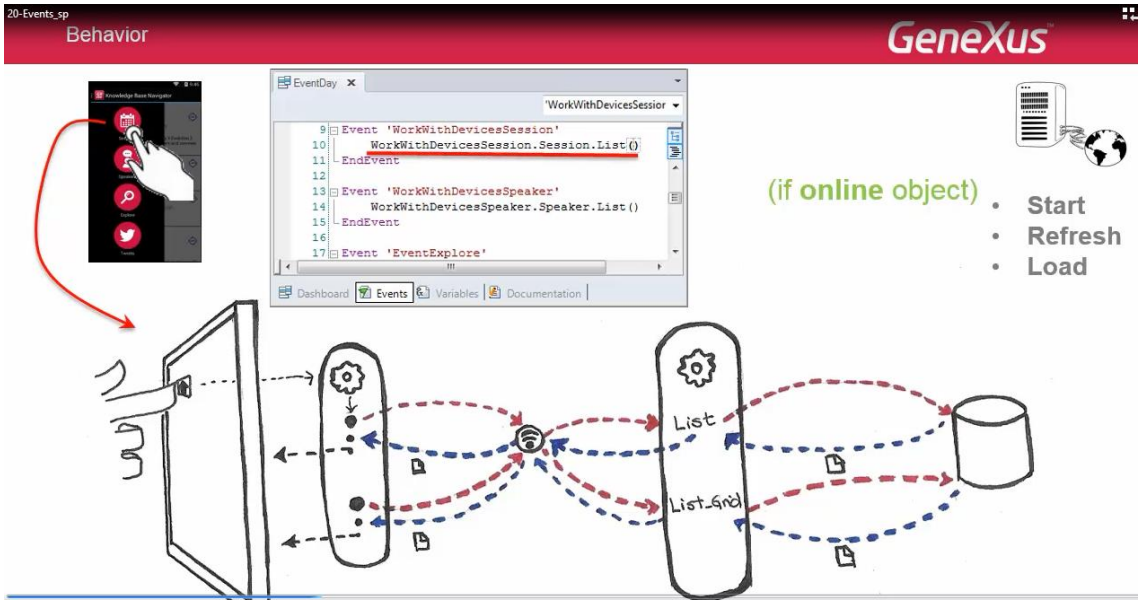
Se encuentra en el folder de las APIs sobre el cual ya hablaremos.

Lo que hace es encapsular la invocación al Business Component Rest que se encuentra en el server, de manera de grabar la información del speaker en la base de datos.

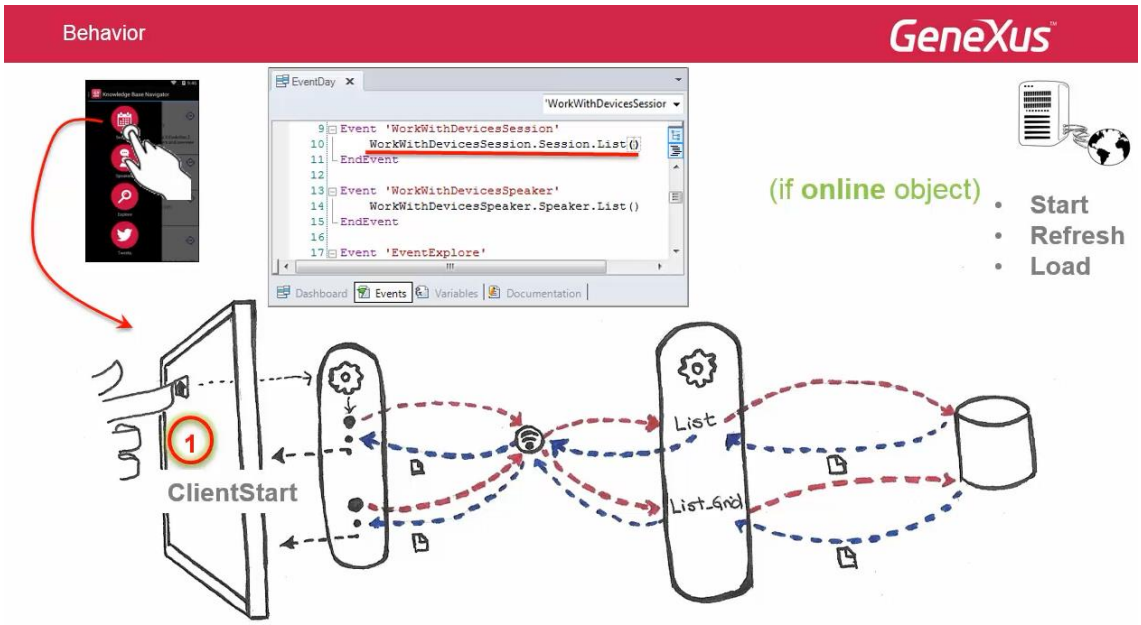




Veamos los eventos del sistema que se ejecutan en el server en el marco de una aplicación online.

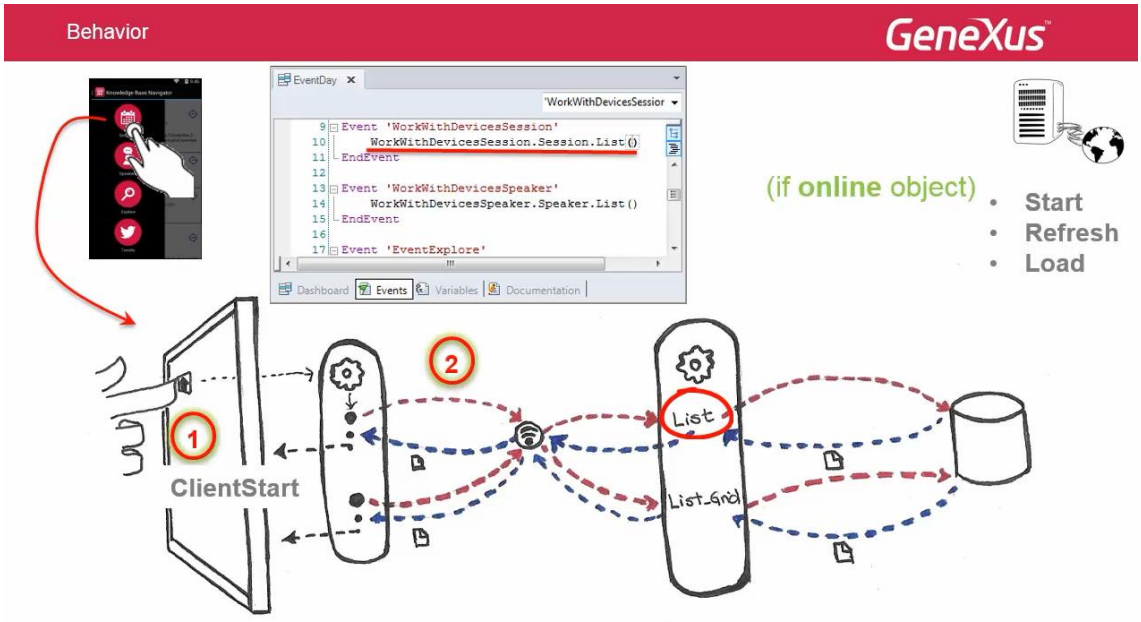


¿Qué sucede cuando desde el Dashboard invocamos al List de Sessions?



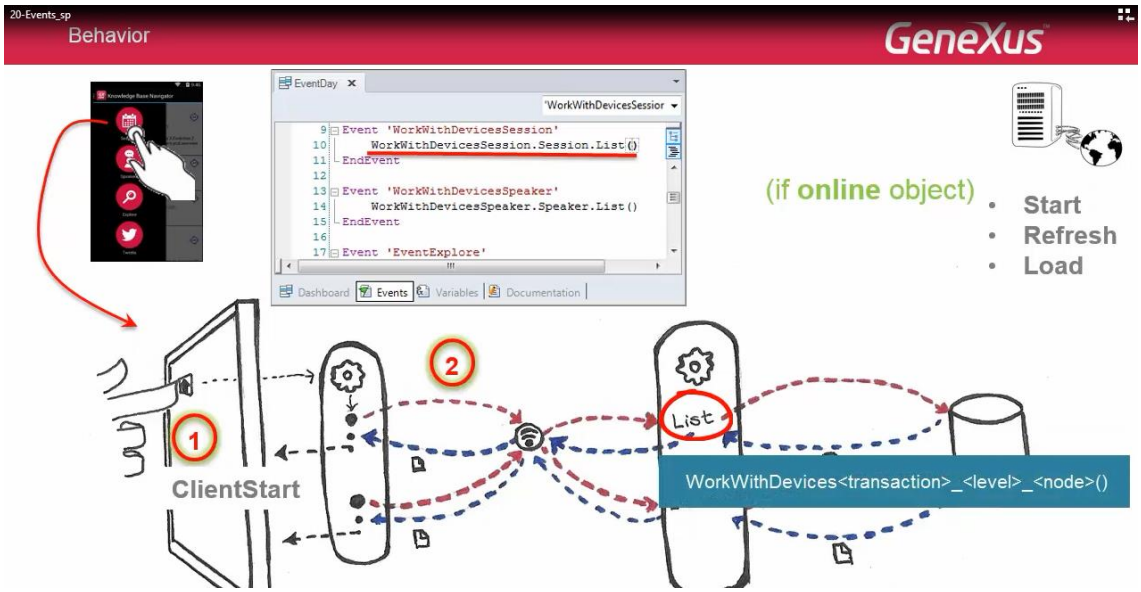
En el cliente este List ejecuta el evento ClientStart.

Y luego hace un call externo al servicio Rest

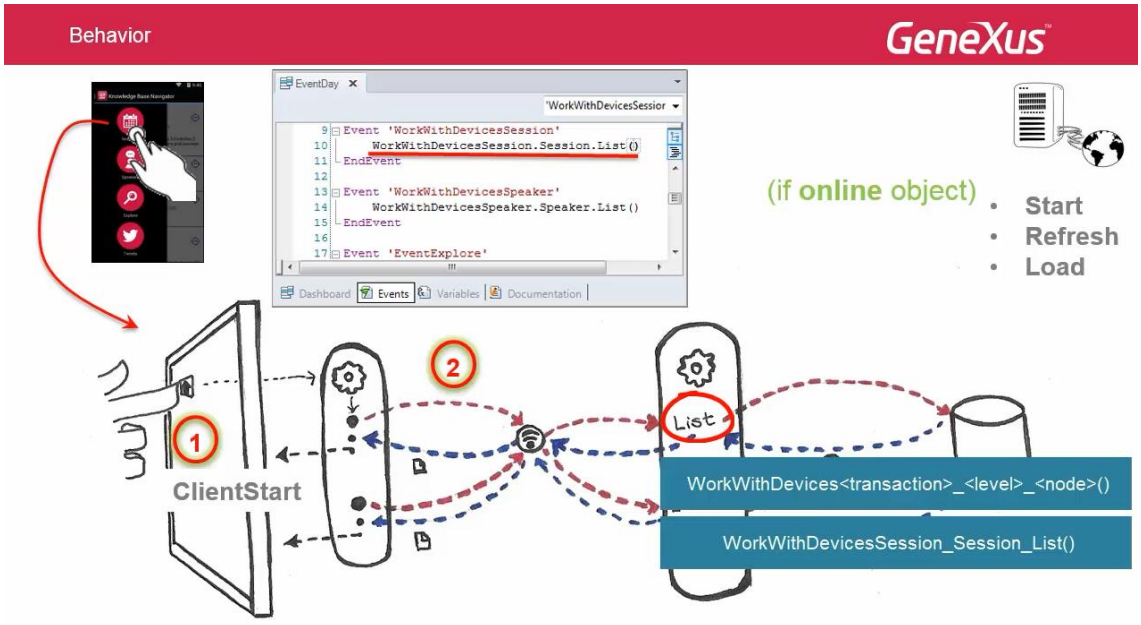


que devolverá los datos de la parte fija del panel.

Será un data provider creado automáticamente por GeneXus, que no queda visible en la KB pero que aparecerá en el listado de navegación con el nombre Work With Devices - Nombre de la transacción – Nivel del workwith del que se trate – y nodo en el que nos encontramos.



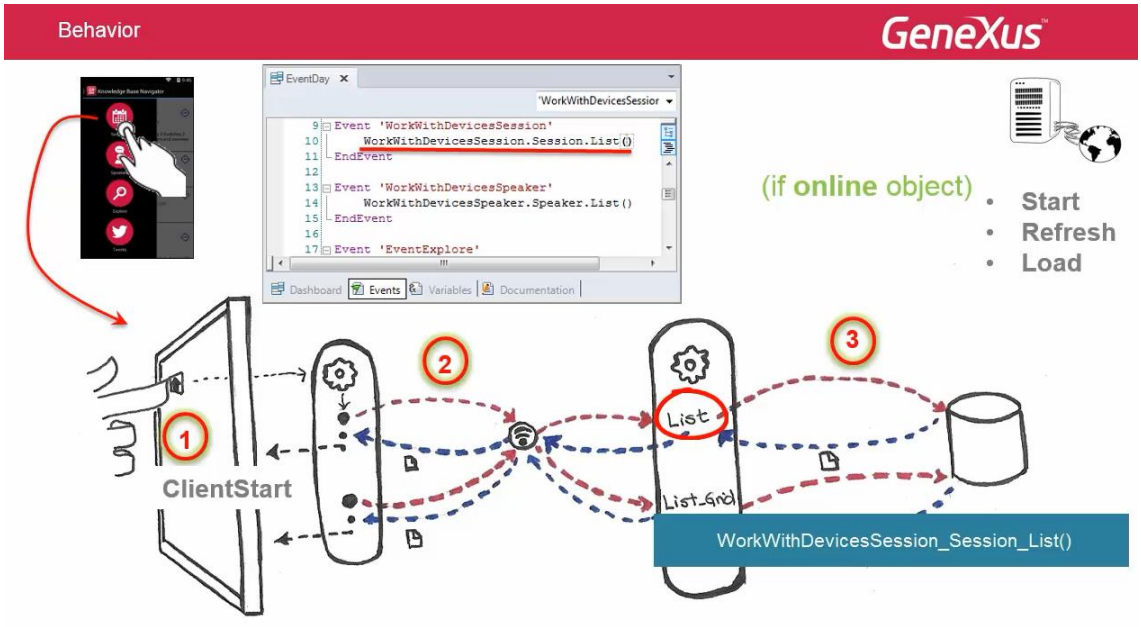
En nuestro caso será:



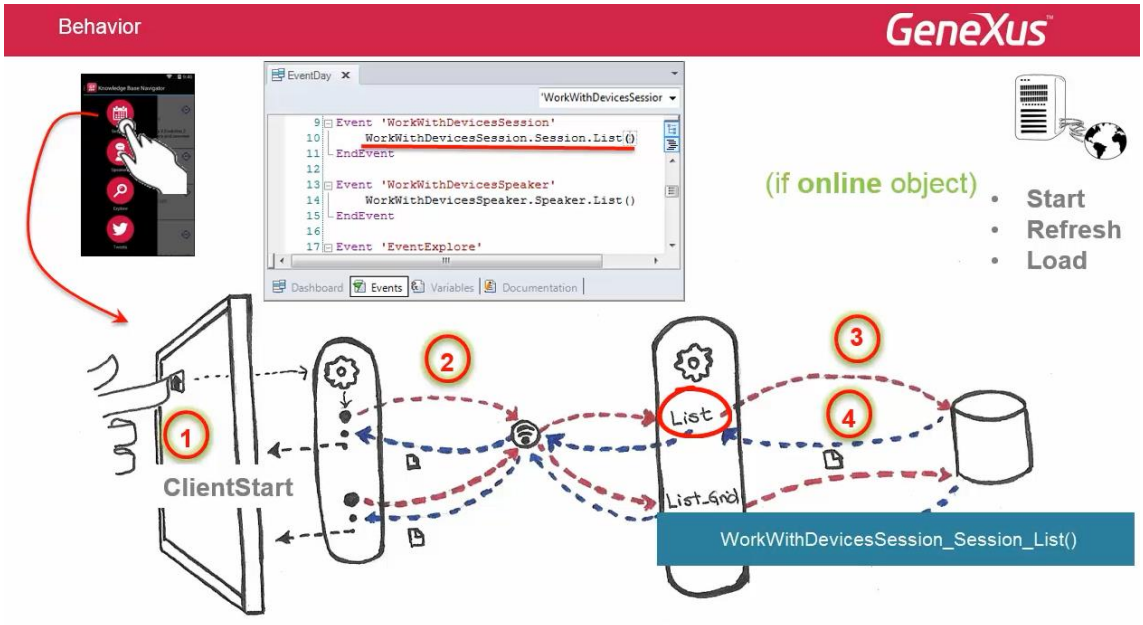
- WorkWithDevicesSession (nombre de la transacción)
- Session (nombre del nivel)
- Y List (el nodo al que estamos invocando)

Este data provider se ejecuta en el servidor, dentro de su lógica interna se ejecutan los eventos Start y Refresh.

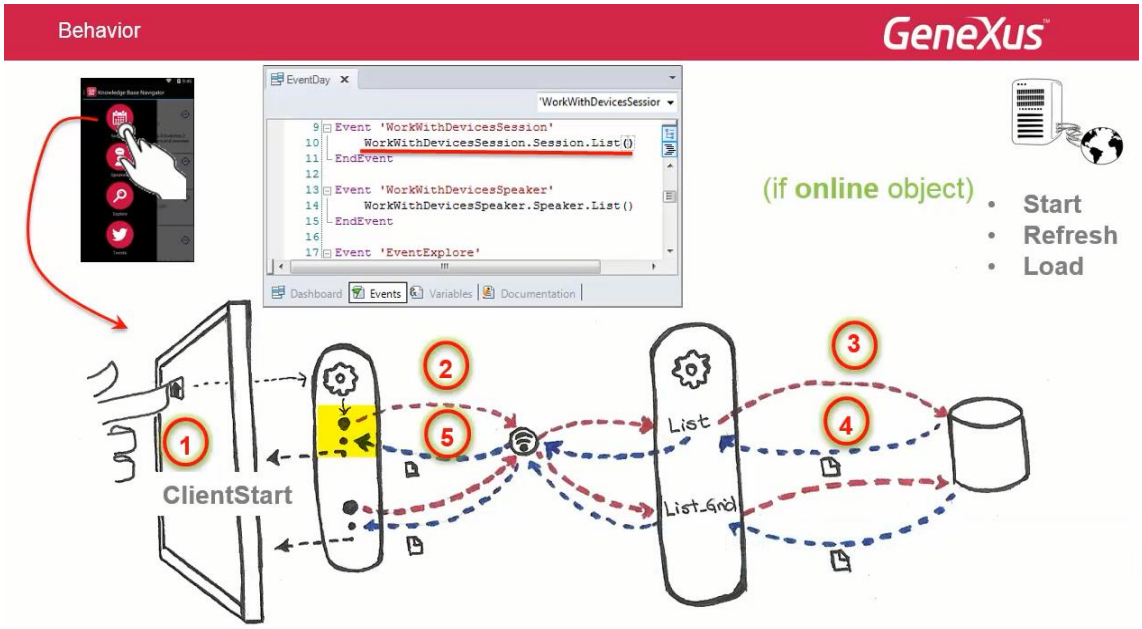
Se accede a la base de datos si es necesario



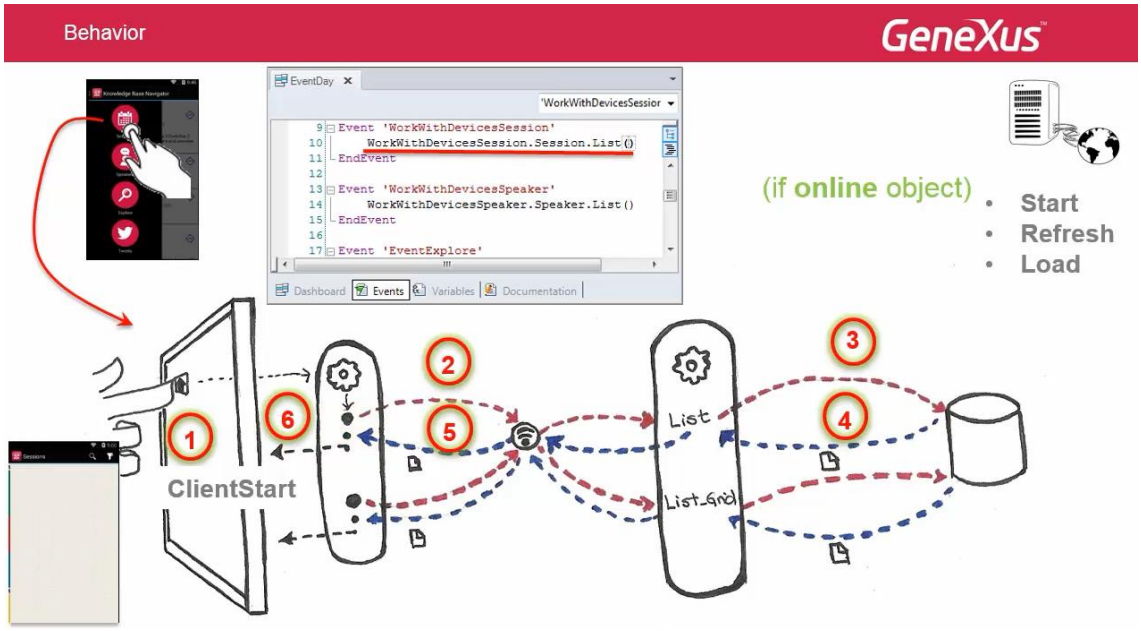
La base de datos encuentra los resgistros y los devuelve al data provider



quien devuelve en su response, la información a la parte de la aplicación que corre en el dispositivo, el work with que lo llamó:

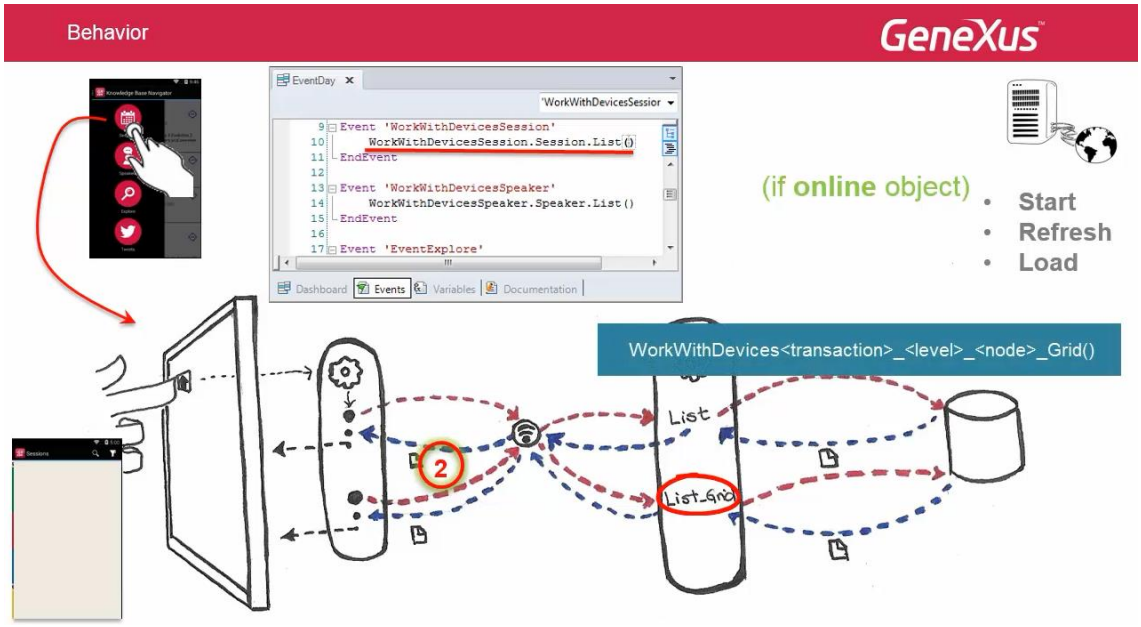


Este work with, busca las imágenes que se necesiten, y con la respuesta recibida, dibuja la interfaz de usuario correspondiente a la parte fija del List del Work With



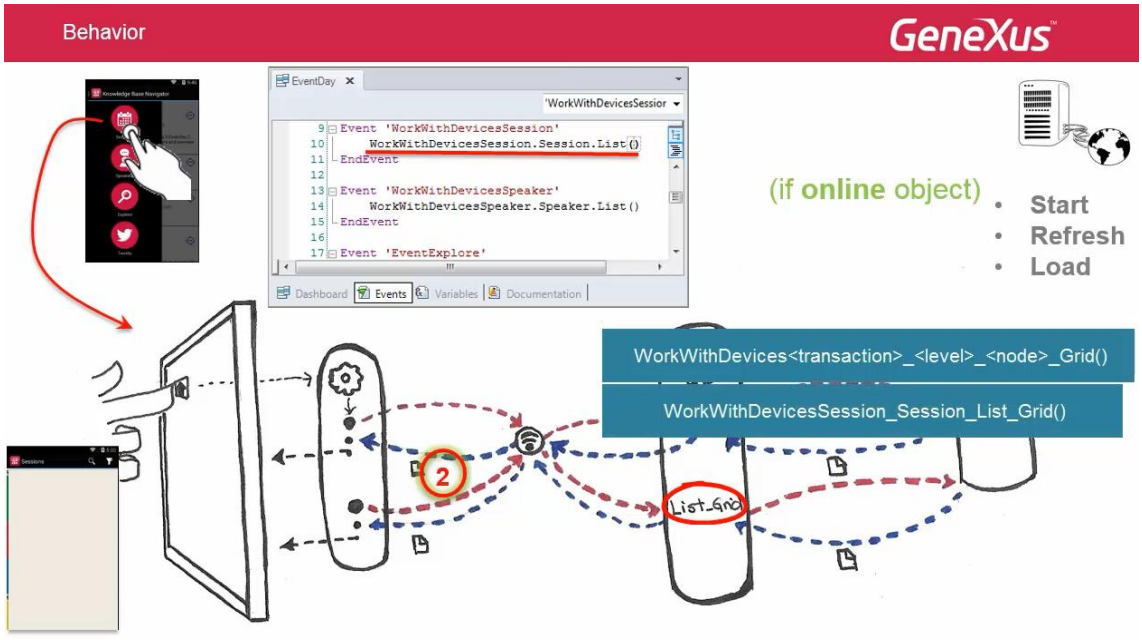
En nuestro caso no requería datos.

Luego, se repiten los pasos anteriores.. pero ahora llamando al data provider creado automáticamente por GeneXus y transparente para nosotros, que devuelve los datos del grid.

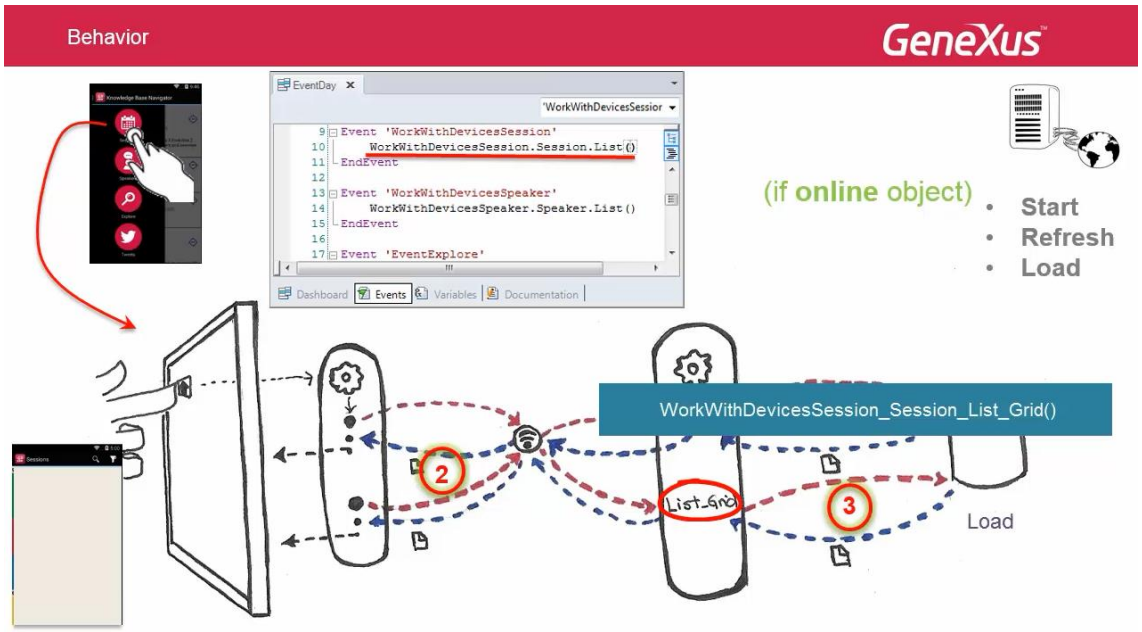


Su nombre será WorkWithDevices – nombre de la transacción – nivel – nodo – Grid

En nuestro caso, será este:

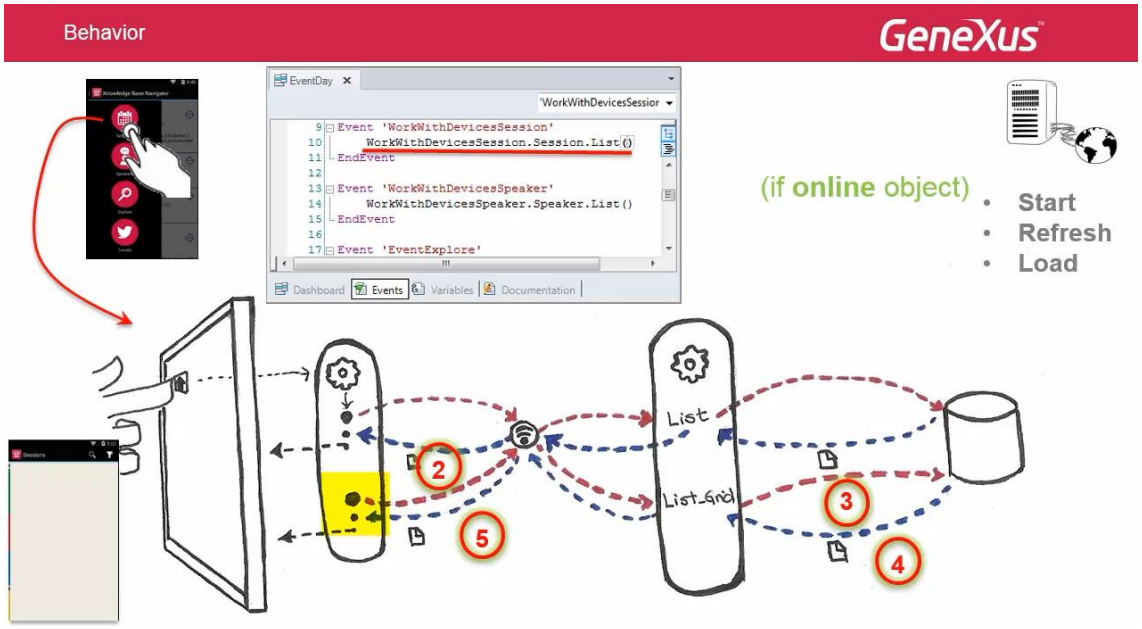


Internamente, este data provider ejecuta el evento Load

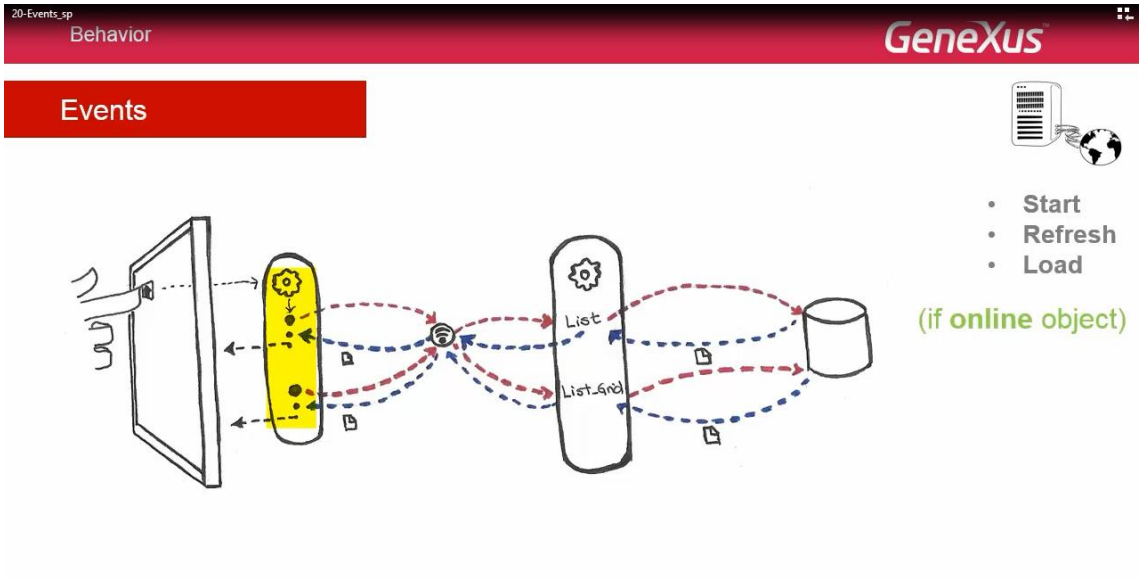


por cada línea del grid a ser cargada.

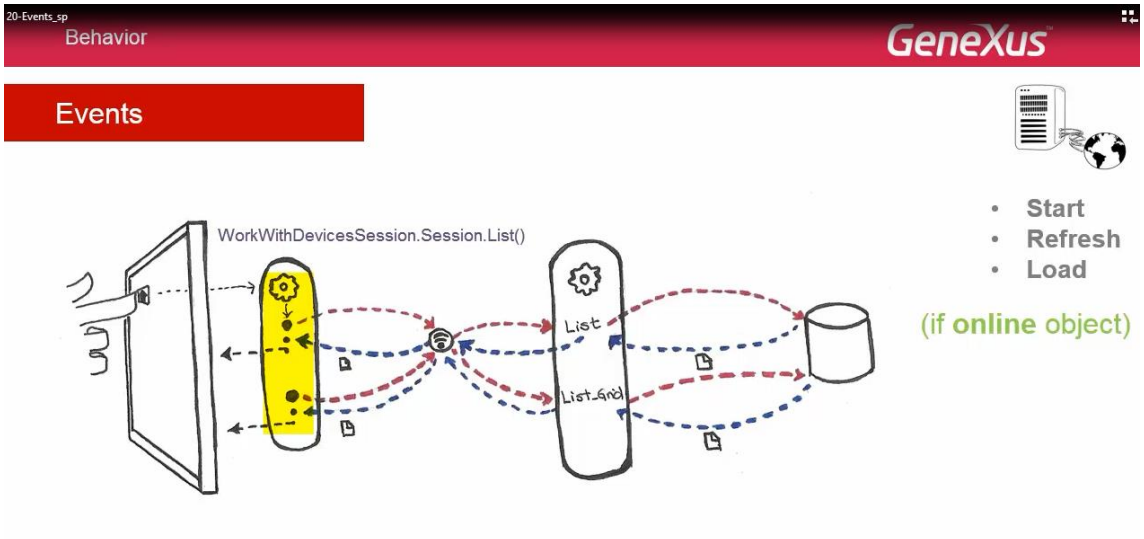
Una vez la aplicación en el dispositivo recibe la respuesta con los datos del grid,



dibuja la interfaz de usuario correspondiente al grid.



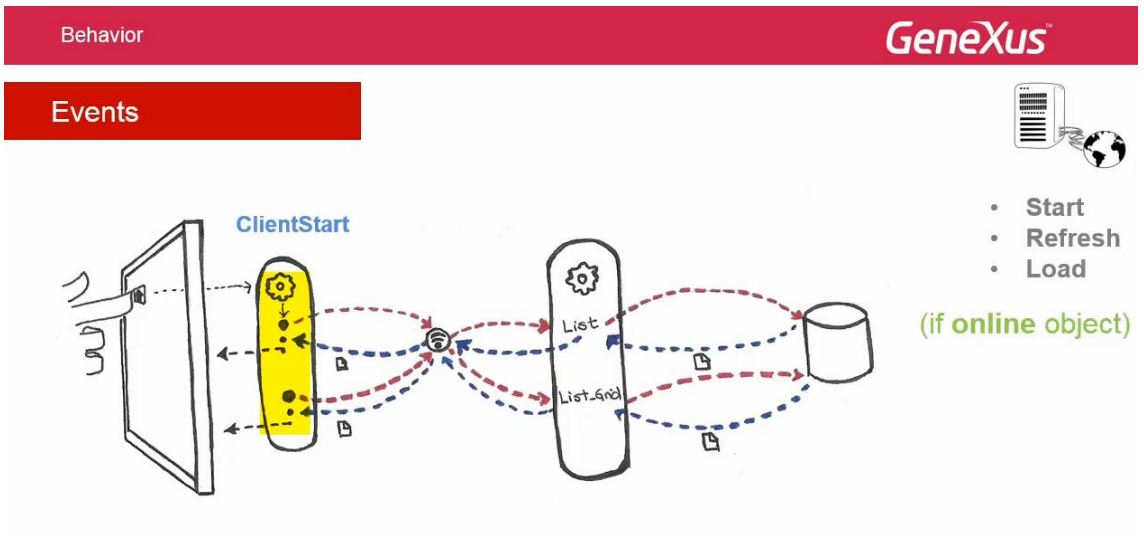
En definitiva, del lado del cliente, el dispositivo, tenemos en el evento del dashboard, que activamos al hacer TAP sobre la imagen



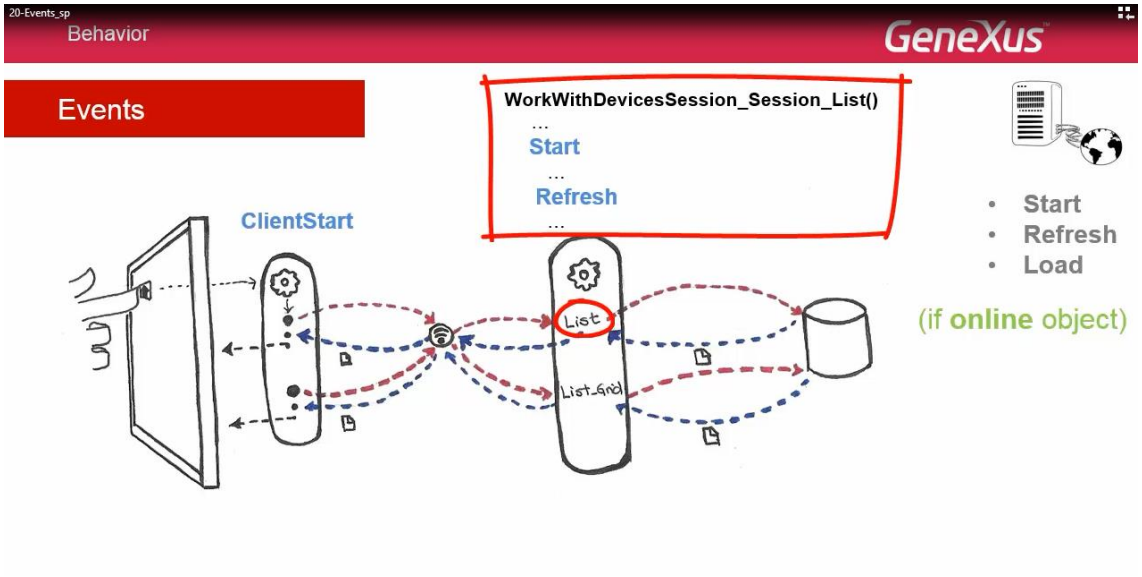
una invocación al List del work with de sessions.

El código correspondiente es el que corre en el cliente, que comienza a ejecutar entonces, ese List.

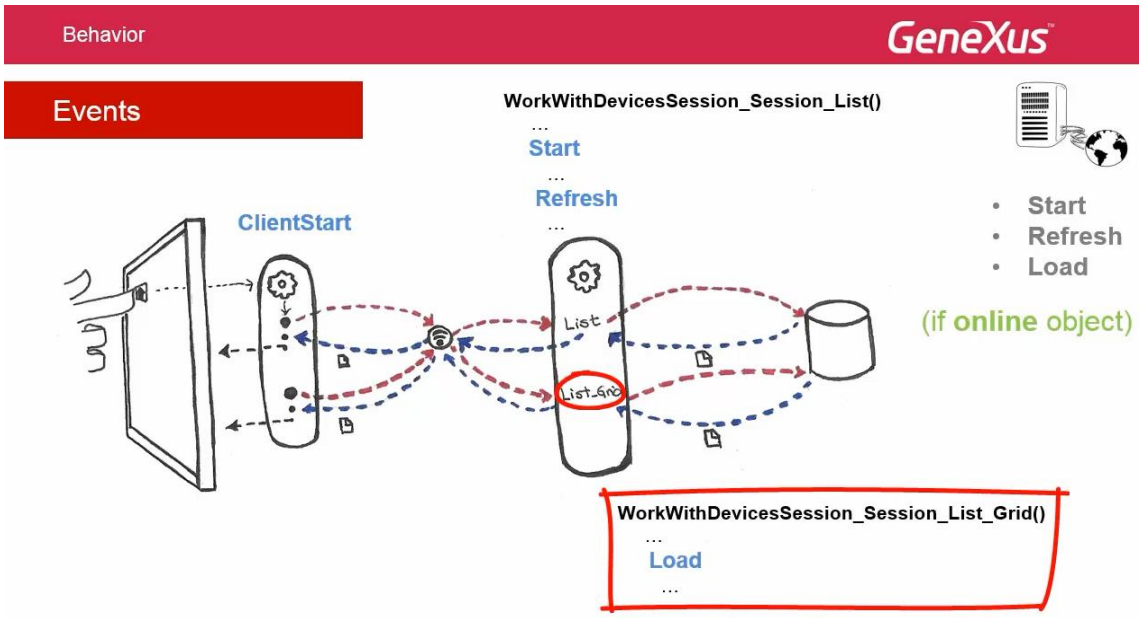
En su lógica interna, este work with, ejecuta el evento Client Start



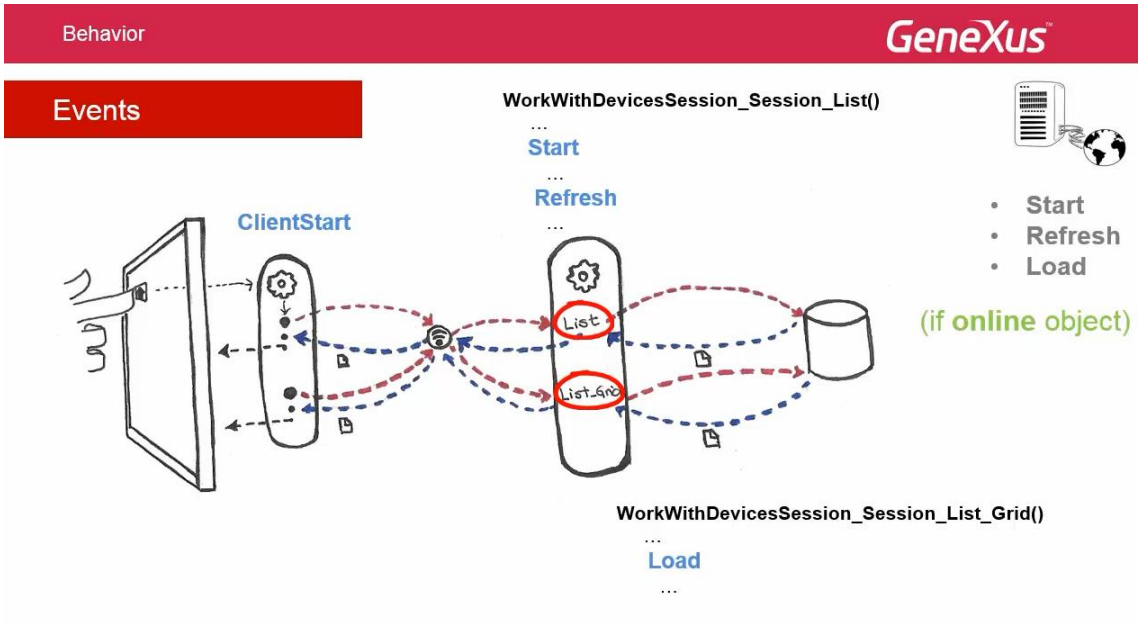
y luego, llama al servidor, para que este le devuelva los datos a cargar en la parte fija a través de un servicio Rest



Y luego le pide al servidor, que ejecute el otro servicio Rest. El que devuelve los datos del grid.

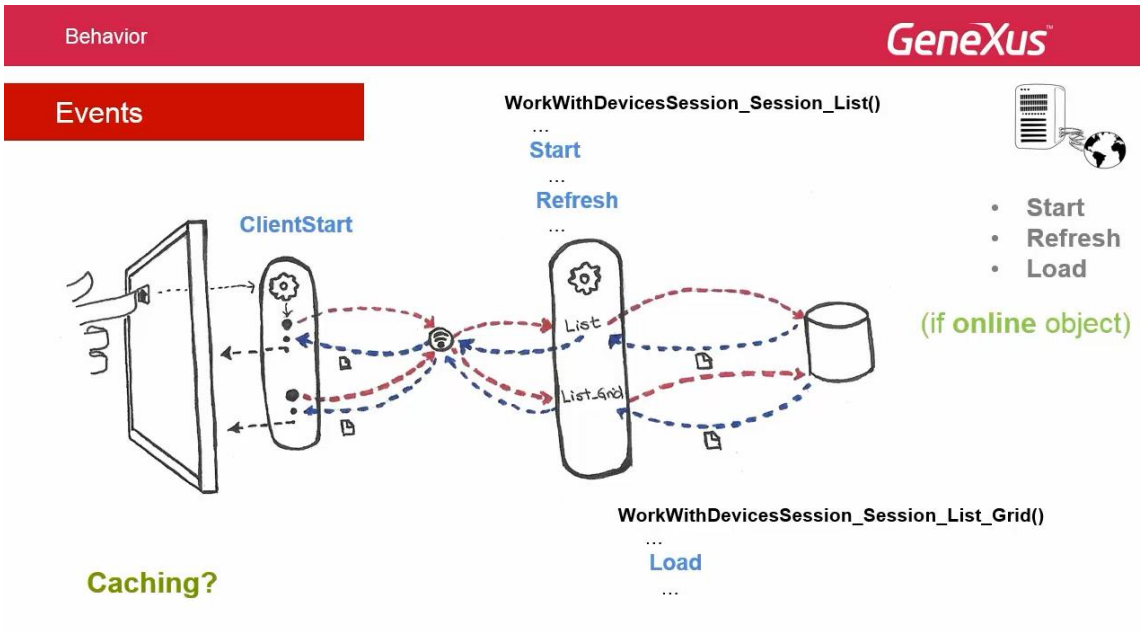


Con los datos devueltos en 2 responses

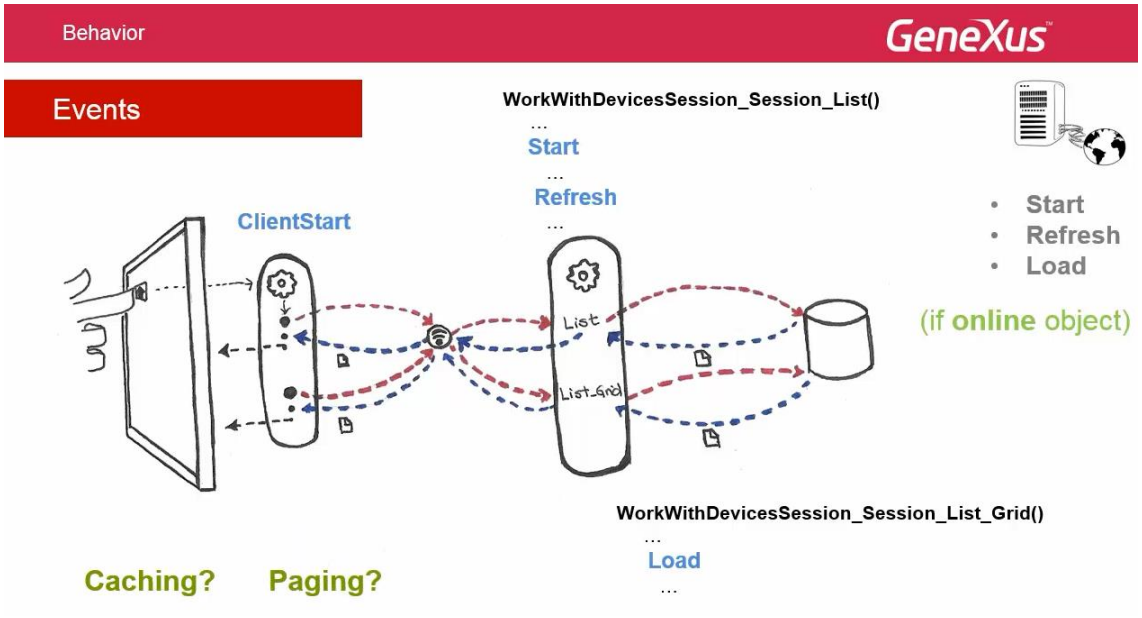


arma la pantalla: por un lado la parte fija y por otro el grid.

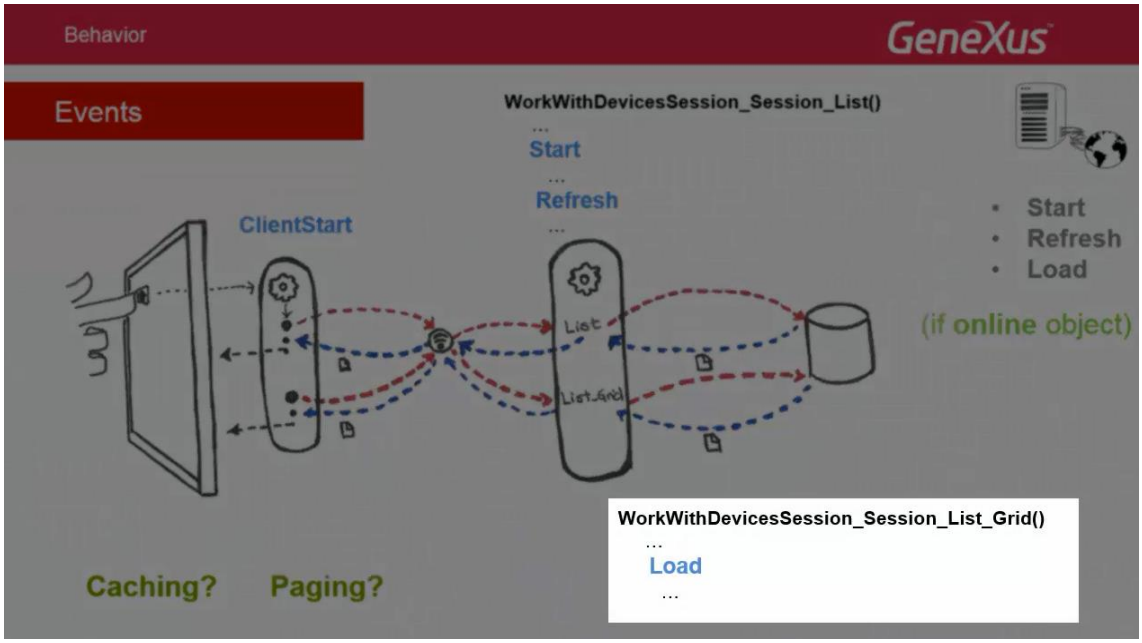
Estamos suponiendo que no hay caching para simplificar la explicación



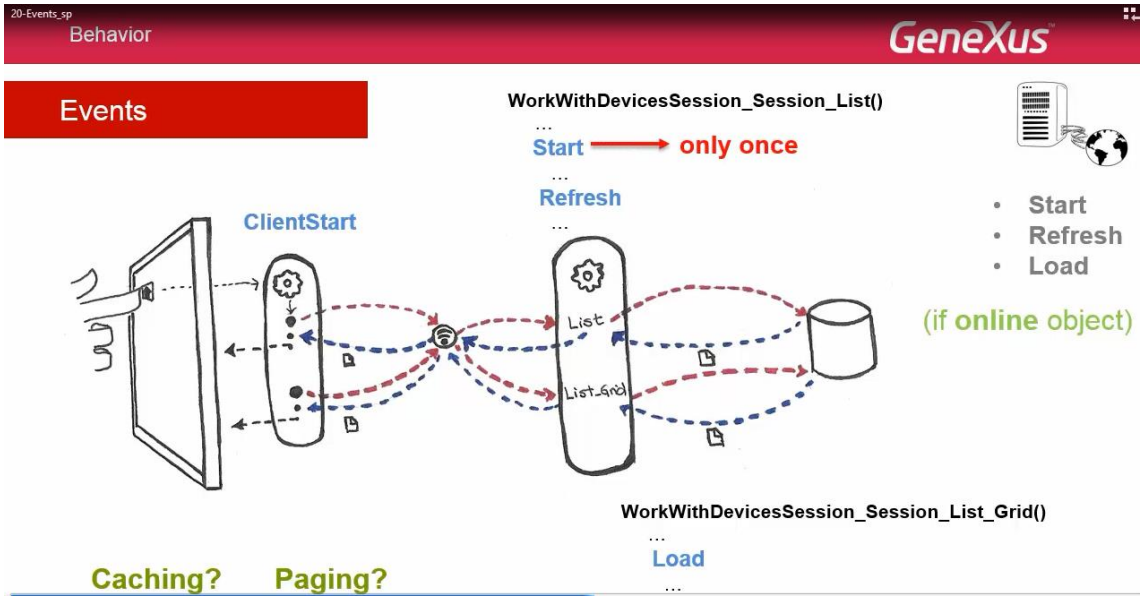
Tampoco estamos poniendo el foco en el paginado del grid, que también se realiza



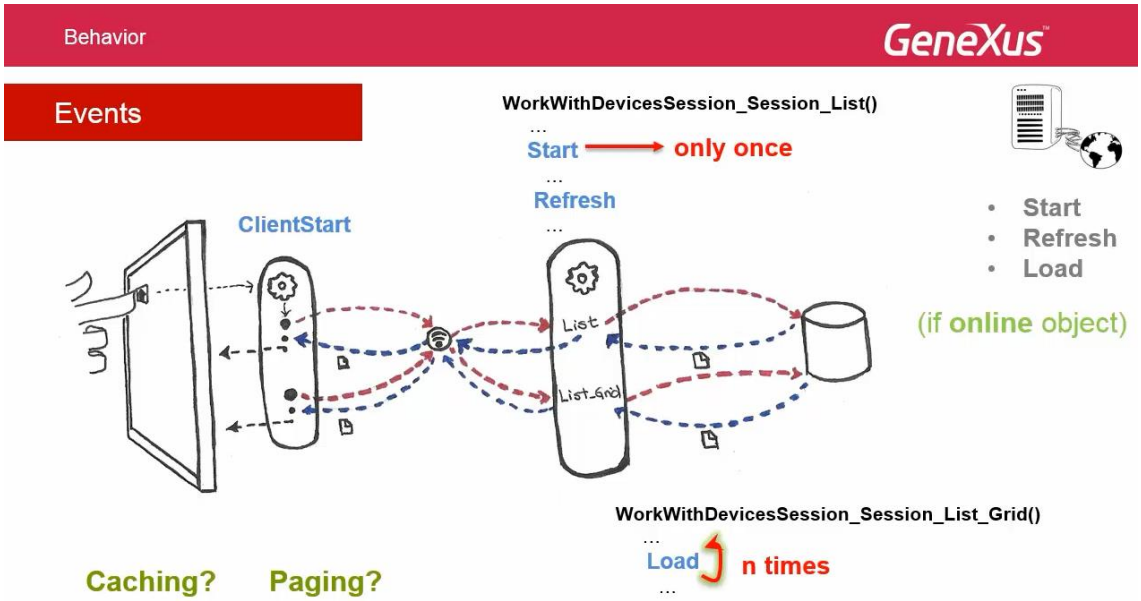
El data provider que ejecuta el Load



Lo hace paginando, es decir, sólo devuelve X registros por vez.



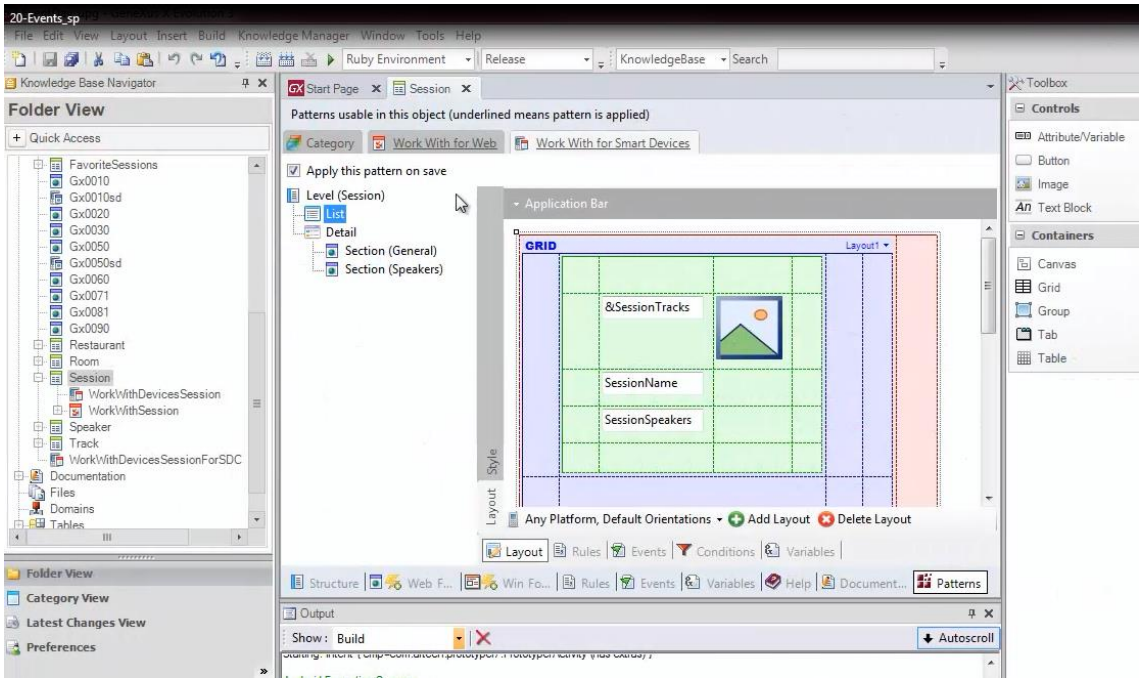
El evento start, se dispara solamente la primera vez. No se vuelve a ejecutar, a menos que se salga del panel y se vuelva a entrar.



El evento Load en el web panel, se ejecuta N veces, si el grid tiene tabla base, como es el caso.

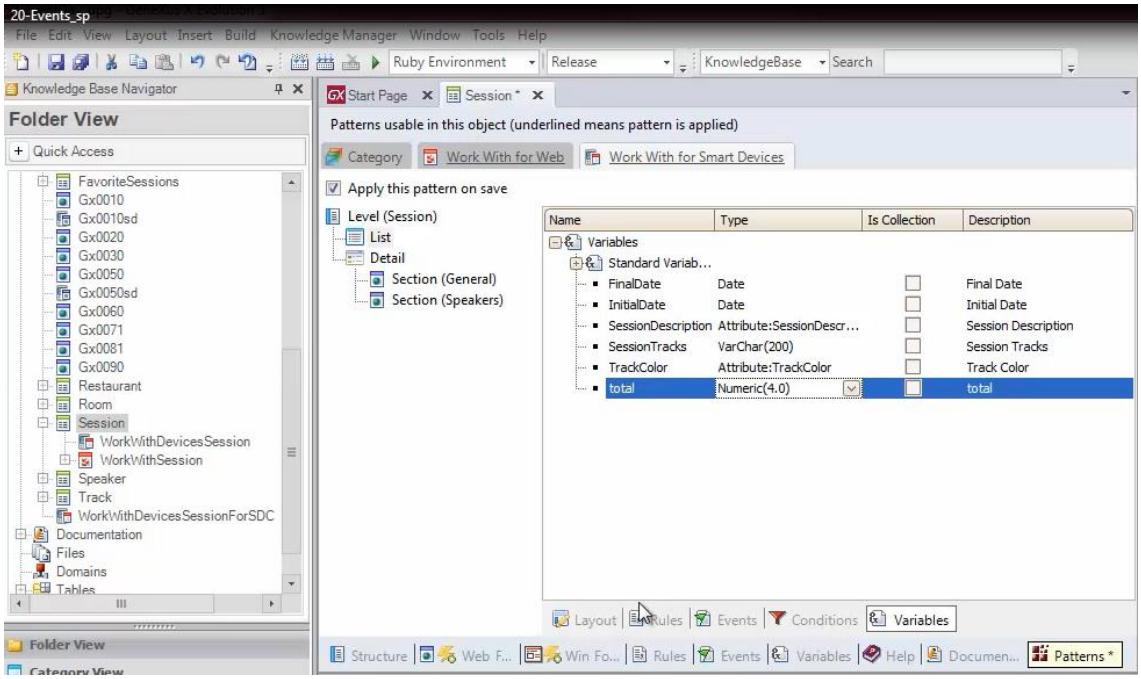
Podemos pensar entonces la diferencia entre un panel de Smart Devices (como es el List o el Detail) y un web panel.

En el de Smart Devices se separan las navegaciones de parte fija y grid… y se dibuja la pantalla correspondiente a la parte fija, con independencia de lo que suceda con el grid. Esto tiene consecuencias.

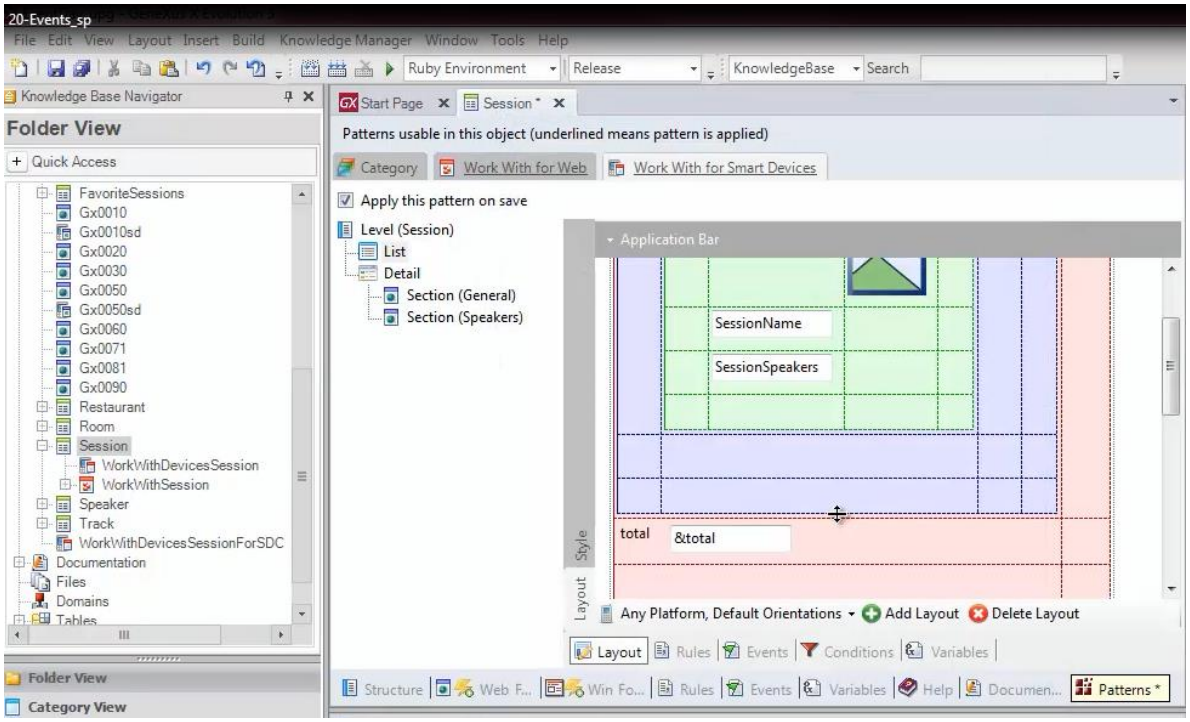


Supongamos que queremos mostrar en el List de Sessions, la cantidad de conferencias.

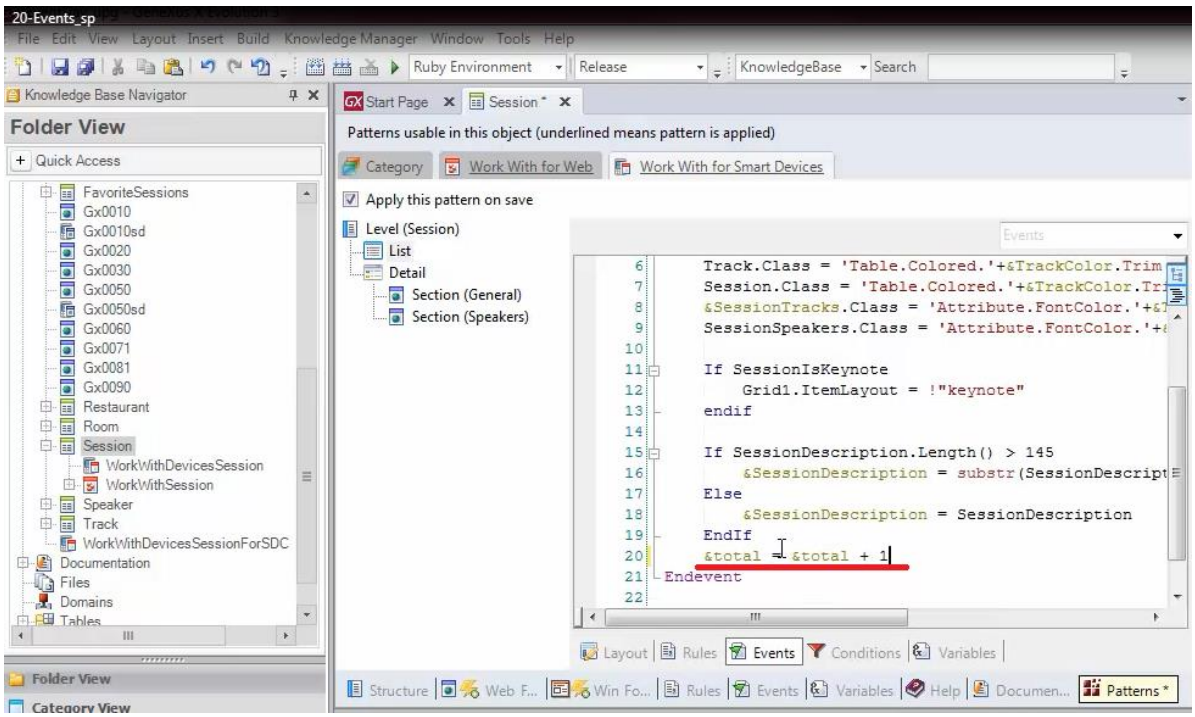
Si programáramos este panel al modo de un web panel, pondríamos una variable &total



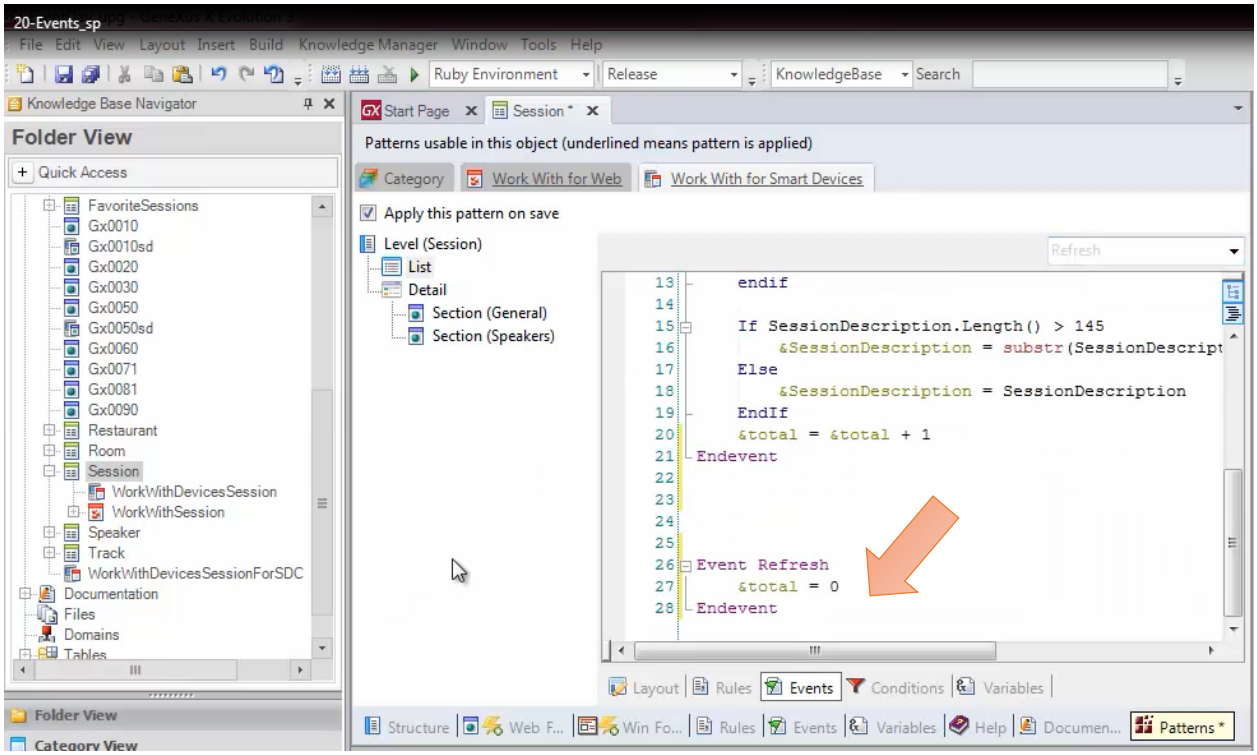
que agregaríamos en el layout, fuera del grid

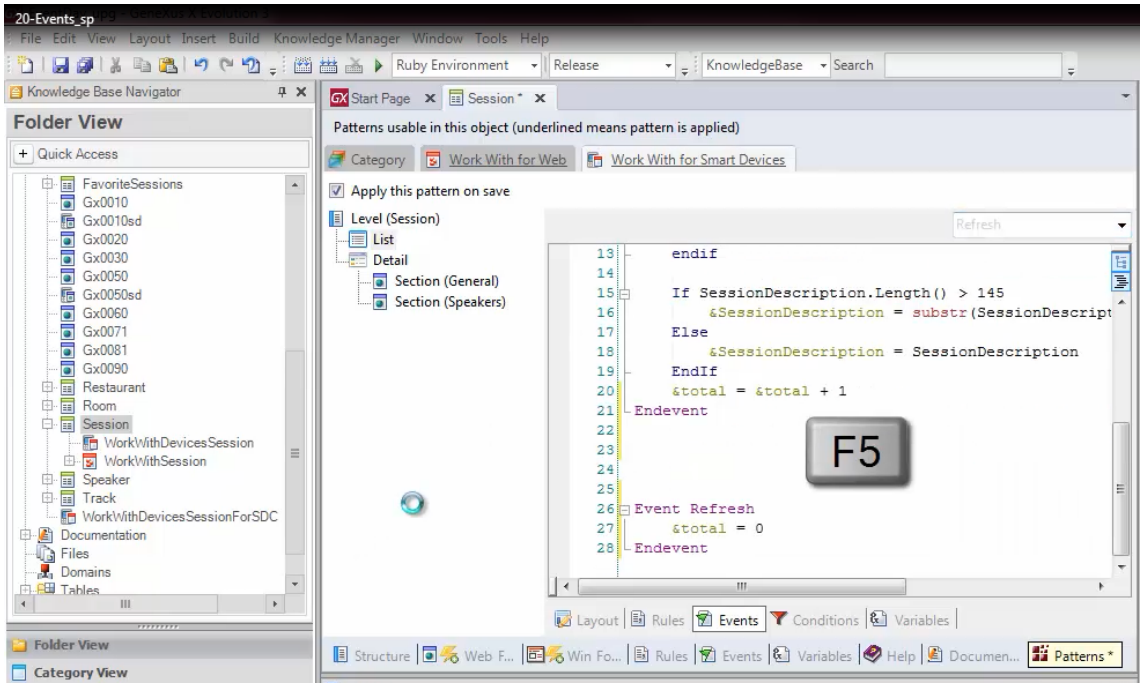


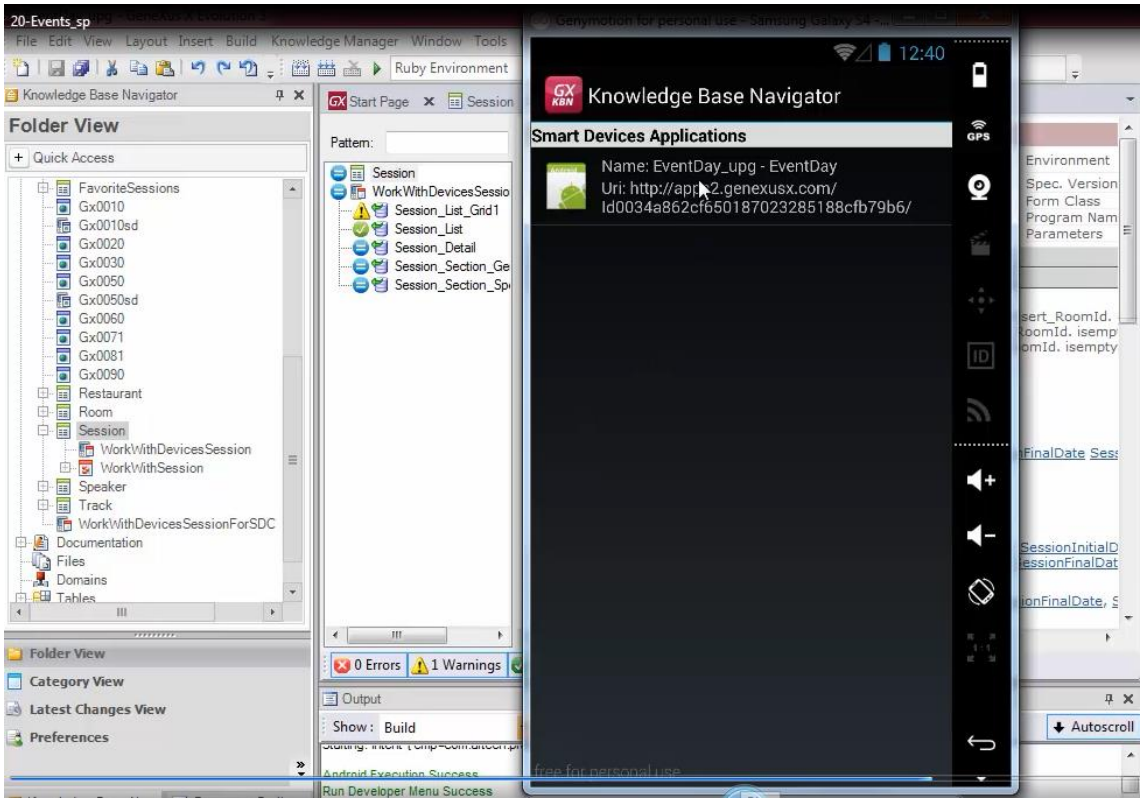
y la cargaríamos en el evento Load, asignándole al valor que tiene + 1

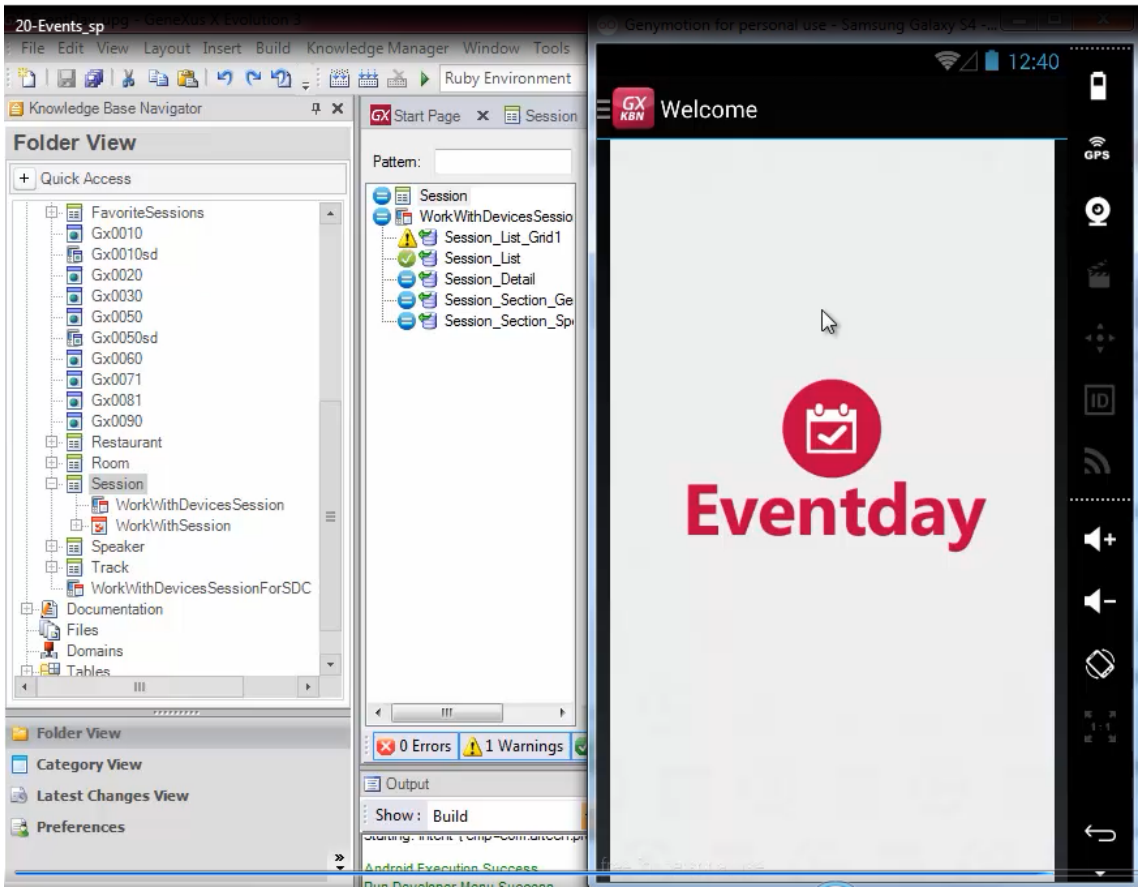


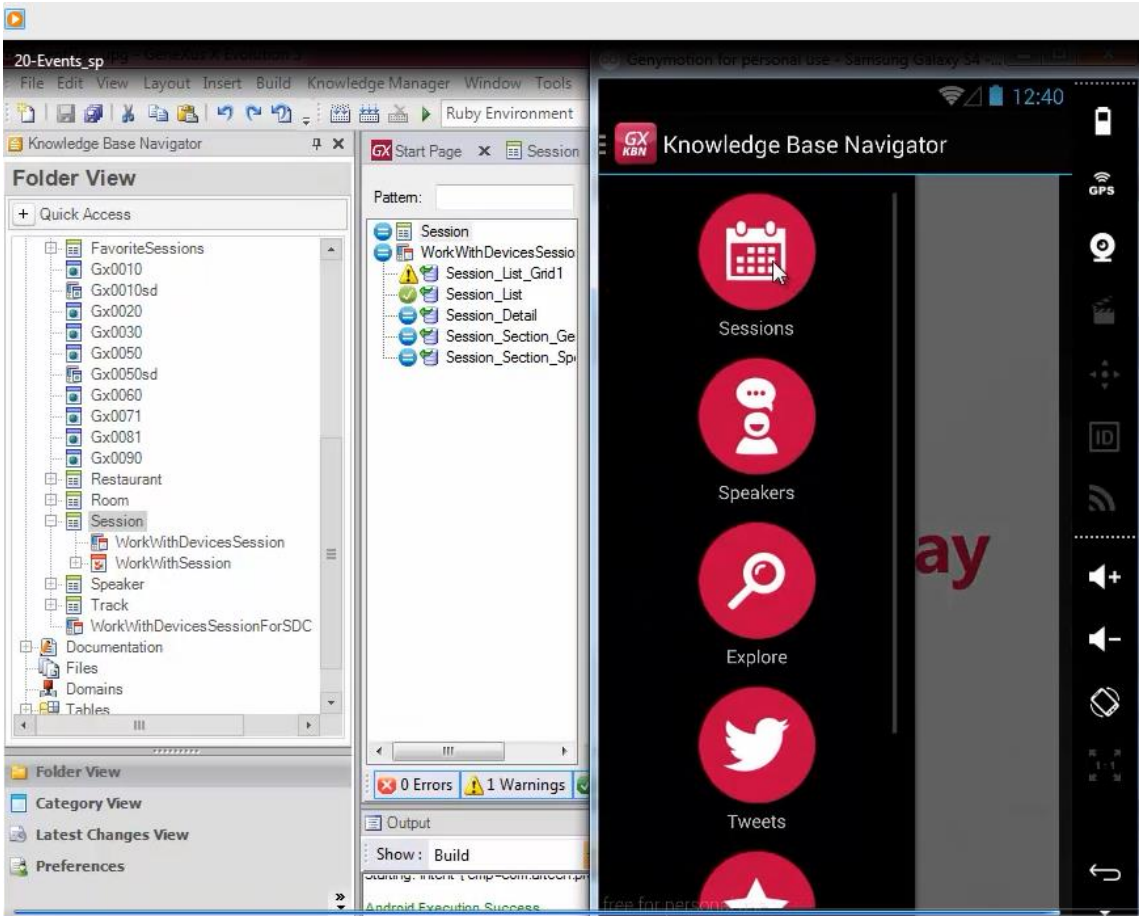
y la inicializaríamos en el evento Refresh

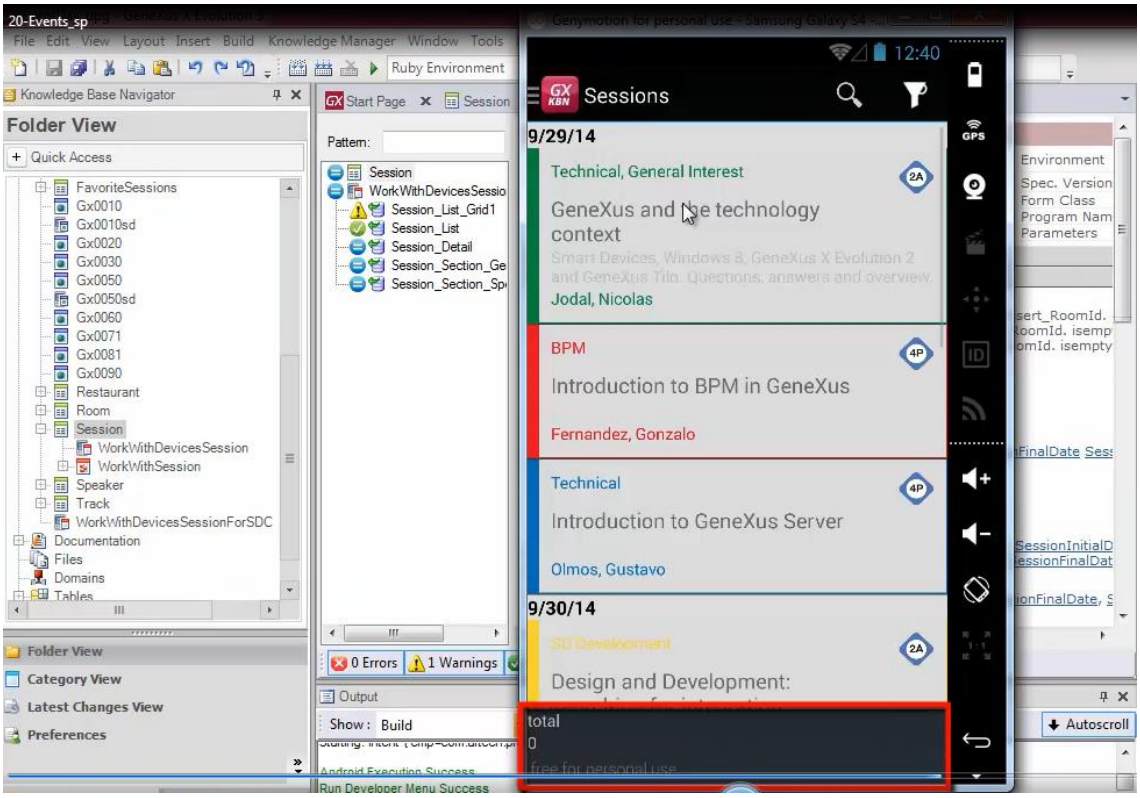

20-Events_sp
File Edit View Layout Insert Build Knowledge Manager Window Tools Help
Ruby Environment
Release
KnowledgeBase
Search
Knowledge Base Navigator
Folder View
Quick Access
FavoriteSessions
Gx0010
Gx0010sd
Gx0020
Gx0030
Gx0050
Gx0050sd
Gx0060
Gx0071
Gx0081
Gx0090
Restaurant
Room
Session
WorkWithDevicesSession
WorkWithSession
Speaker
Track
WorkWithDevicesSessionForSDC
Documentation
Files
Domains
Tables
Folder View
Category View
Start Page
Session *
Patterns usable in this object (underlined means pattern is applied)
Category
Work With for Web
Work With for Smart Devices
Apply this pattern on save
Level (Session)
List
Detail
Section (General)
Section (Speakers)
Refresh
13 endif
14
15 If SessionDescription.Length() > 145
16 &SessionDescription = substr(SessionDescript
17 Else
18 &SessionDescription = SessionDescription
19 EndIf
20 &total = &total + 1
21 Endevent
22
23
24
25
26 Event Refresh
27 &total = 0
28 Endevent
Layout
Rules
Events
Conditions
Variables
Structure
Web F...
Win Fo...
Rules
Events
Variables
Help
Documen...
Patterns *


20-Events_sp
File Edit View Layout Insert Build Knowledge Manager Window Tools Help
Ruby Environment
Release
KnowledgeBase
Search
Knowledge Base Navigator
Folder View
Quick Access
FavoriteSessions
Gx0010
Gx0010sd
Gx0020
Gx0030
Gx0050
Gx0050sd
Gx0060
Gx0071
Gx0081
Gx0090
Restaurant
Room
Session
WorkWithDevicesSession
WorkWithSession
Speaker
Track
WorkWithDevicesSessionForSDC
Documentation
Files
Domains
Tables
Folder View
Category View
Start Page
Session *
Patterns usable in this object (underlined means pattern is applied)
Category
Work With for Web
Work With for Smart Devices
Apply this pattern on save
Level (Session)
List
Detail
Section (General)
Section (Speakers)
Refresh
13 endif
14
15 If SessionDescription.Length() > 145
16 &SessionDescription = substr(SessionDescript
17 Else
18 &SessionDescription = SessionDescription
19 EndIf
20 &total = &total + 1
21 Endevent
22
23
24
25
26 Event Refresh
27 &total = 0
28 Endevent
F5
Layout
Rules
Events
Conditions
Variables
Structure
Web F...
Win Fo...
Rules
Events
Variables
Help
Documen...
Patterns *


20-Events_sp
Knowledge Base Navigator
Folder View
Quick Access
Start Page
Session
Pattern:
Ruby Environment
12:40
Knowledge Base Navigator
Smart Devices Applications
Name: EventDay_upg - EventDay
Uri: http://app2.genexusx.com/
Id0034a862cf650187023285188cfb79b6/
0 Errors
1 Warnings
Output
Show: Build
Folder View
Category View
Latest Changes View
Preferences
Autoscroll

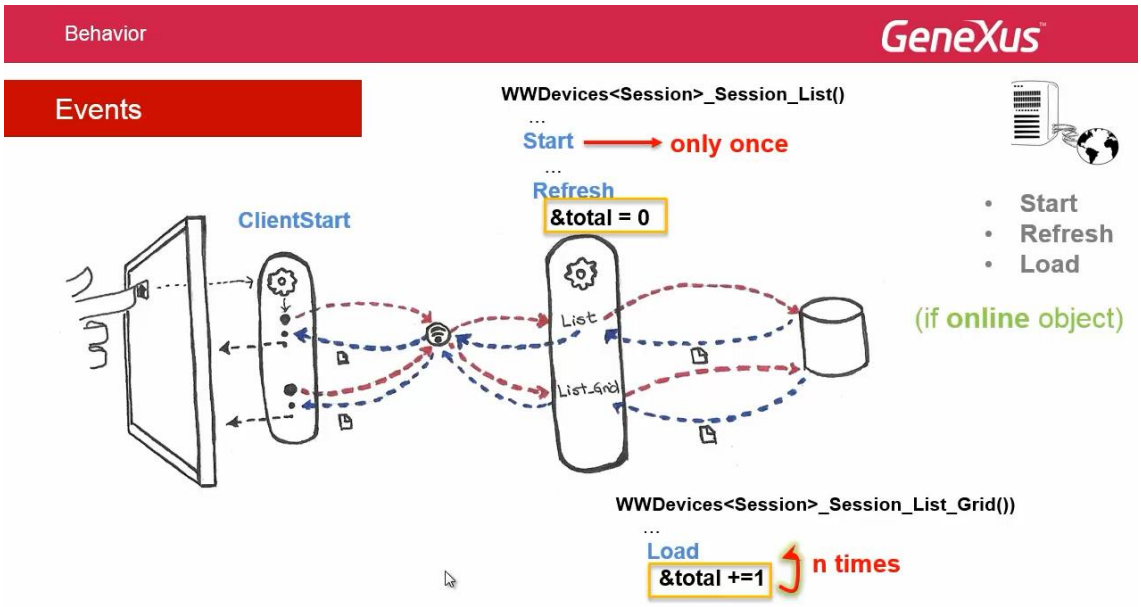

20-Events_sp
Knowledge Base Navigator
Folder View
Quick Access
Start Page
Session
Pattern:
Ruby Environment
12:40
Welcome
Eventday
0 Errors
1 Warnings
Output
Show: Build
Folder View
Category View
Latest Changes View
Preferences



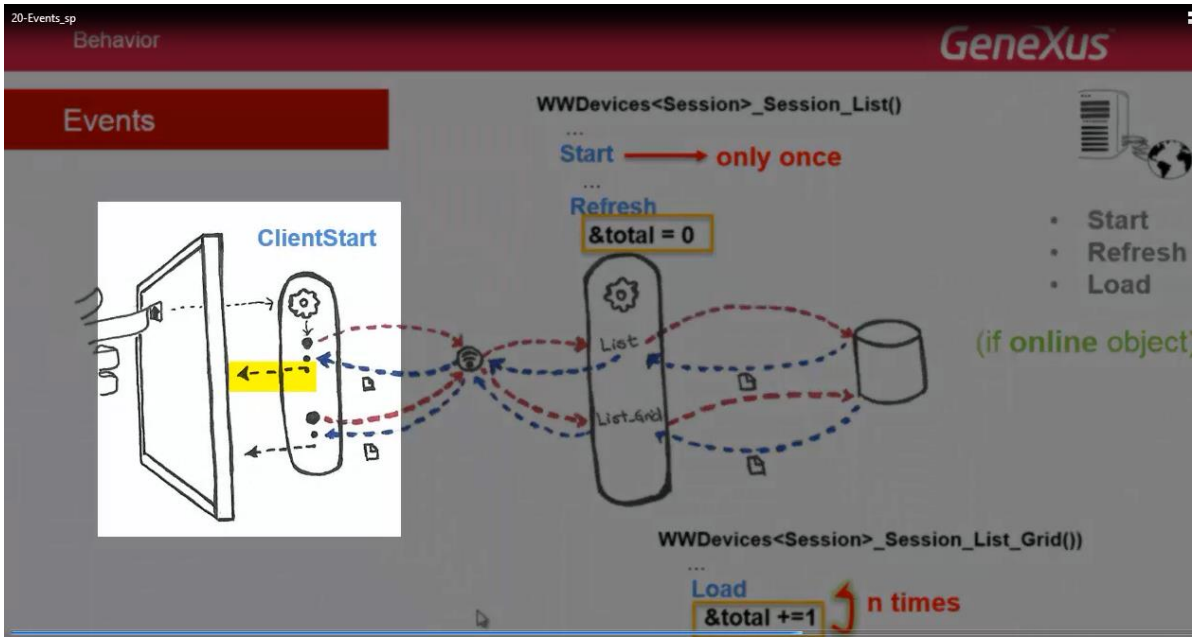


Pero… nos está mostrando cero… ¿por qué?

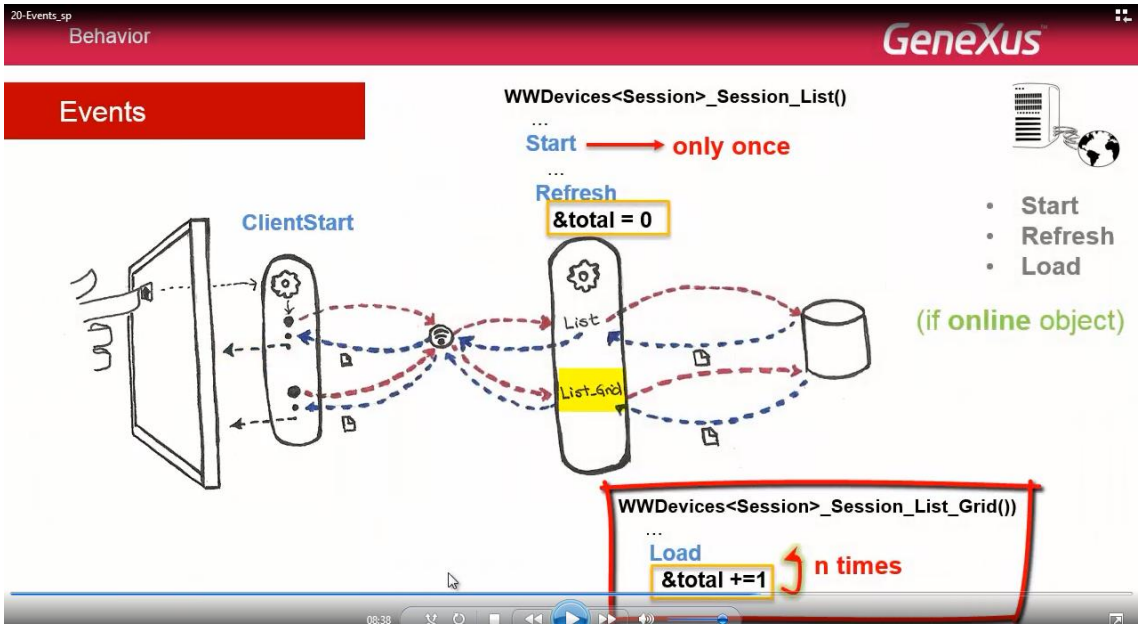
Aquí podemos ver esquemáticamente lo que hicimos.



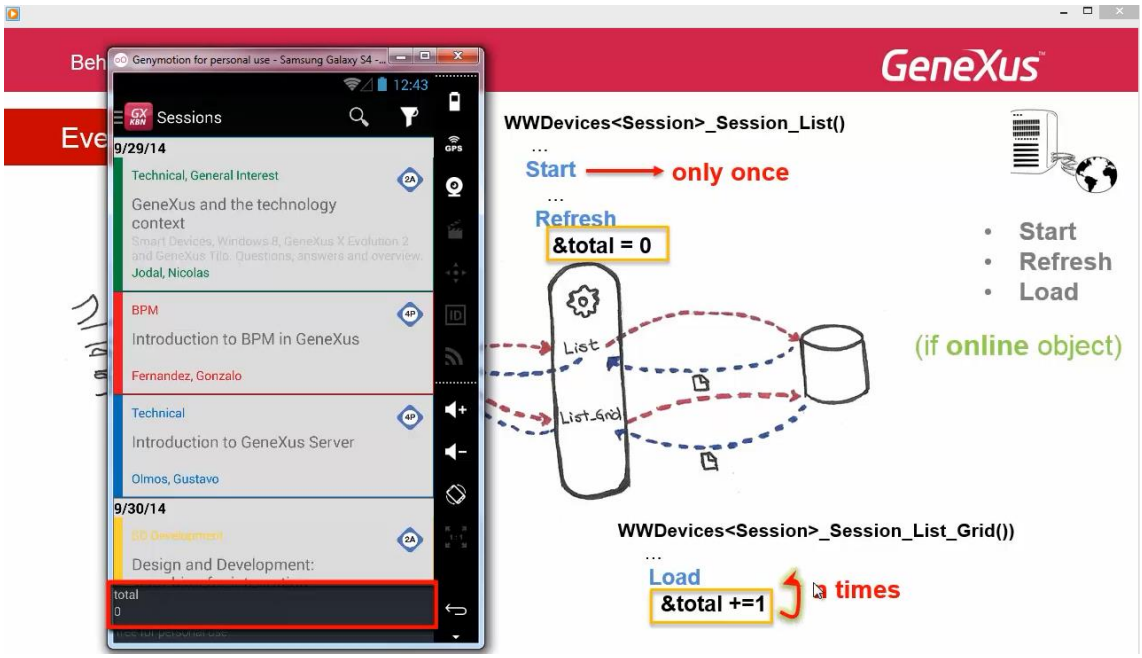
El problema es que programamos este panel como si fuera un web panel, ignorando el hecho de que la pantalla correspondiente a la parte fija, será dibujada en el dispositivo

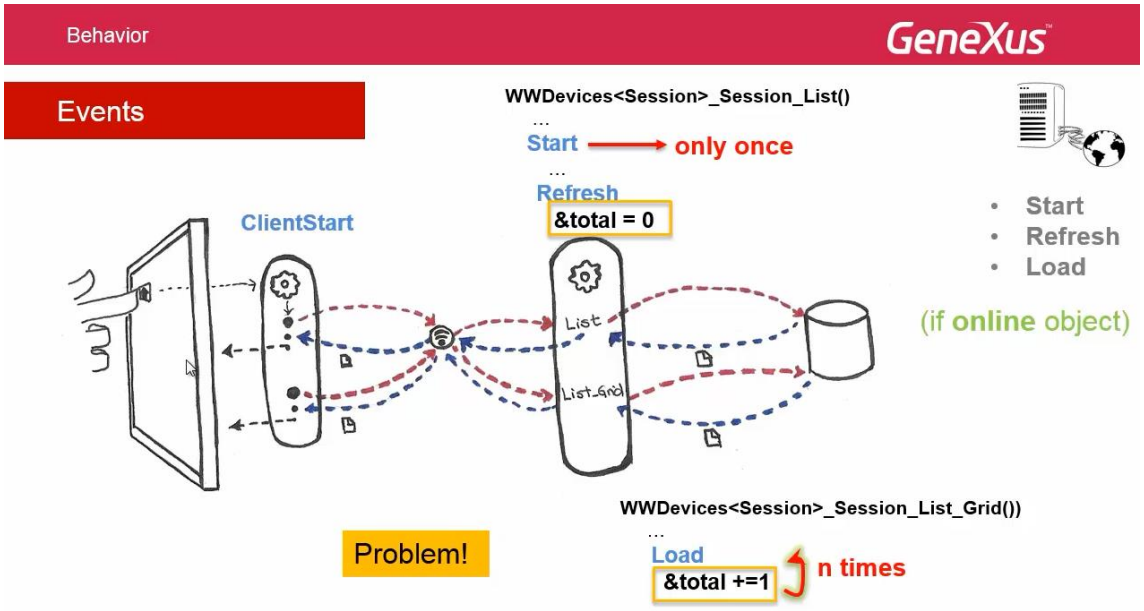


antes de llamar al data provider que devolverá las líneas del grid, para dibujarlo luego en la pantalla



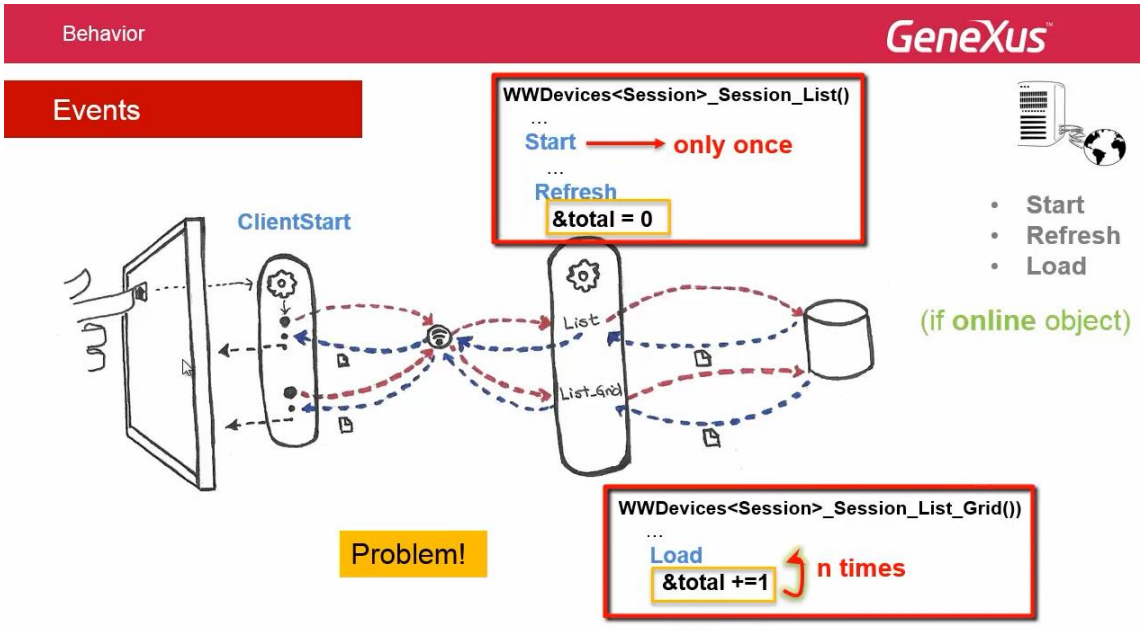
Por tanto al tiempo de mostrarse la variable &total, todavía no se habrá invocado las N veces al evento Load que la incrementa… y mucho menos se habrá dibujado el grid…





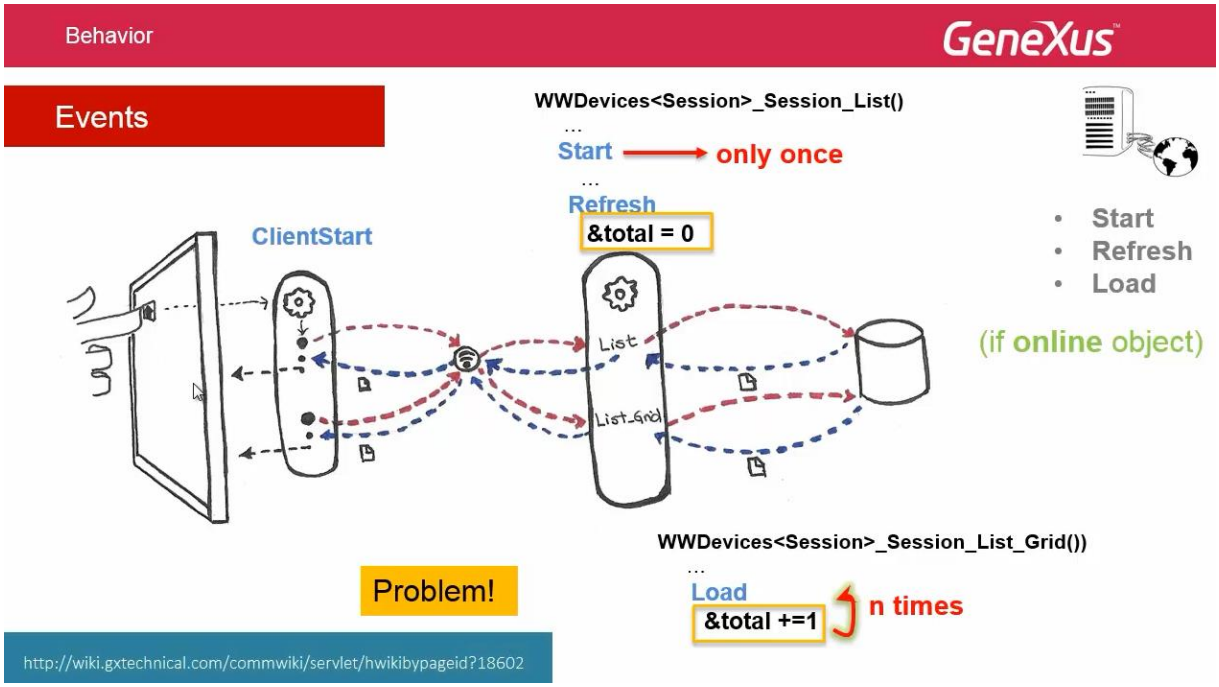
Obsérvese que esa es la razón del problema que aparece..

No el hecho de que los eventos Refresh y Load estén separados en 2 programas (2 data providers independientes)



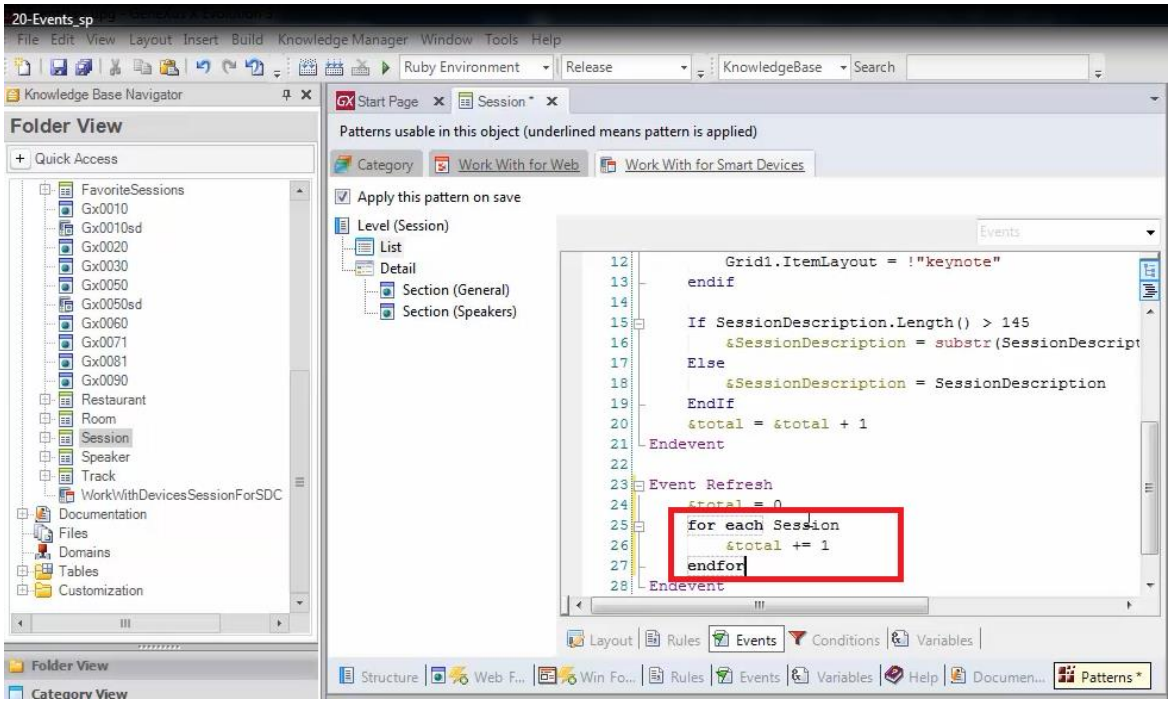
A los efectos del desarrollador, esta separación es transparente y lo que se haga en el Refresh será visto por el Load.

Existe adicionalmente otro problema correspondiente al caching, que puede repasar en nuestro wiki aquí

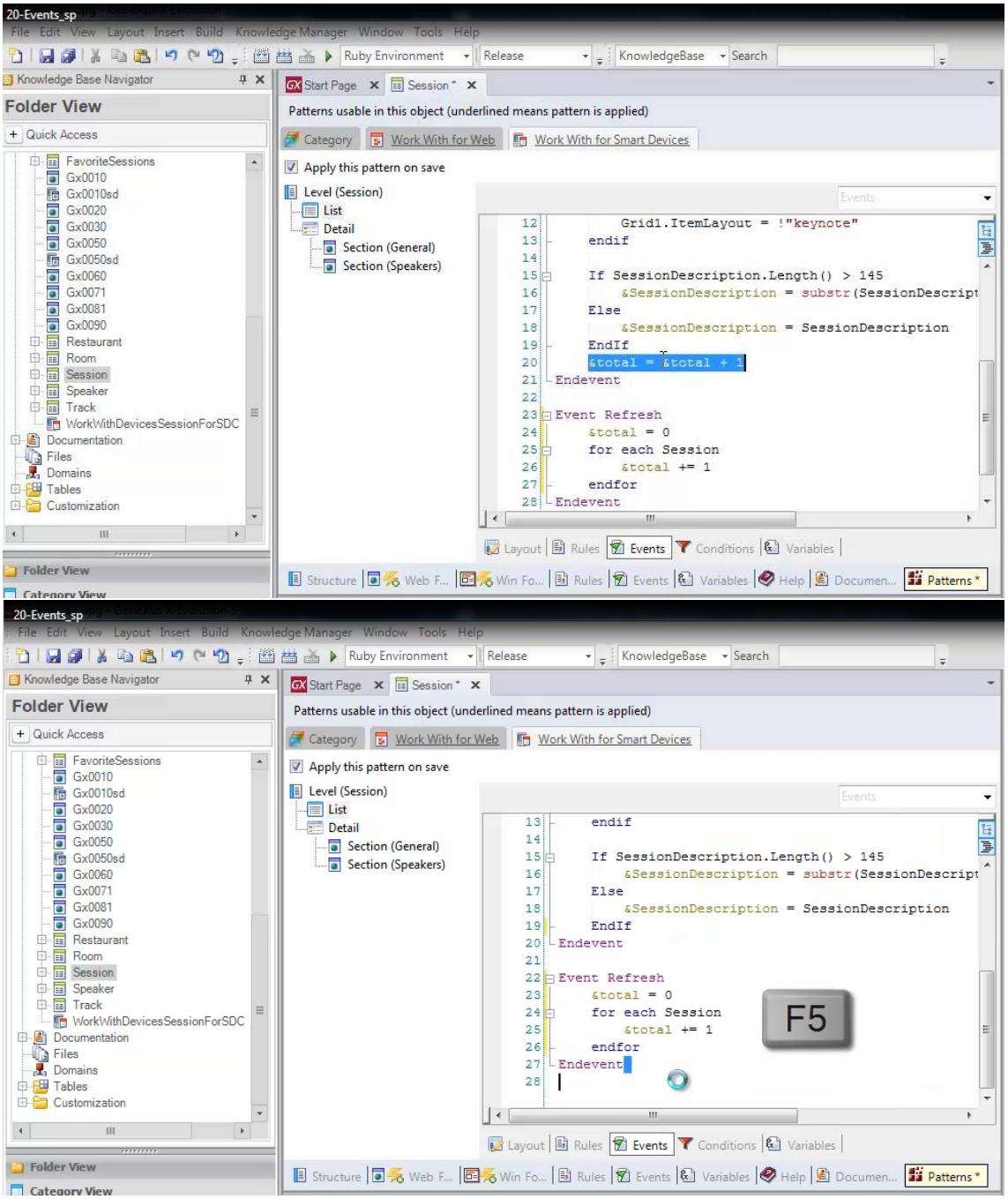


Por tanto, una solución para conseguir mostrar el nro de registros, es dentro del evento Refresh, programar un For each que los calcule.

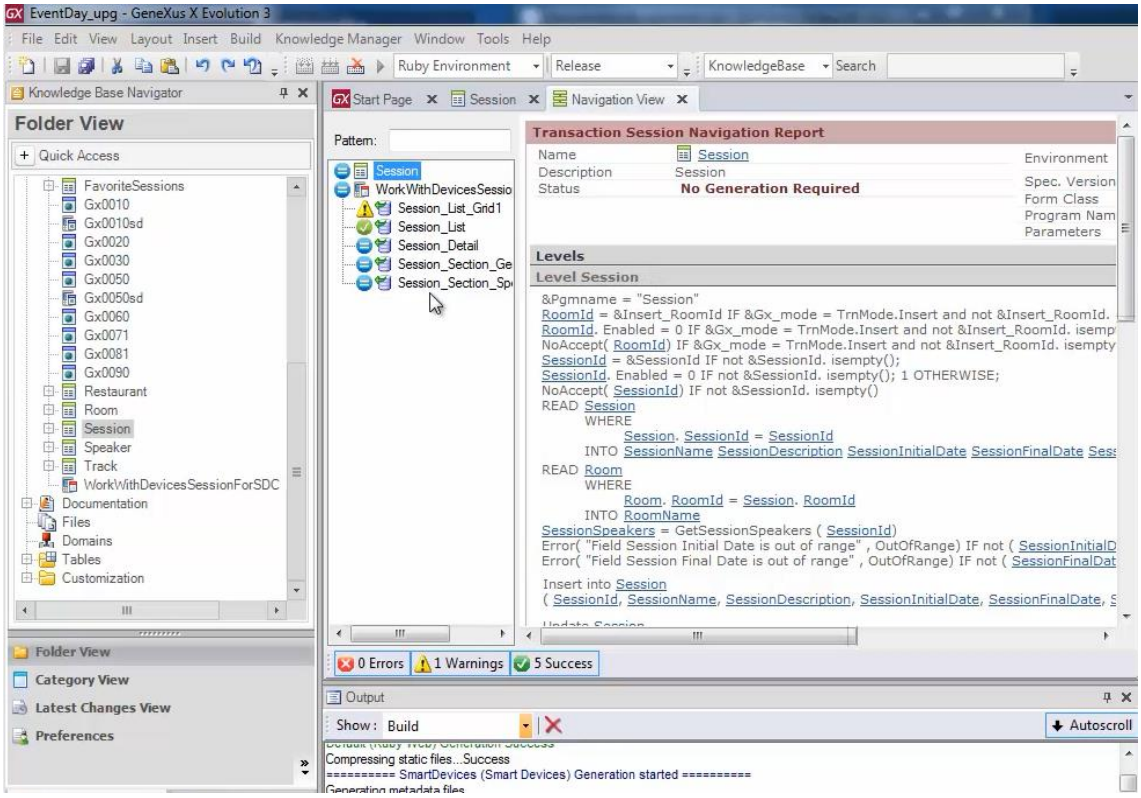
For each que tendrá como tabla base, la tabla asociada al primer nivel de la transacción Session y que para registro de esa tabla, al valor de la variable &total, le sumará 1.



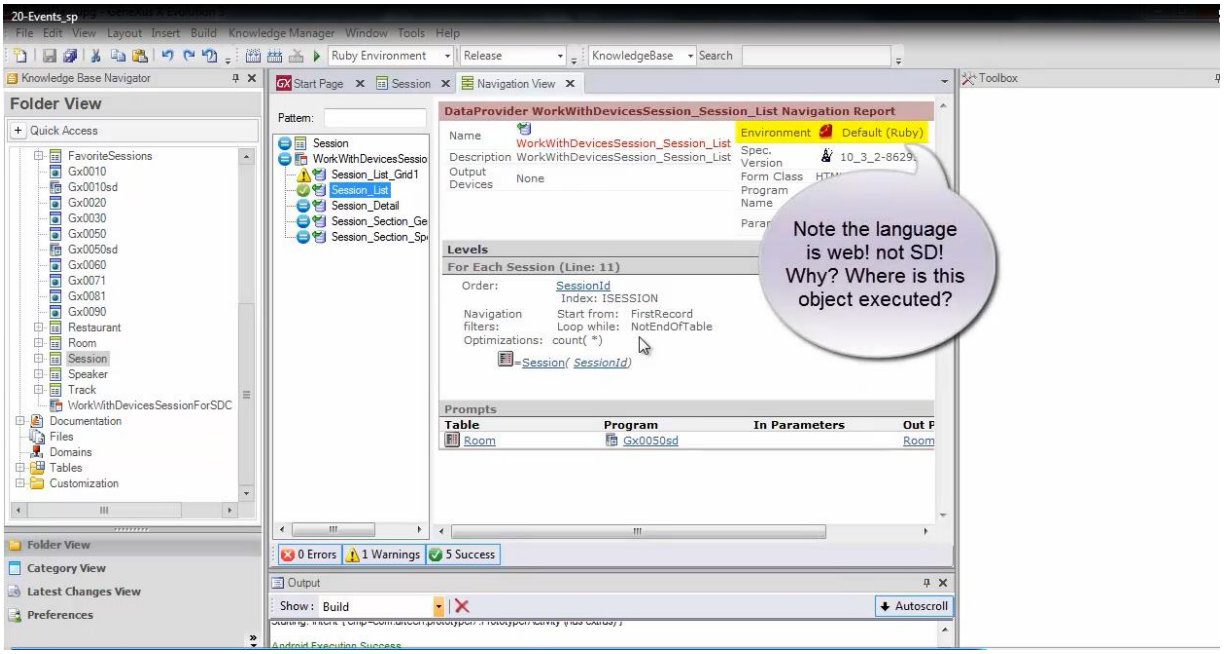
Eliminamos esta asignación del evento Load… y hagamos F5..



Si observamos los listados de navegación

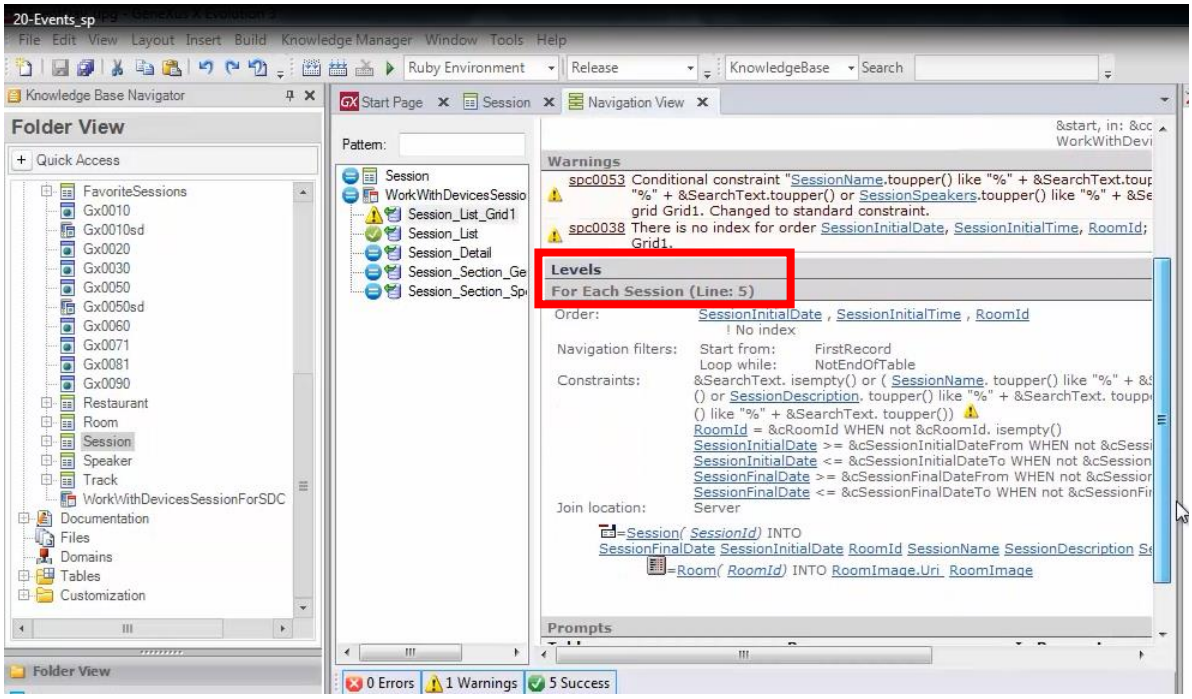


vemos que tenemos para el data provider Session_List correspondiente a la parte fija del List, el For each que acabamos de programar, que va a recorrer entonces la tabla Session contando los registros.



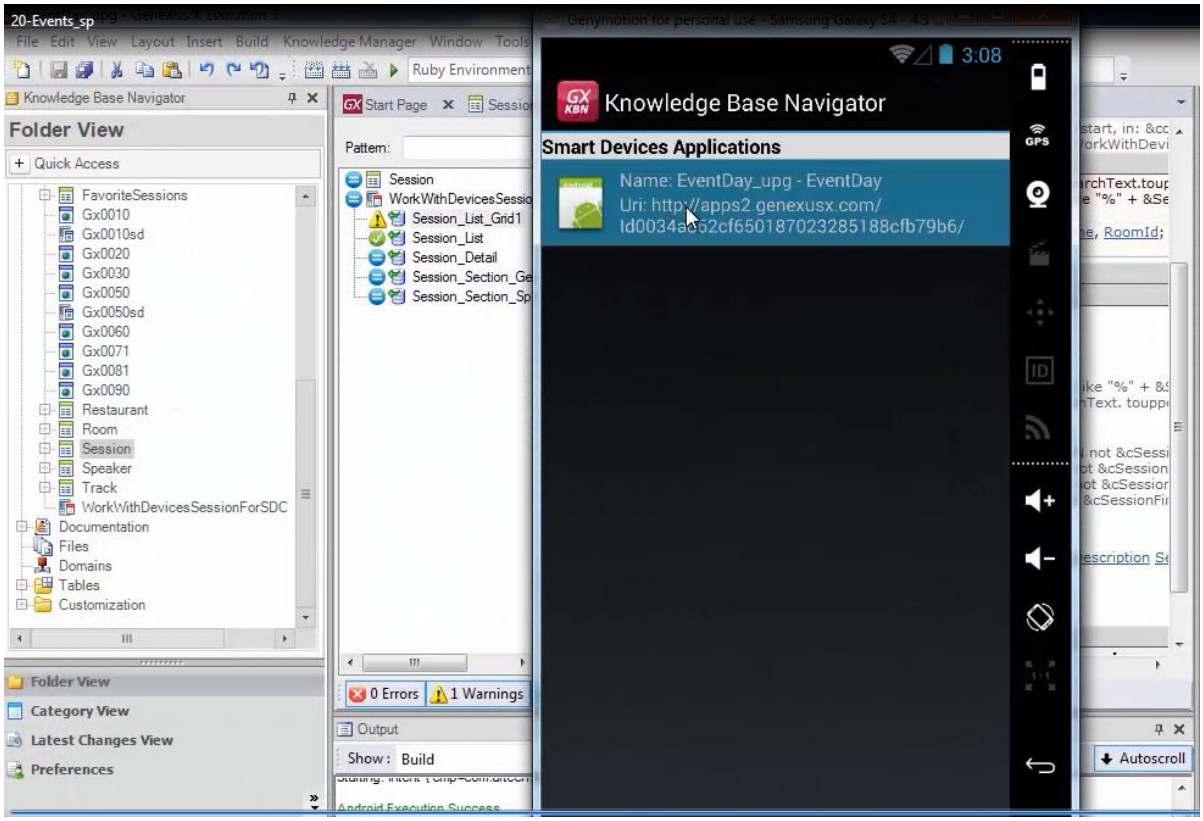
Luego de dispararse este data provider, se caragrá la parte fija… y allí entonces la variable &total sí tendrá el valor deseado.

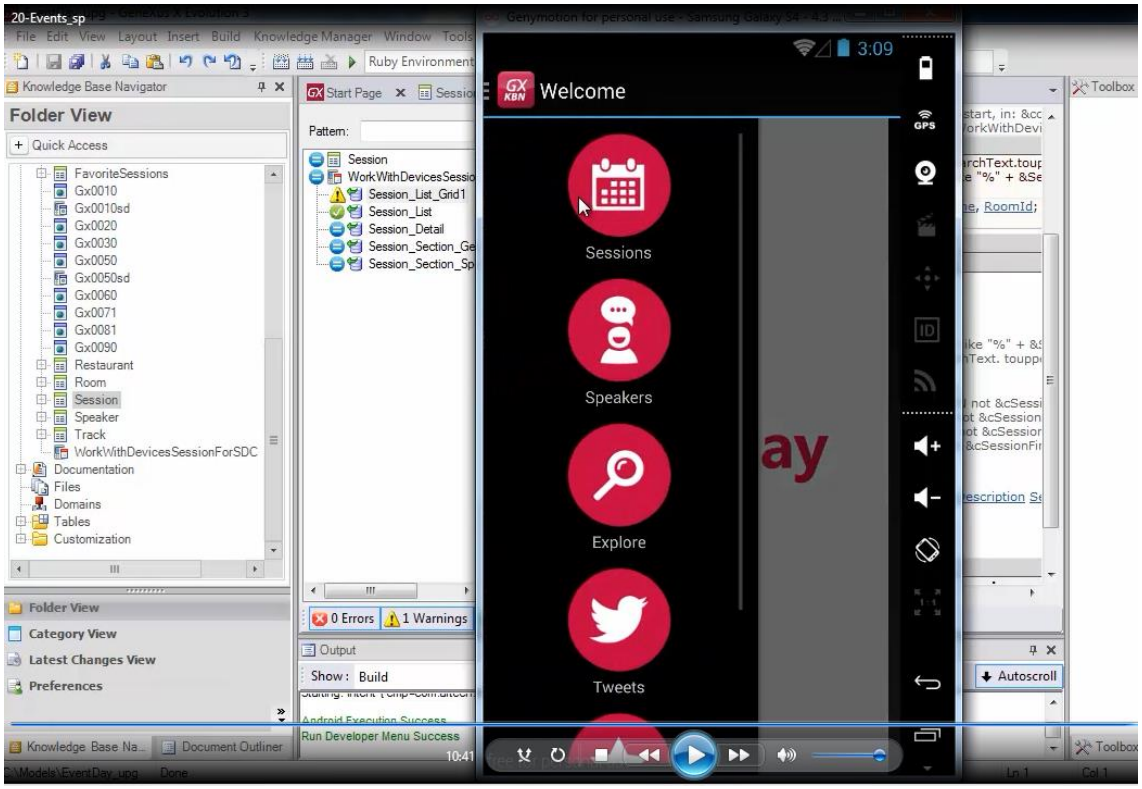
Observemos el otro data provider, el que corresponde a las líneas.



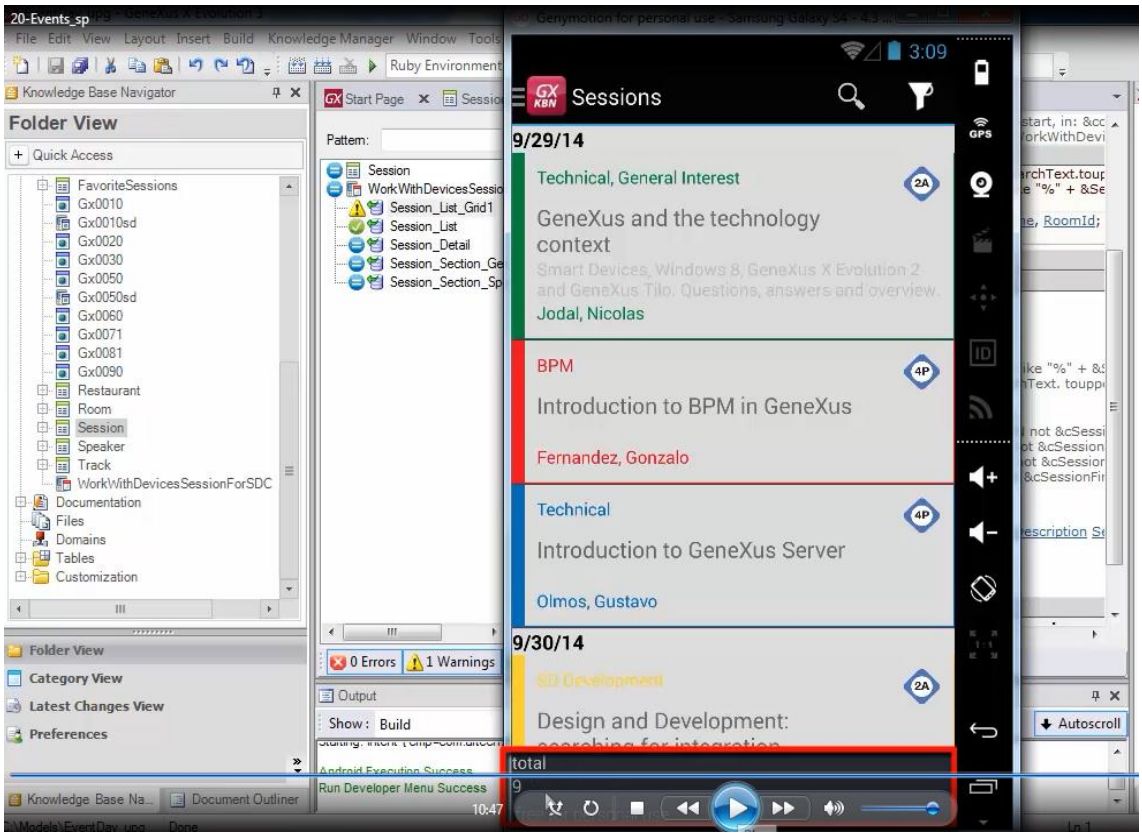
también yendo sobre la tabla Session.

Veámoslo en ejecución



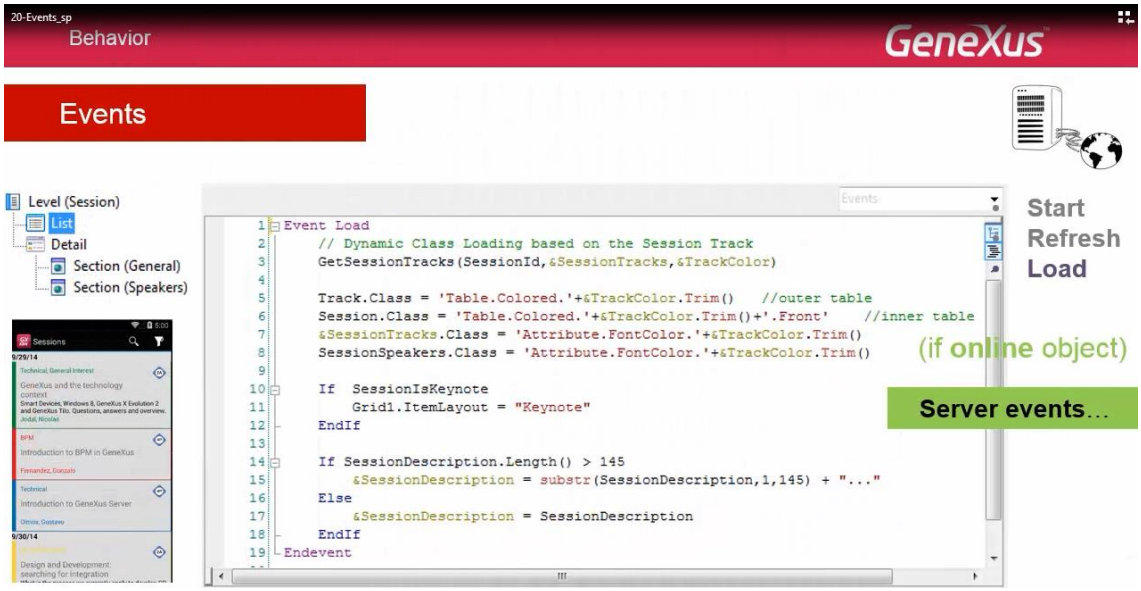


Ahora la variable &total está mostrando el valor 9



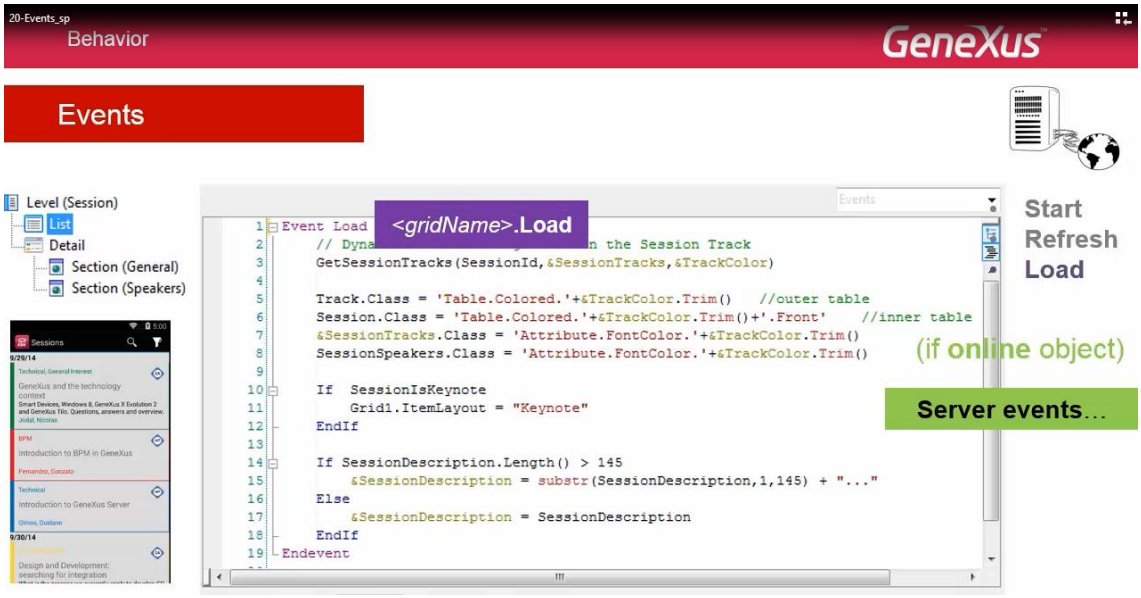
que coincide exactamente con la cantidad de registros contenidos en la tabla.

El evento Load del sistema



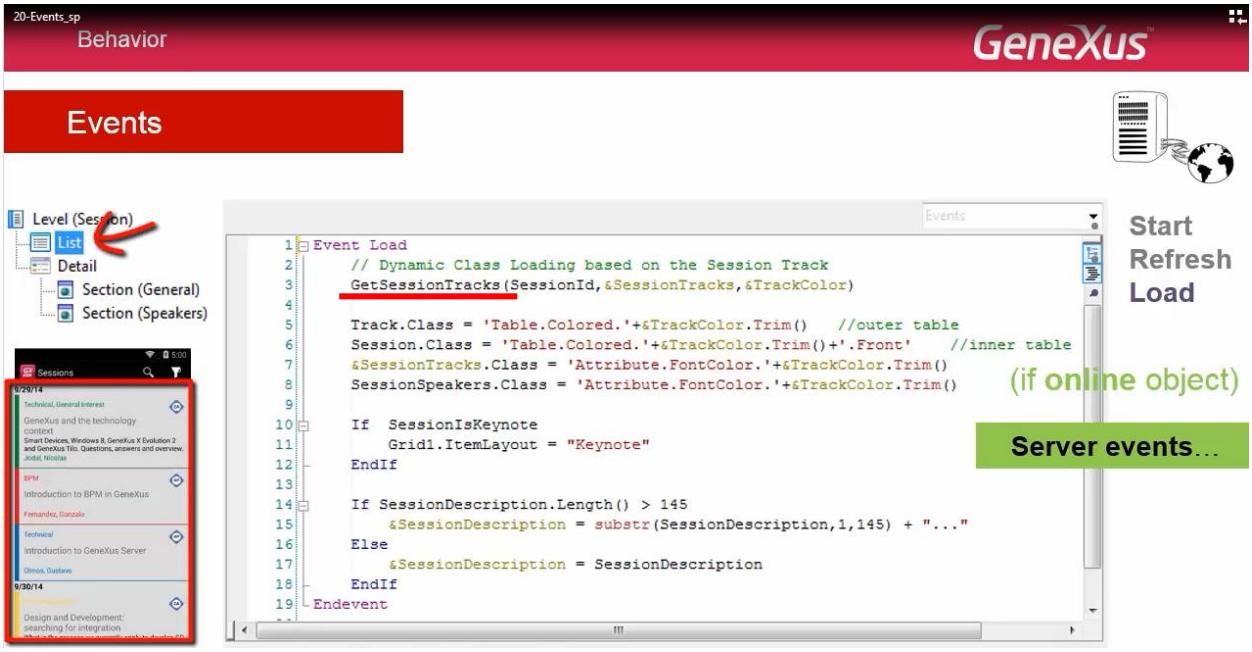
no requiere ninguna consideración especial, puesto que es análogo al Load de un web panel

Como se pueden incluir varios grids con tabla base en el mismo layout, al igual que con web panels cuando tenemos más de 1 grid, será necesario especificar el Load de qué gid estamos programando, escribiendo en lugar de Event Load,



Event - nombre de grid – punto – Load

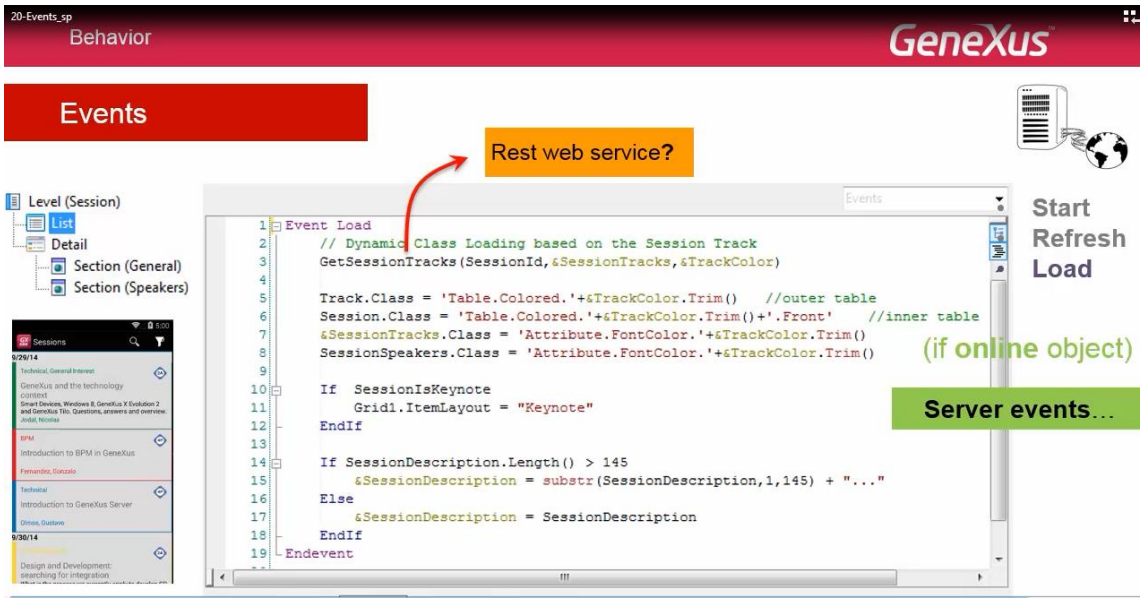
Aquí mostramos el ejemplo del Load para cargar los items del grid del List

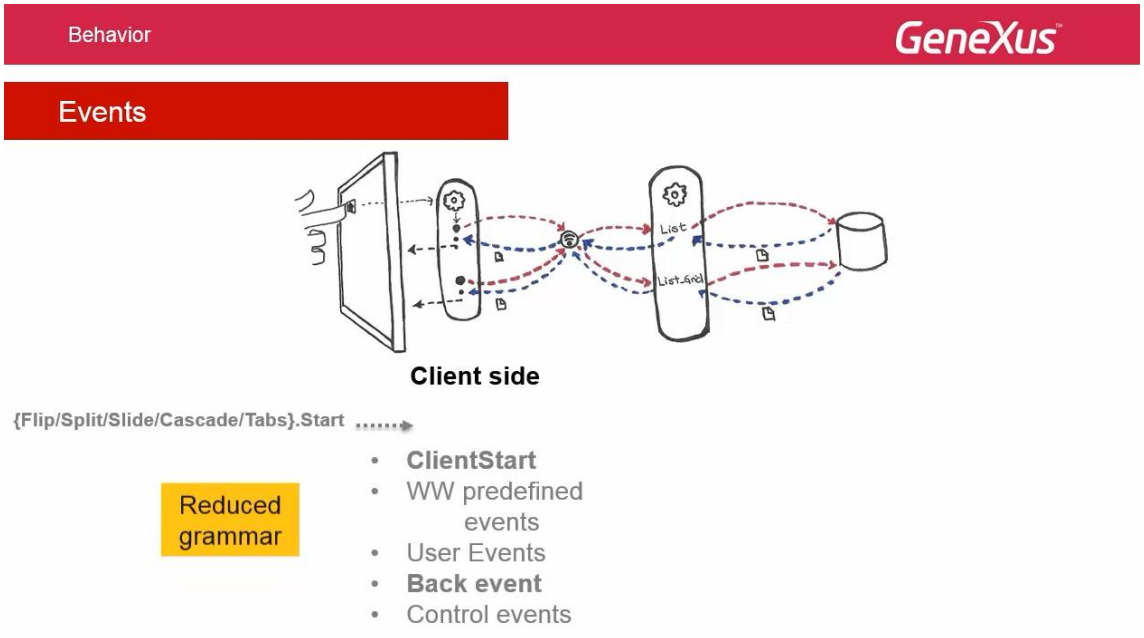


del Work With Devices Session

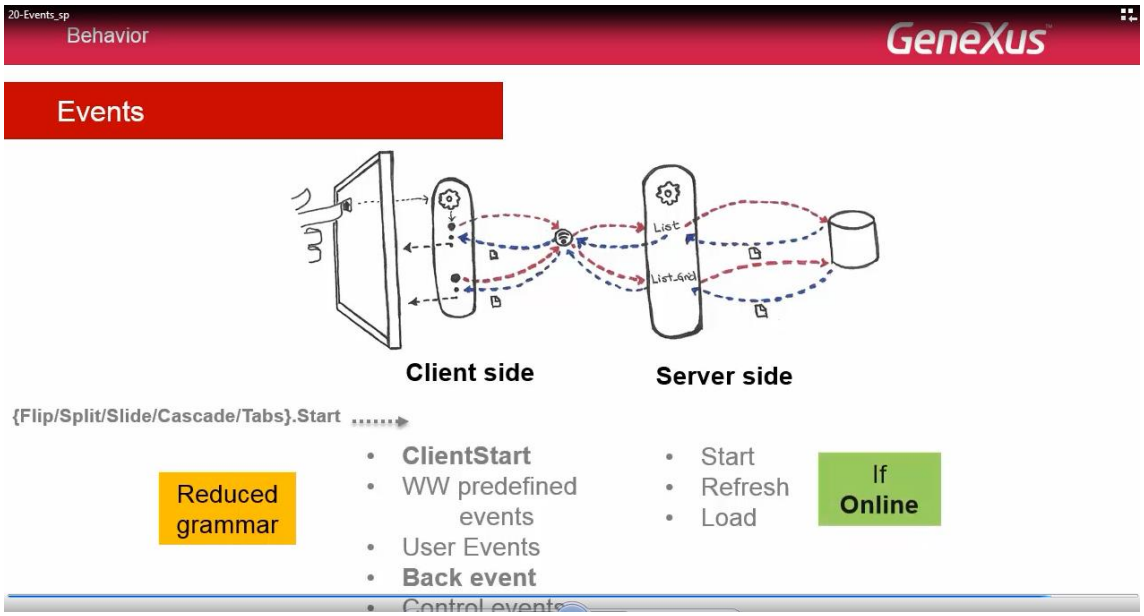
Este evento se ejecuta en el servidor, por lo que tiene a disposición todos los atributos de la tabla extendida para ser utilizados

Observe que el procedimiento GetSessionTracks no tiene que estar expuesto como servicio Rest, dado que se está invocando desde el propio server





Hemos visto entonces, los eventos que se ejecutan en el cliente y los que se ejecutan en el servidor



Es importante diferenciarlos dado que lo que podemos programar en los eventos del cliente, sigue una gramática un poco más reducida que lo que podemos hacer en el servidor como veremos específicamente en un video aparte.

Adelantamos que esto no afectará a la implementación de una aplicación offline. Es decir, en cuanto a la gramática, ud. programará los eventos de la misma manera si su aplicación es online u offline. En ese sentido es transparente.

En los videos siguientes, continuaremos estudiando aspectos relacionados a los eventos, como las apis para entre otras cosas poder integrarnos con las funcionalidades nativas del dispositivo, el orden de ejecución de los eventos, la gramática de los eventos del cliente, etc.


18-INDEX-Behavior_sp
LIVE
GeneXus™
TRAINING PROGRAM
training.genexus.com