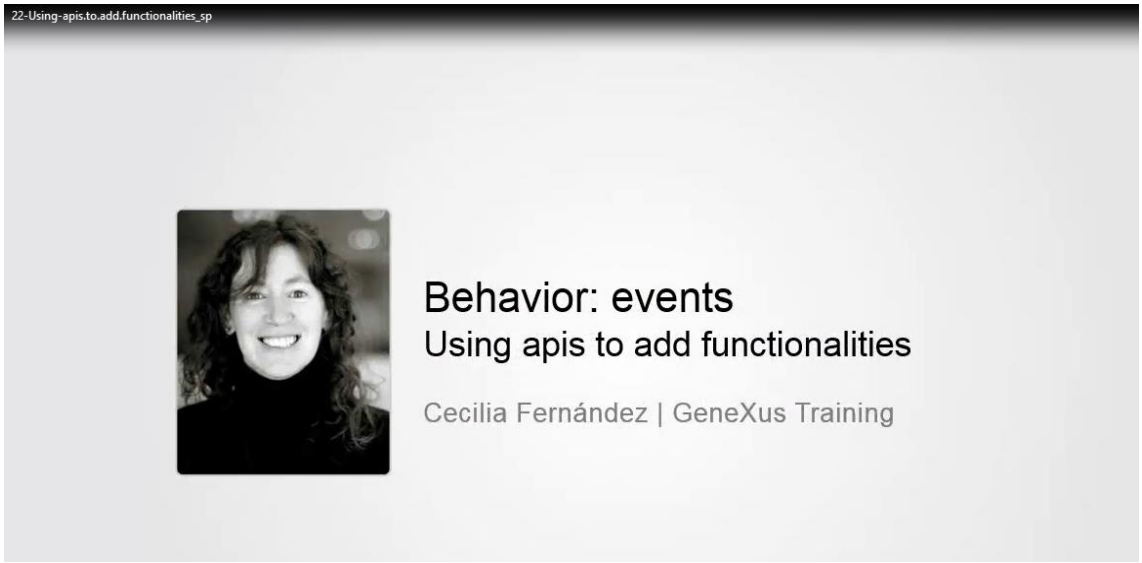
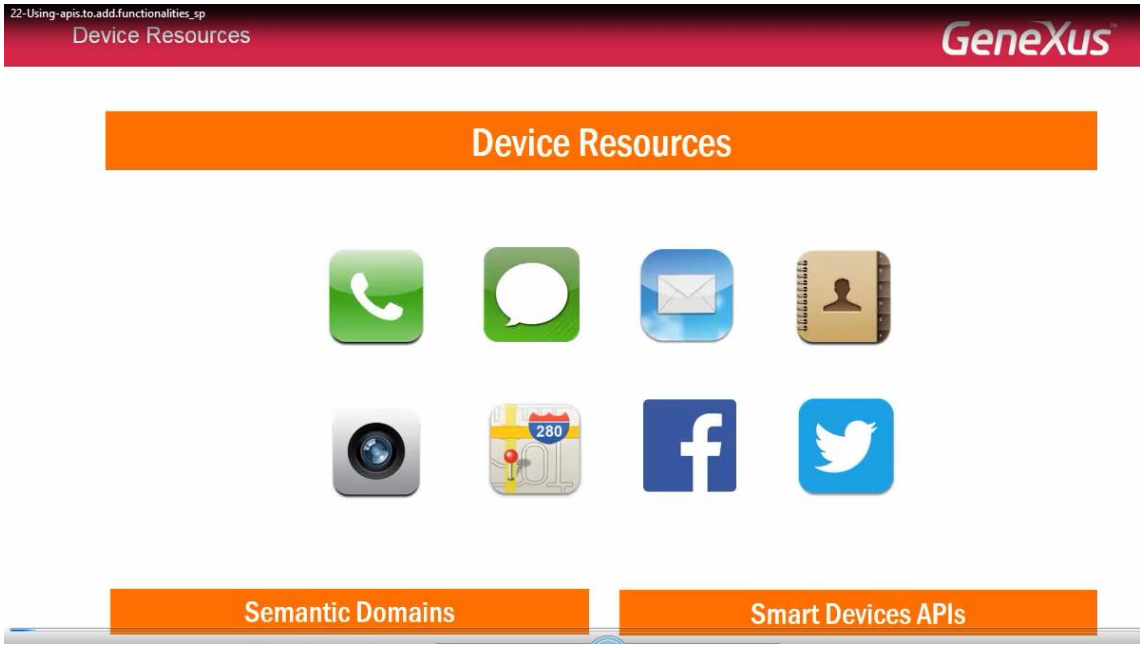
Using Apis to add functionalities

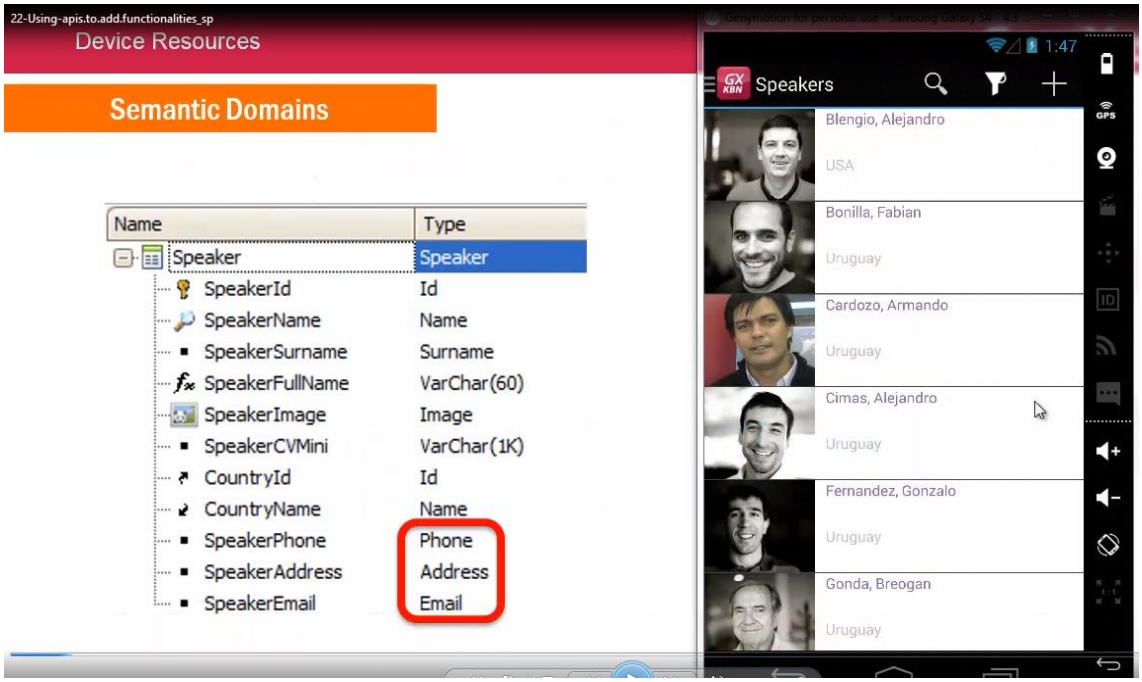




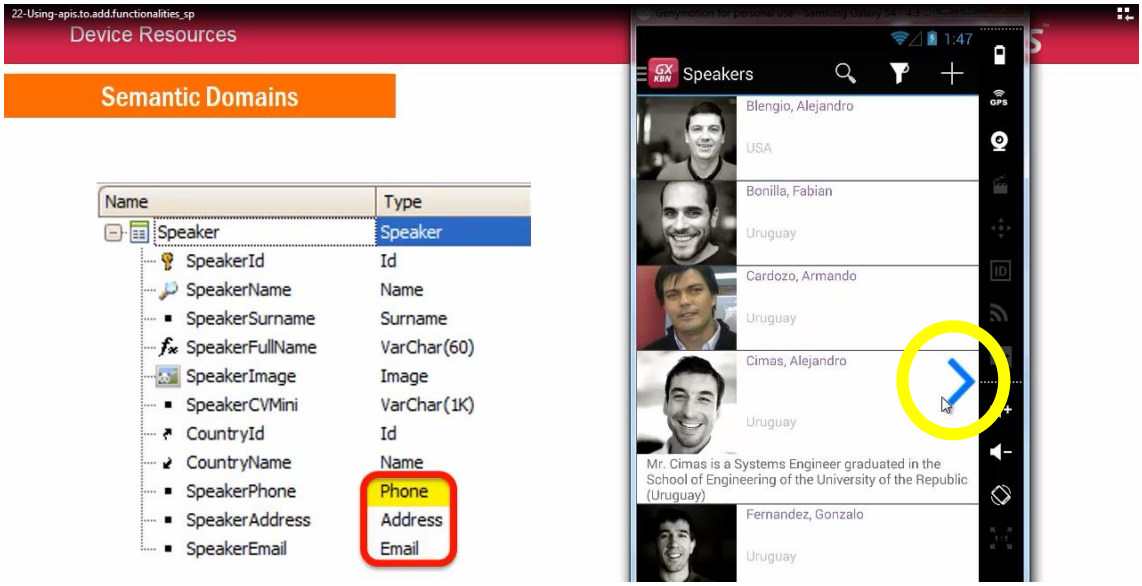
Las aplicaciones para dispositivos móviles requerirán integrarse con recursos del dispositivo, tanto físicos como lógicos.

Por ejemplo deberán poder realizar llamadas telefónicas, enviar mensajes a través de los programas de chat o de mail instalados, acceder y usar la libreta de direcciones, la cámara, el GPS junto con las aplicaciones de mapas, así como interactuar con Facebook y Twitter.

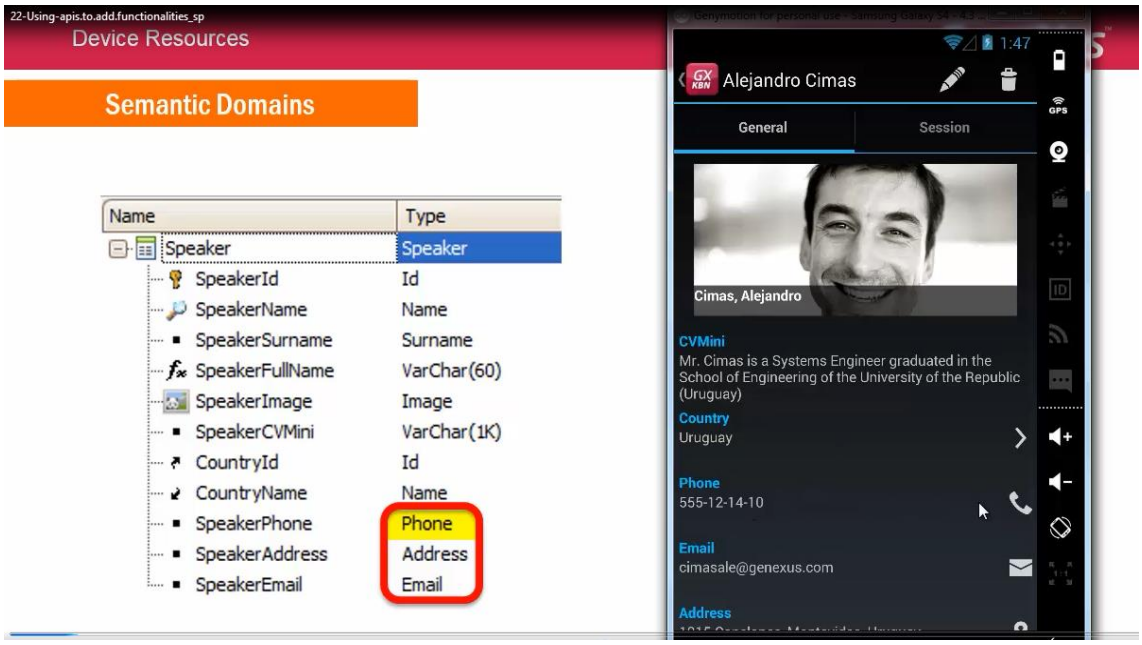
Algunas de estas funcionalidades se implementaban automáticamente a través de los dominios semánticos… como por ejemplo…



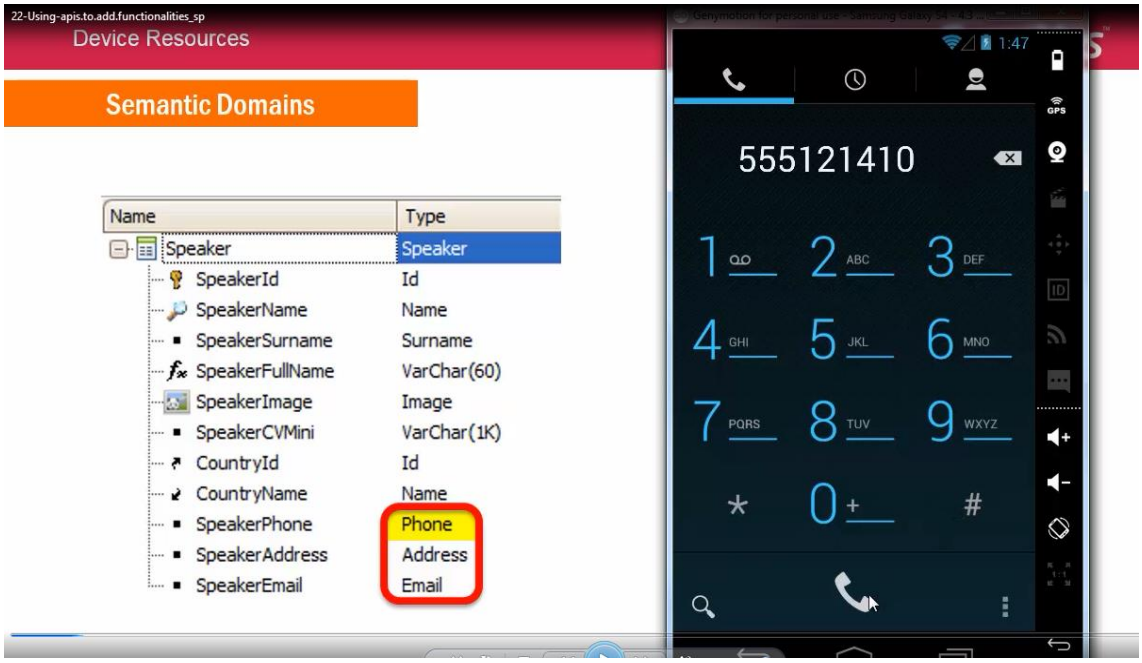
teniendo el dominio semántico Phone para el atributo SpeakerPhone



si vamos al detalle de un orador,

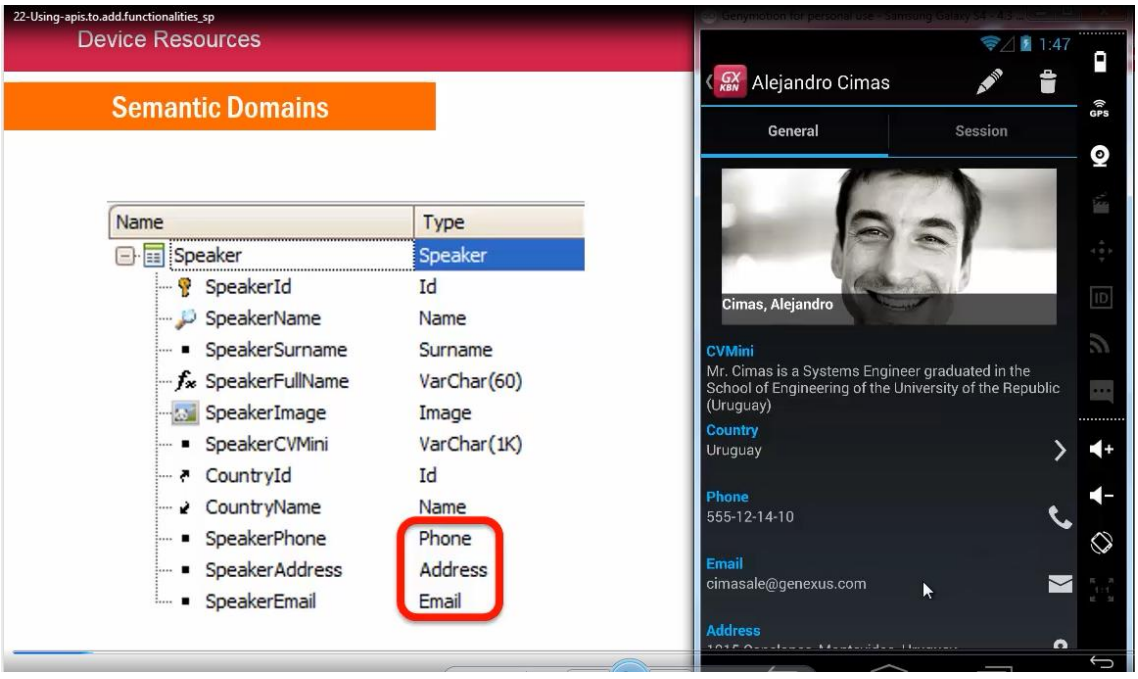


y hacemos TAP sobre su campo Phone

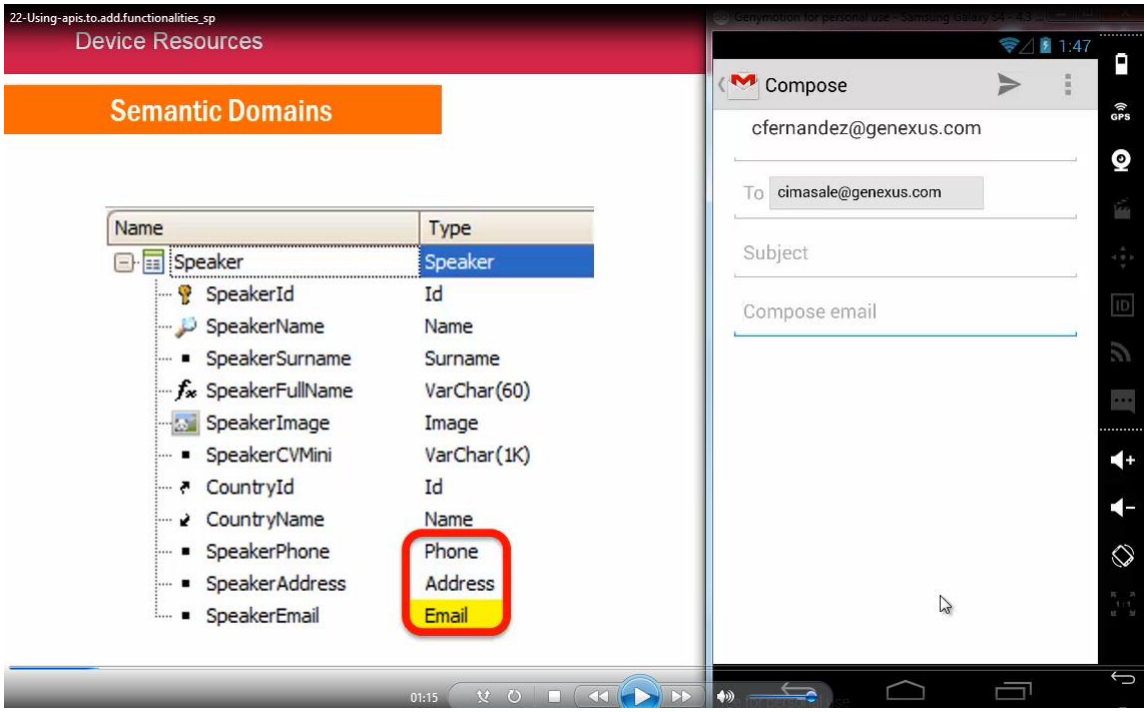


se nos abre la aplicación instalada en el dispositivo, para hacer la llamada telefónica.

Por otro lado, si hacemos TAP sobre el campo Email, de dominio semántico: Email

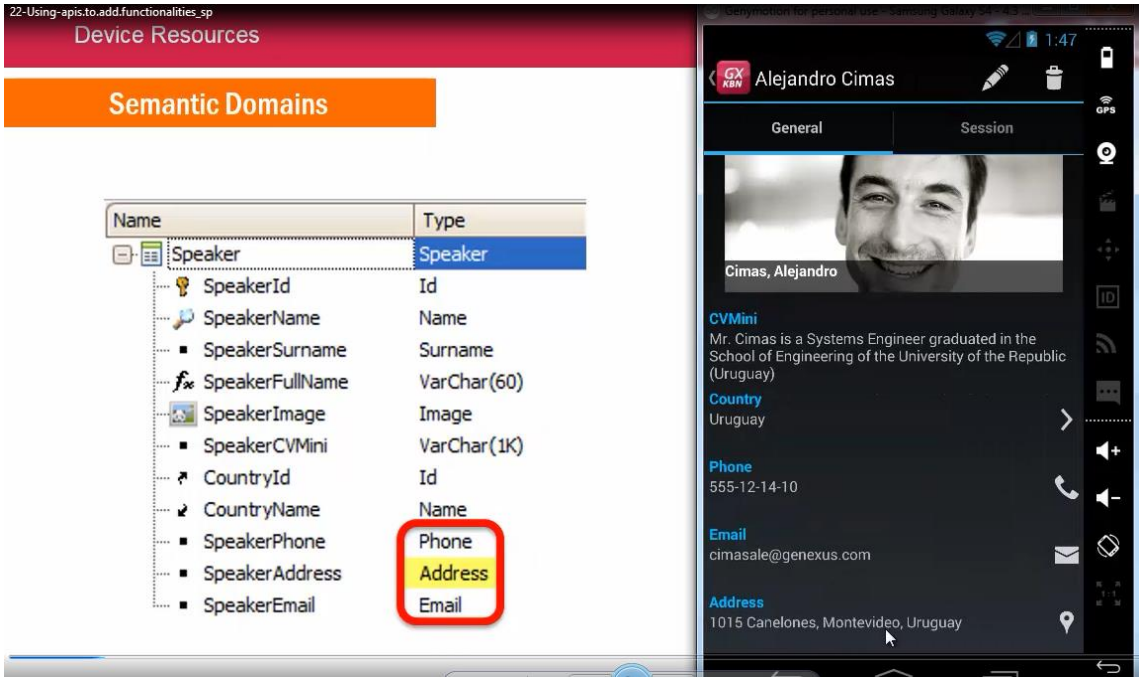


se nos abre la aplicación instalada en el dispositivo

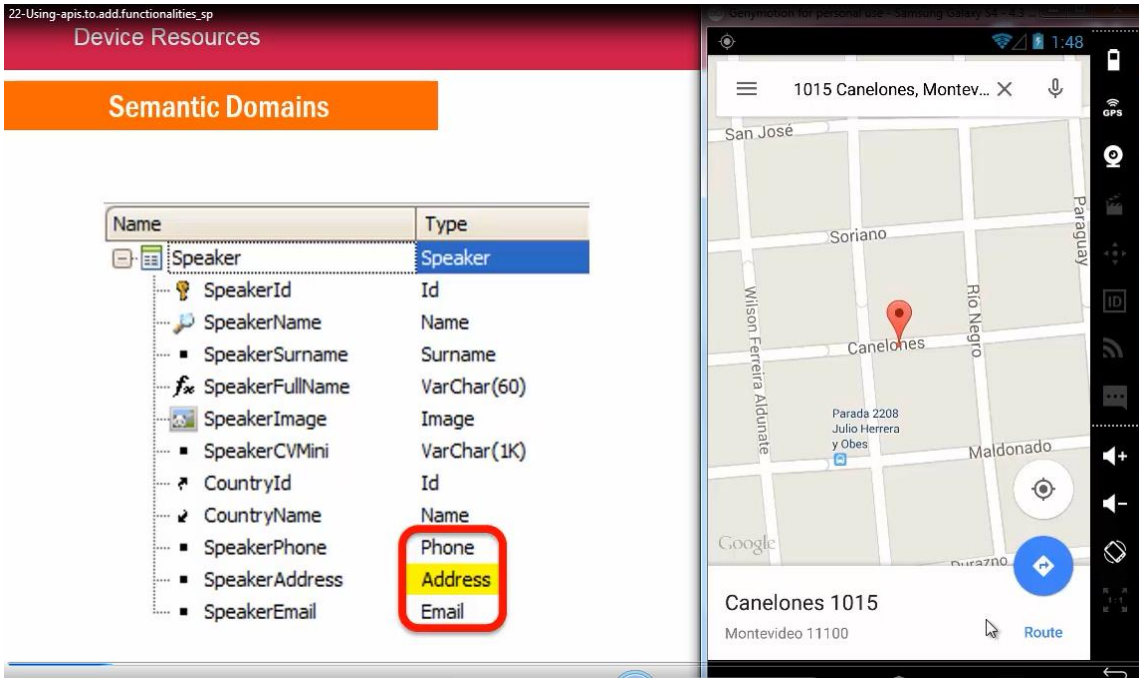


para escribir y enviar un email al orador correspondiente.

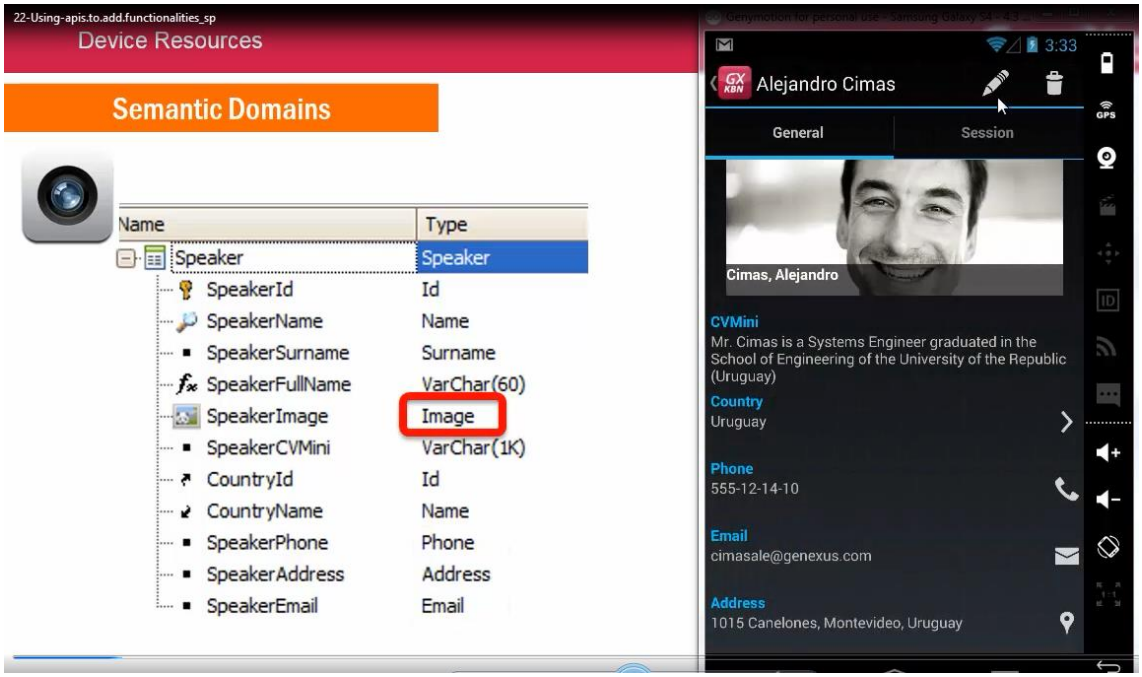
Y por último, si vamos al campo Address y hacemos TAP sobre la dirección



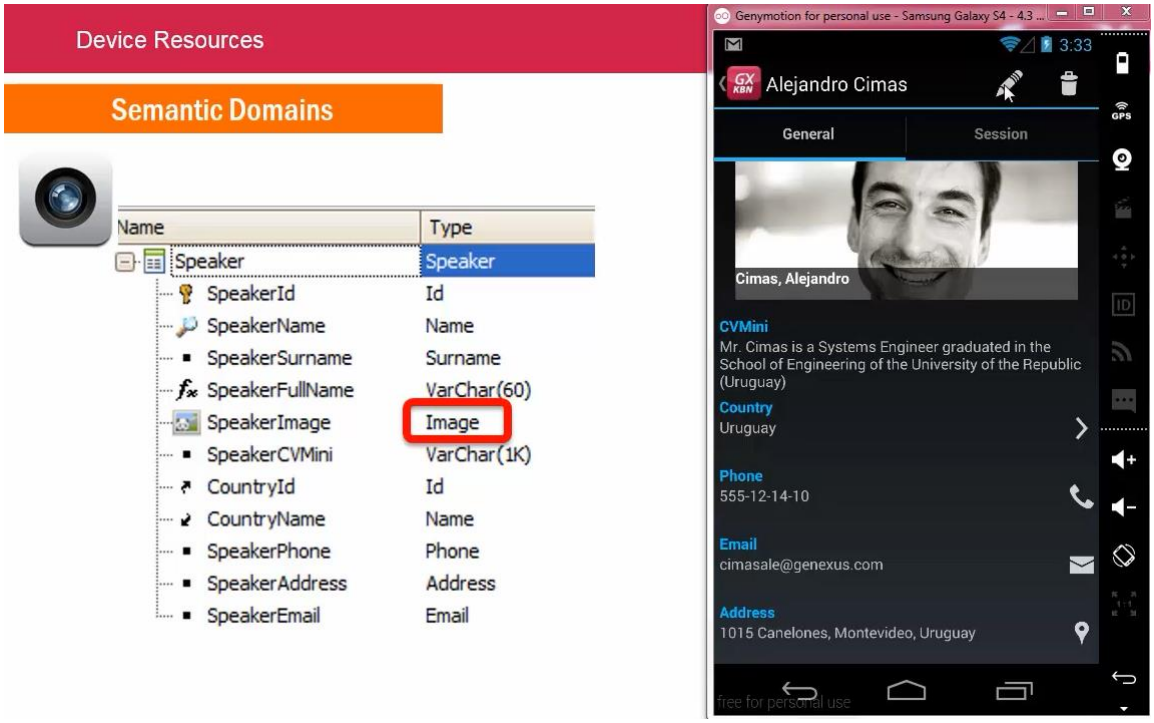
se nos abren los mapas instalados en el dispositivo



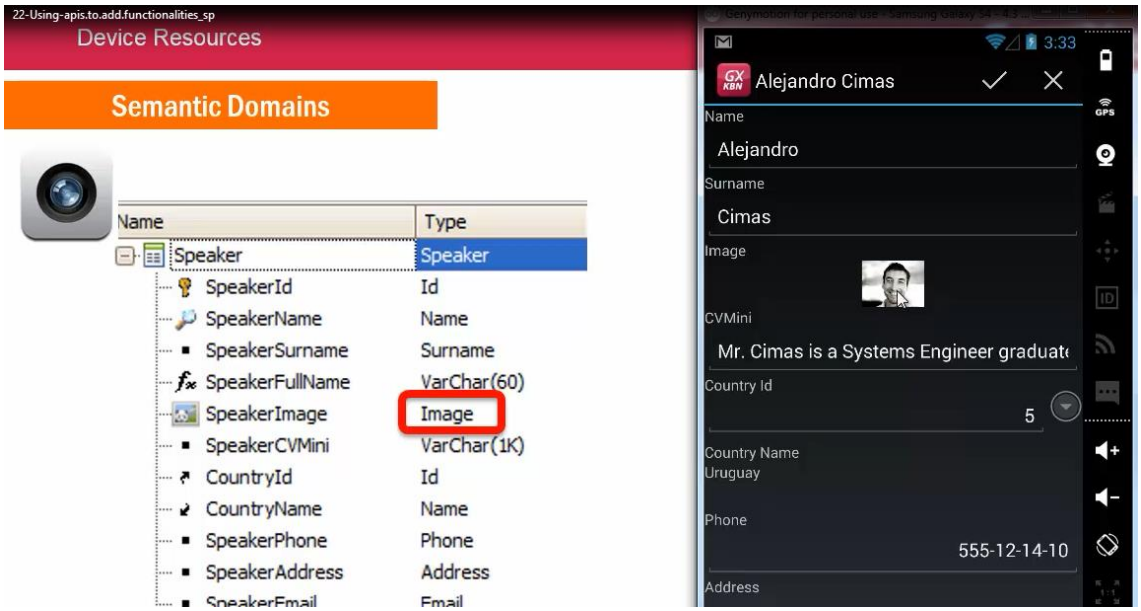
mostrándonos entonces la dirección en el mapa.



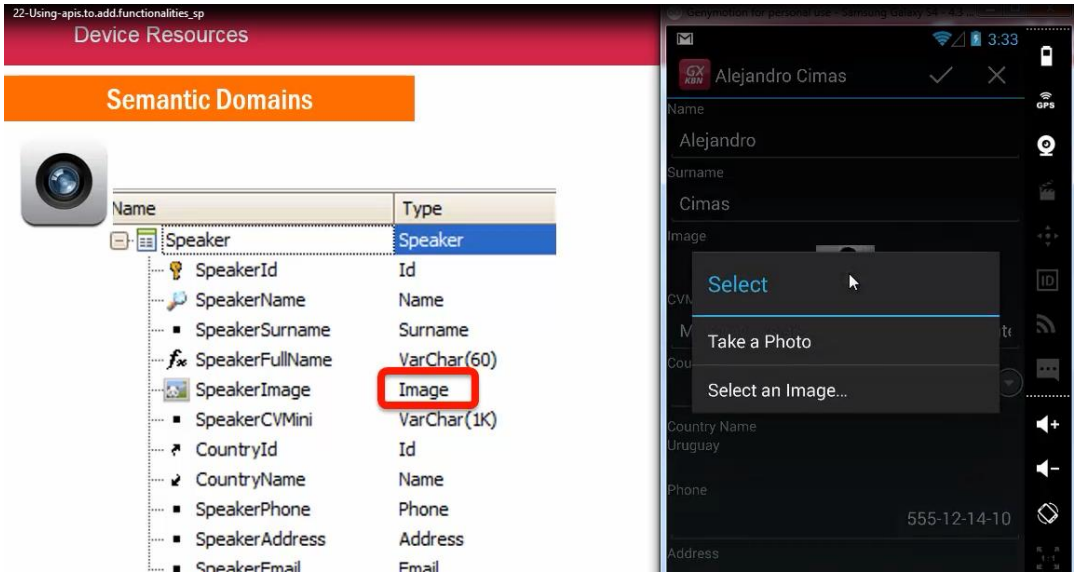
Por otro lado, sólo con tener el tipo de datos Image para un atributo, esto hará que en el Detail



cuando editemos la información

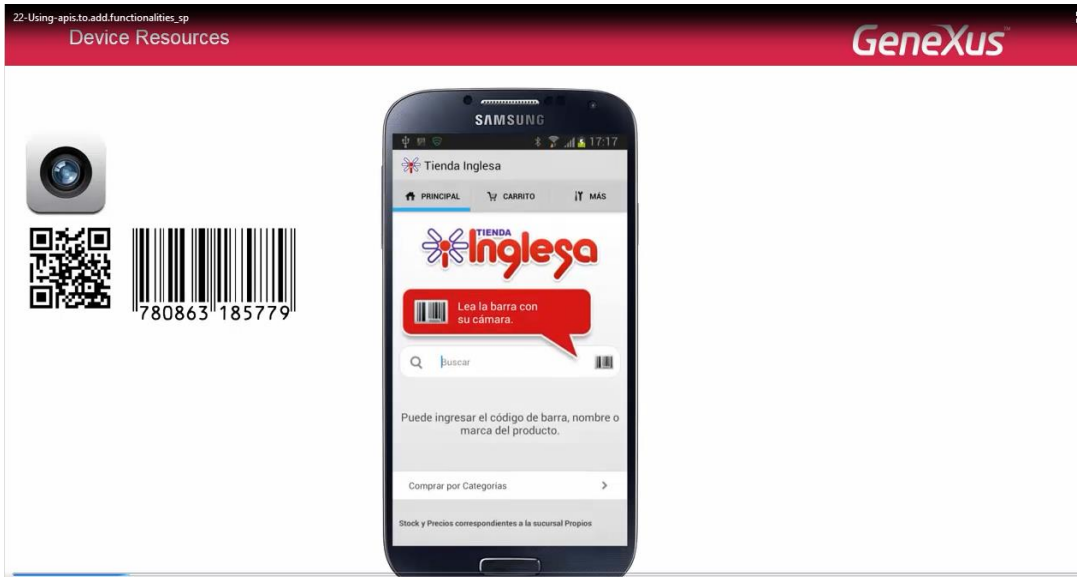


nos ofrezca el control correspondiente, tomar una foto haciendo uso de la cámara del dispositivo

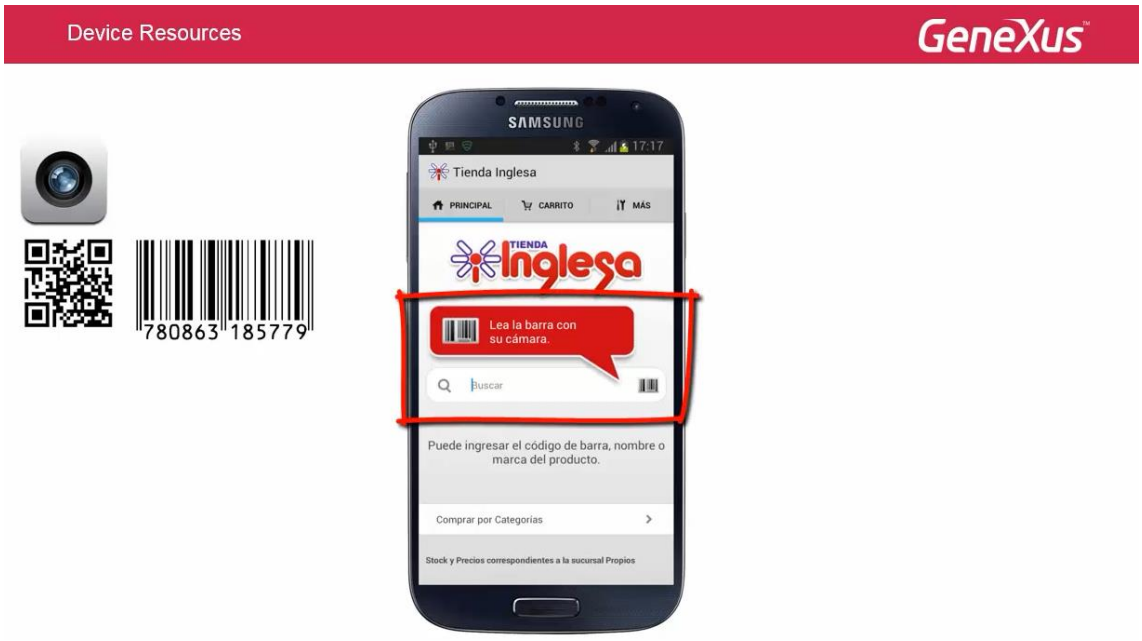


La cámara también puede ser usada para scanear QR codes o códigos de barra.

Por ejemplo para esta aplicación desarrollada para una cadena de supermercados

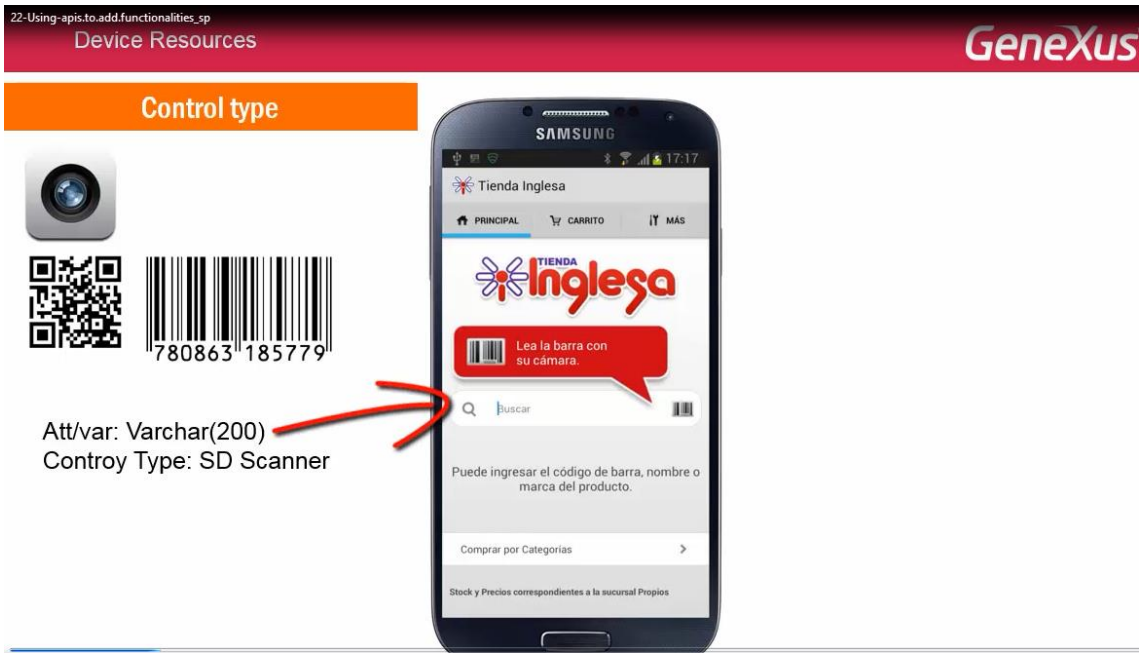


podemos escanear el código de barra del producto que tenemos en casa y que se nos acaba de terminar

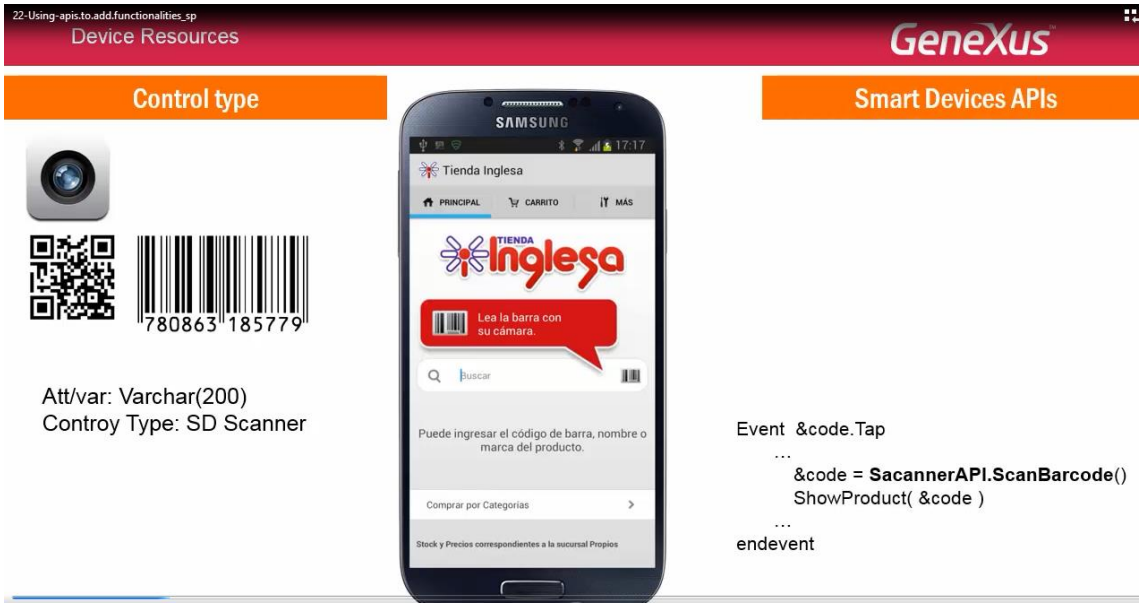


para buscarlo en la aplicación y ver su precio actual o agregarlo al carrito de compras.

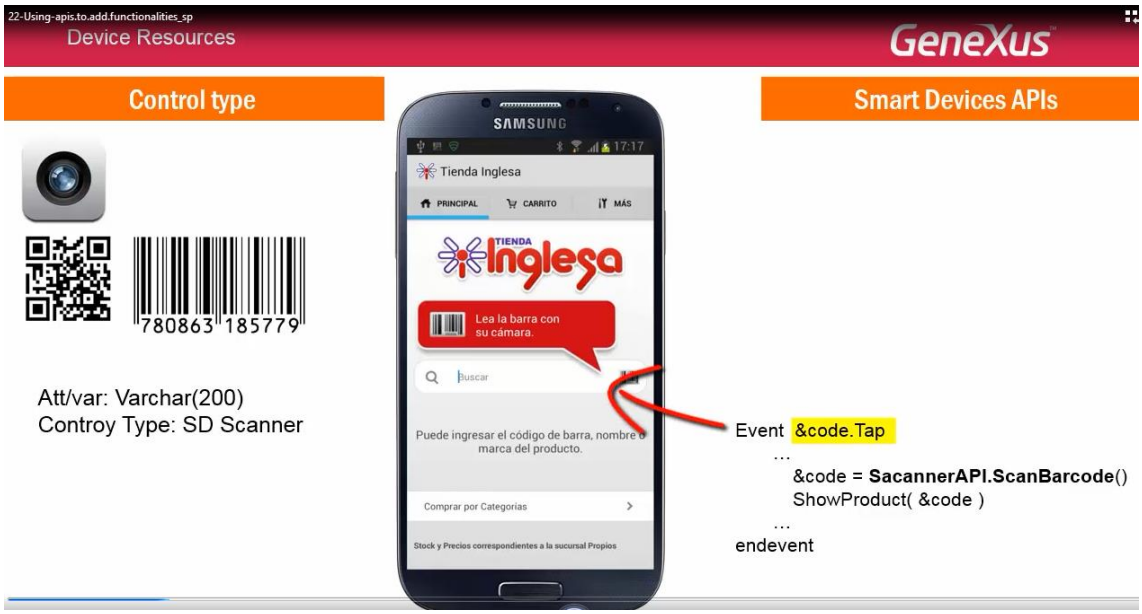
Para ello tenemos un par de opciones.



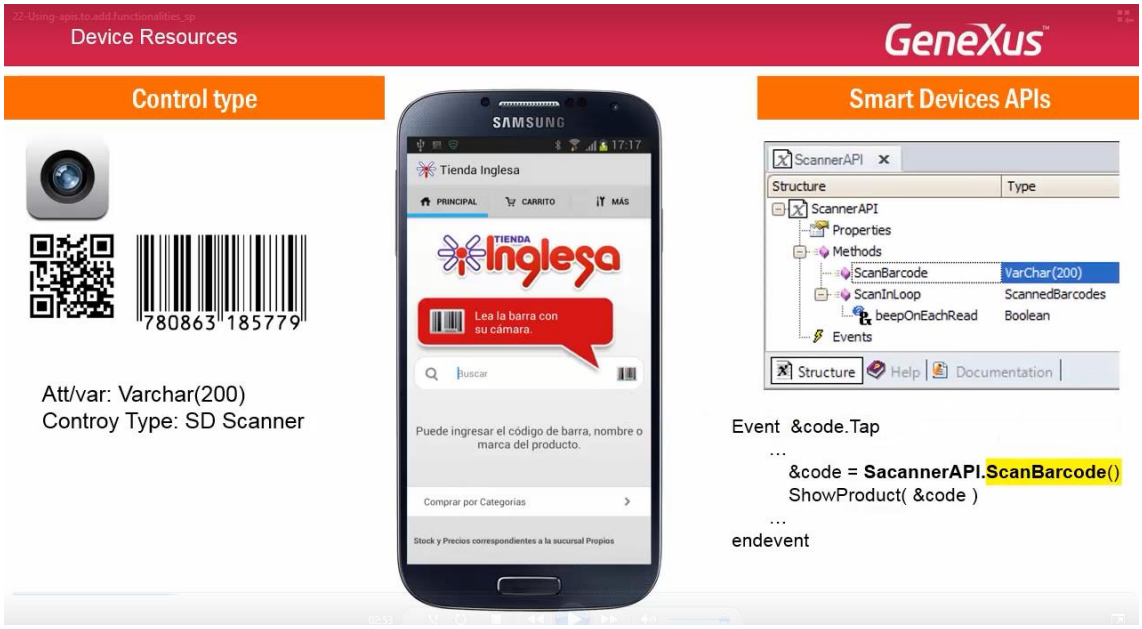
Teniendo un atributo o variable de tipo character o varchar, cambiándole su tipo de control a SD Scanner, cuando el control es de entrada, se agregará automáticamente un botón de Scan, que al ser presionado, abrirá la aplicación lectora de códigos de barras que el dispositivo tenga instalada.



La otra alternativa, será programar un evento, que podría ser el Tap sobre la variable



donde utilicemos el método ScanBarcode de una api

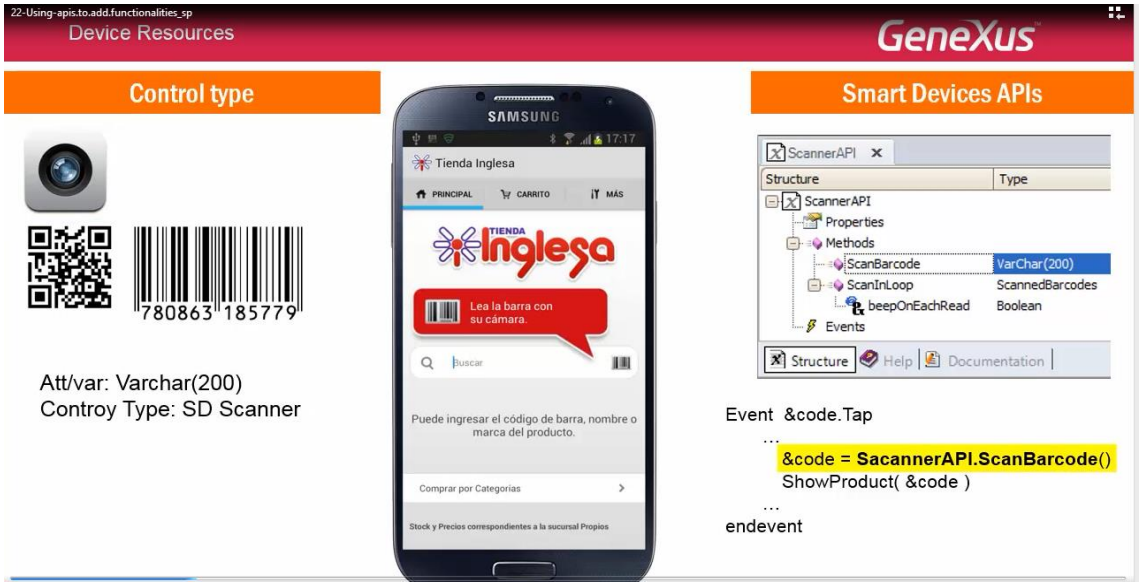


provista por GeneXus para tal fin.

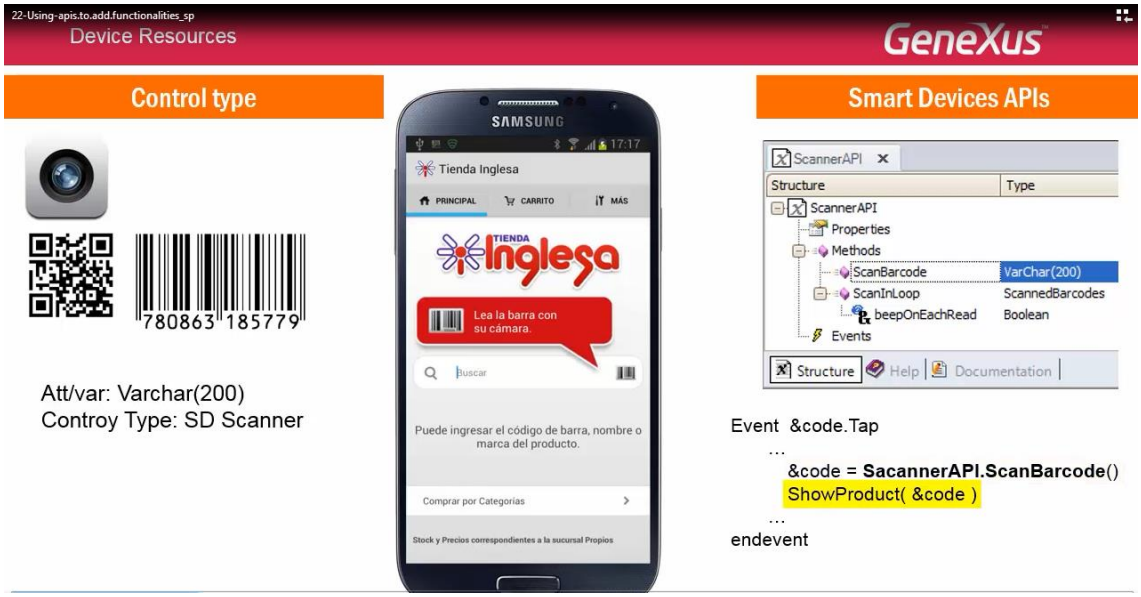
La api es como un objeto externo que provee propiedades, métodos y eventos, para abstraer su implementación y proveer las funcionalidades.

Aquí estaríamos invocando un método que ejecutará el programa lector de códigos de barras que tenga instalado el dispositivo que utilizará, por su parte, la cámara:

El valor leído, será devuelto por el método

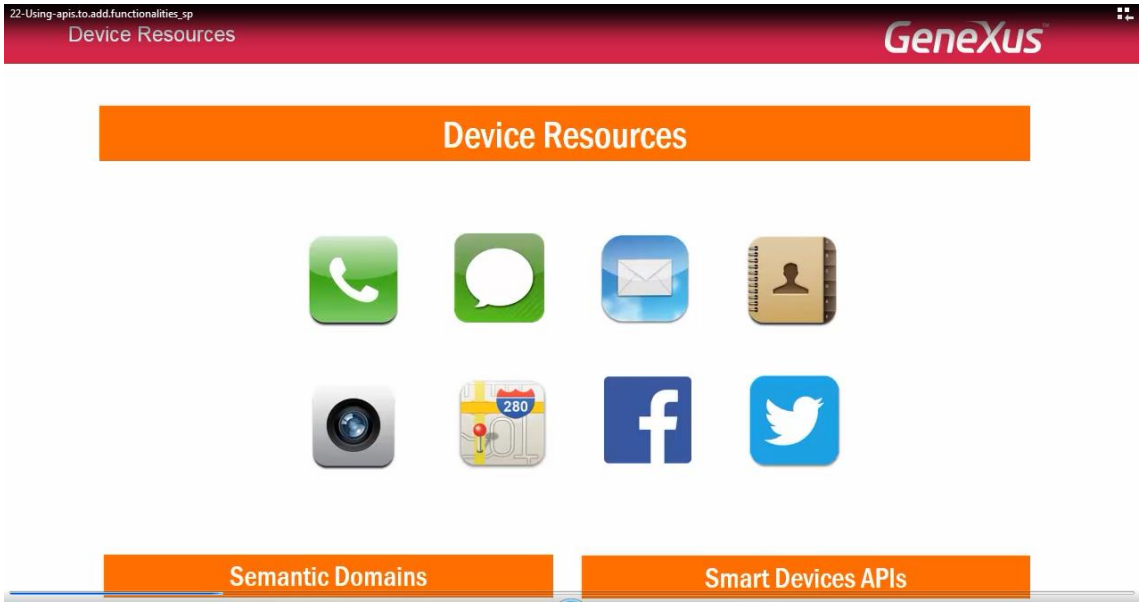


y podremos luego, invocar a un panel for Smart Devices que muestre toda la información de ese producto

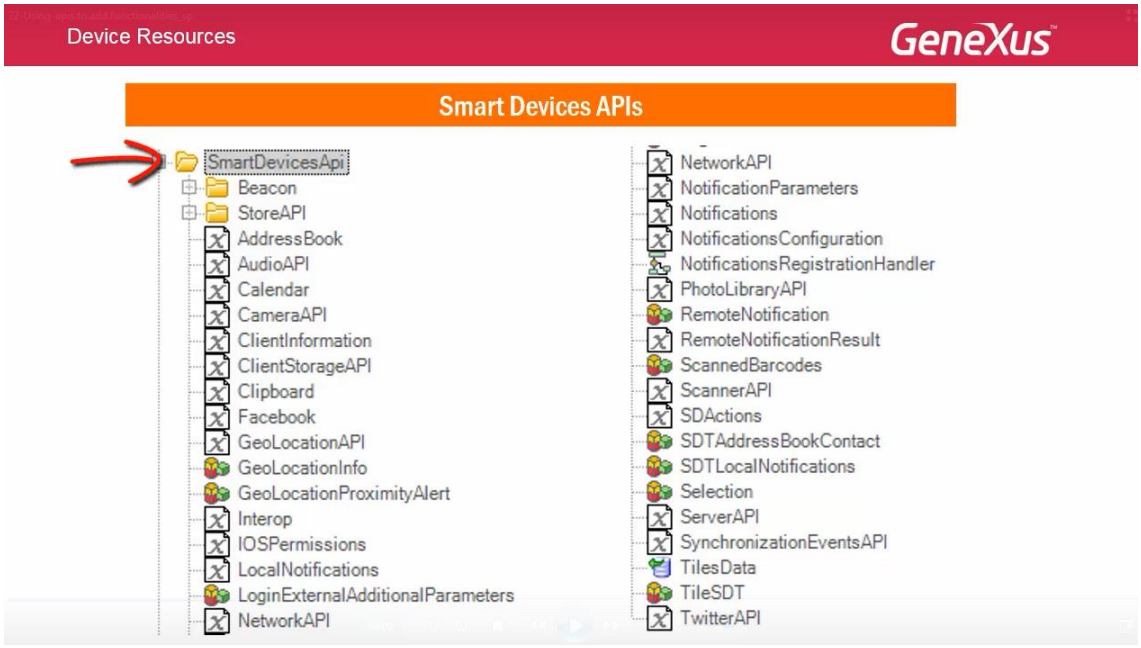


En algunos casos, las Apis necesitarán trabajar con tipos de datos estructurados, que vendrán definidos automáticamente en GeneXus, junto con ellas.

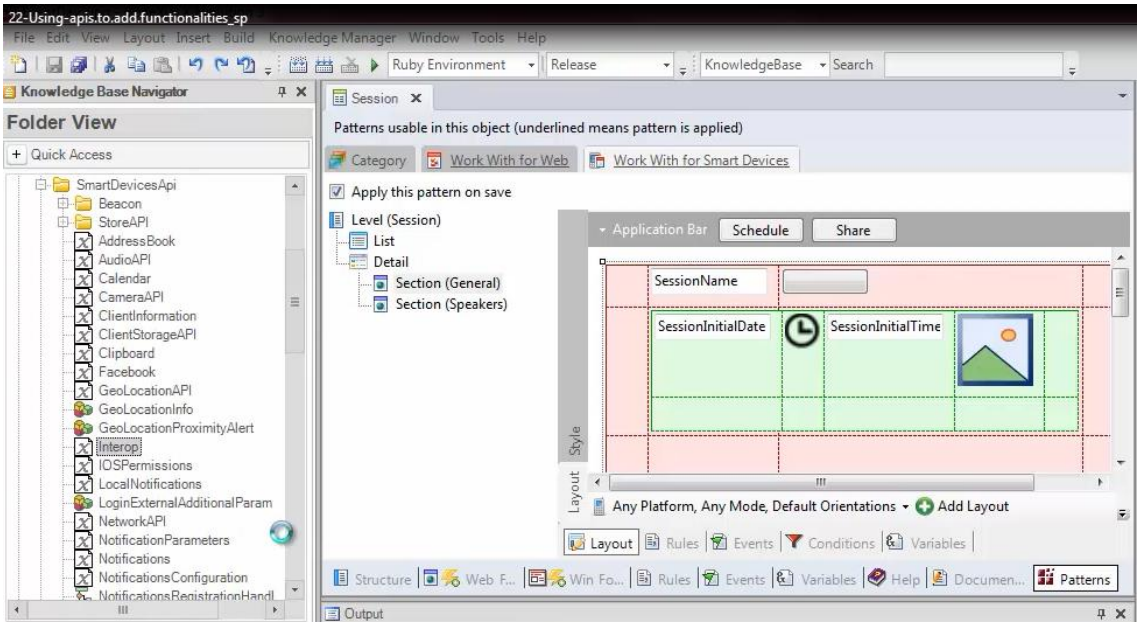
Por ejemplo, para poder leer una serie de códigos de barras, uno detrás del otro, para recién luego procesarlos, es que existe un sdt colección y un método ScanInLoop.



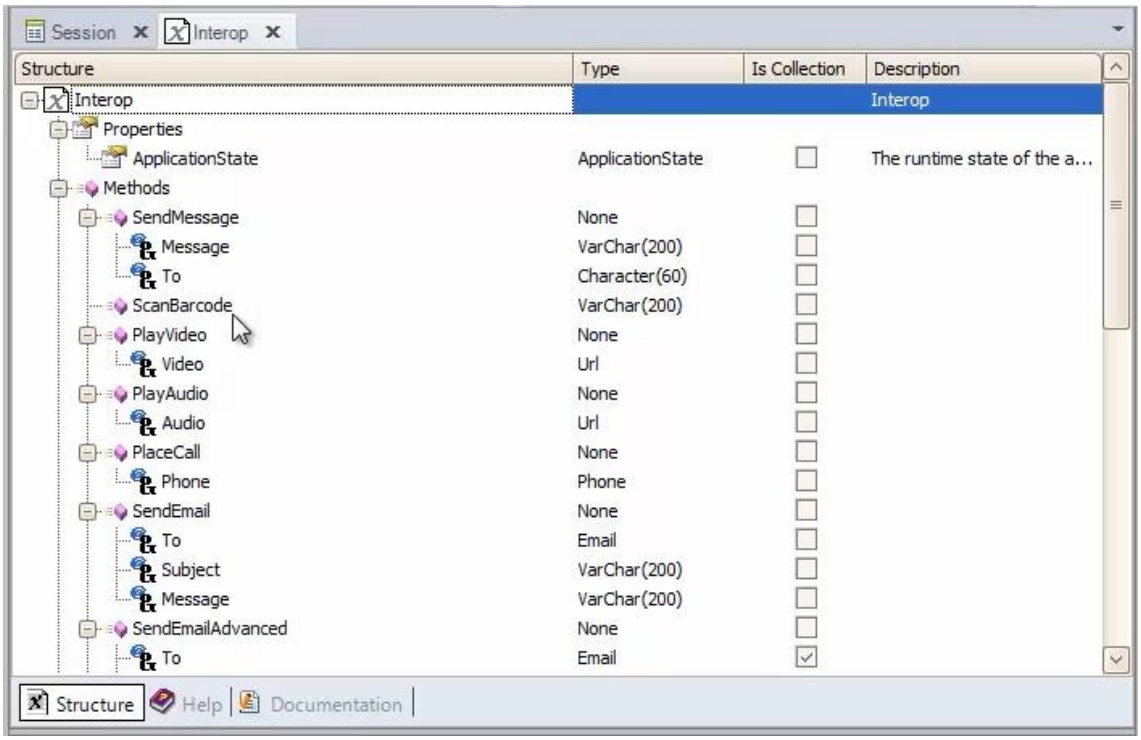
En definitiva, GeneXus nos brindará un conjunto de apis para poder agregar diferentes funcionalidades a la aplicación móvil; en particular para permitir la integración con otras aplicaciones y funcionalidades del dispositivo.



Es dentro del folder SmartDevicesApi que se encuentran todas las apis y los sdt's, procs y data providers, que estas requieran.

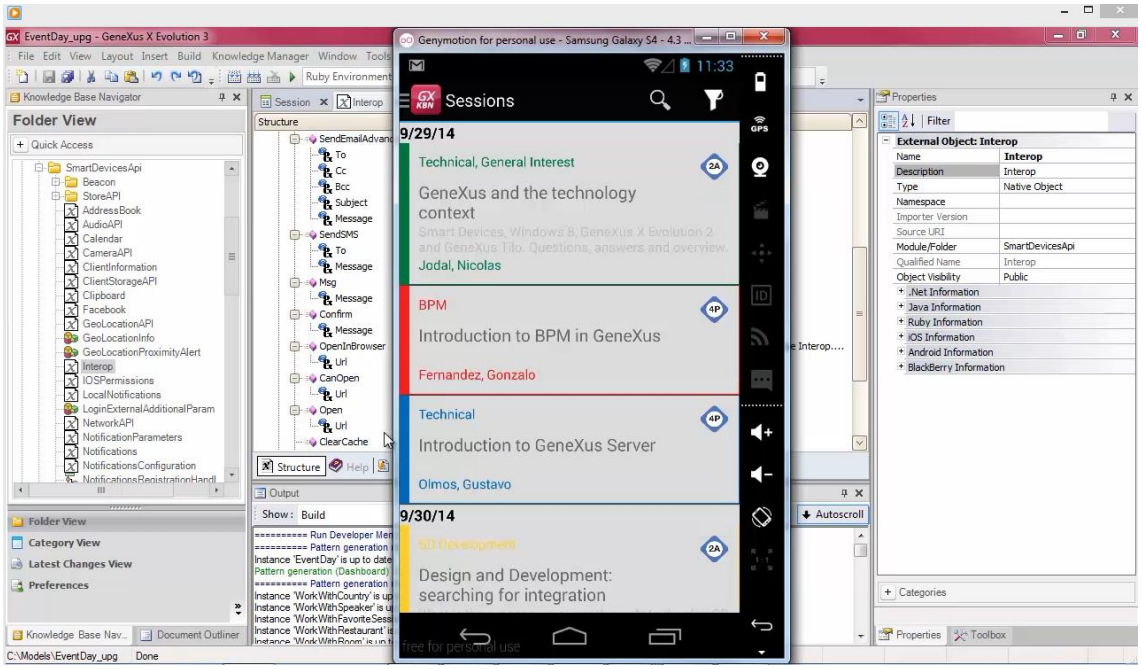


Así, algunas ofrecen la interoperabilidad con aplicaciones de mensajería que permiten

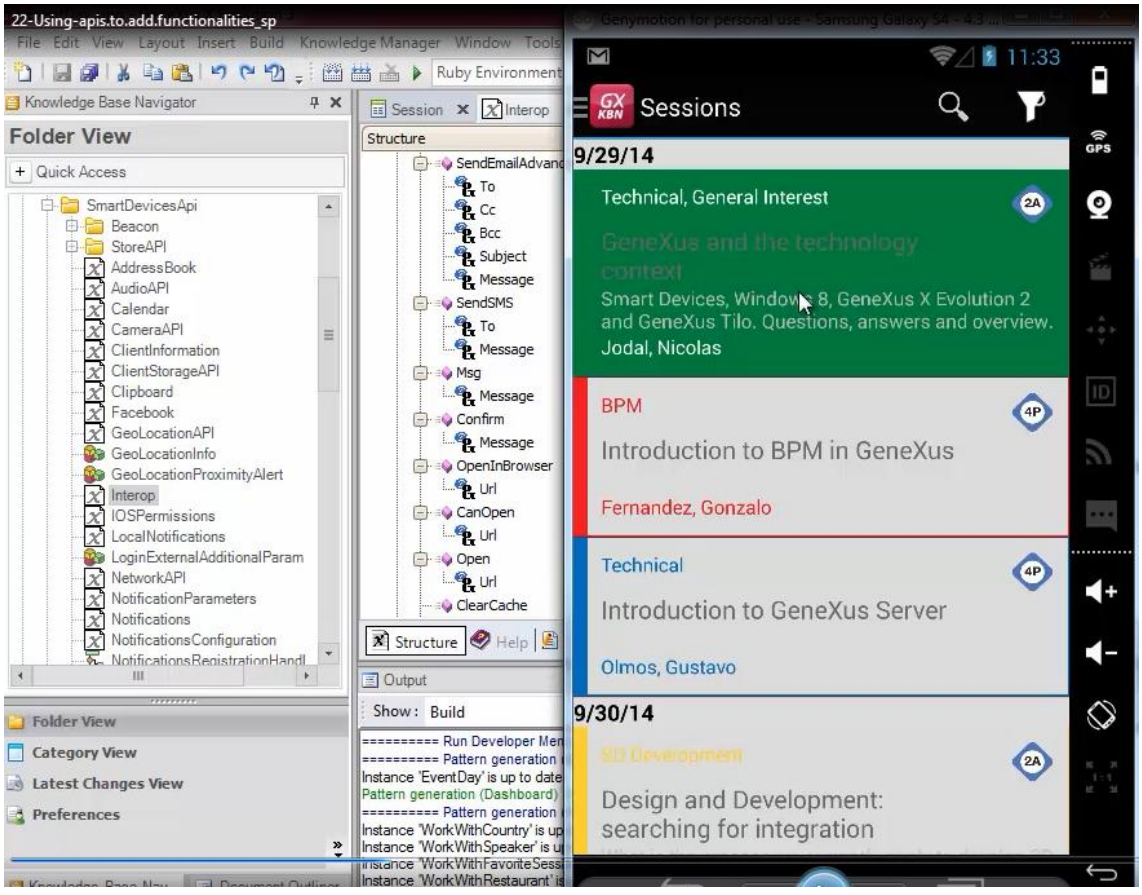


enviar mensajes, escanear códigos de barras, ejecutar videos o audios, enviar un email… un sms… desplegar mensajes al usuario o pedirle confirmación… entre otras cosas.

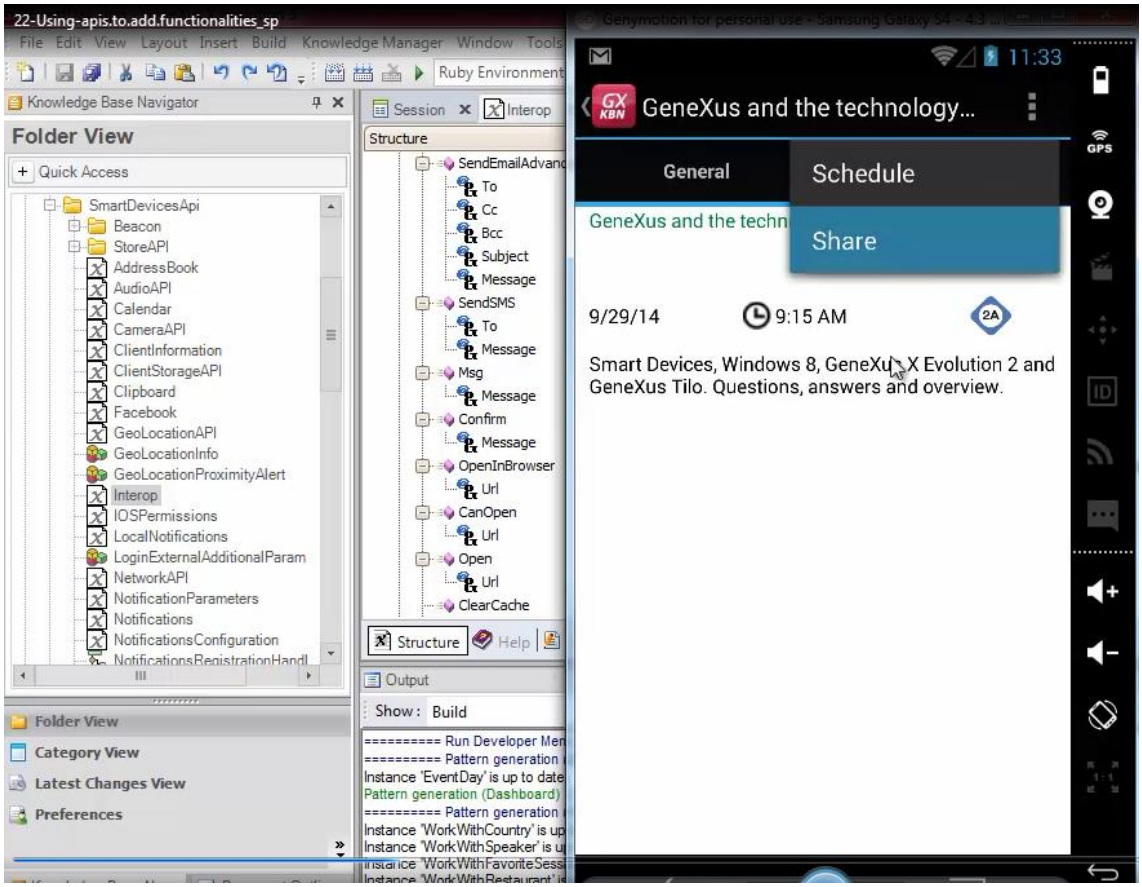
Por ejemplo



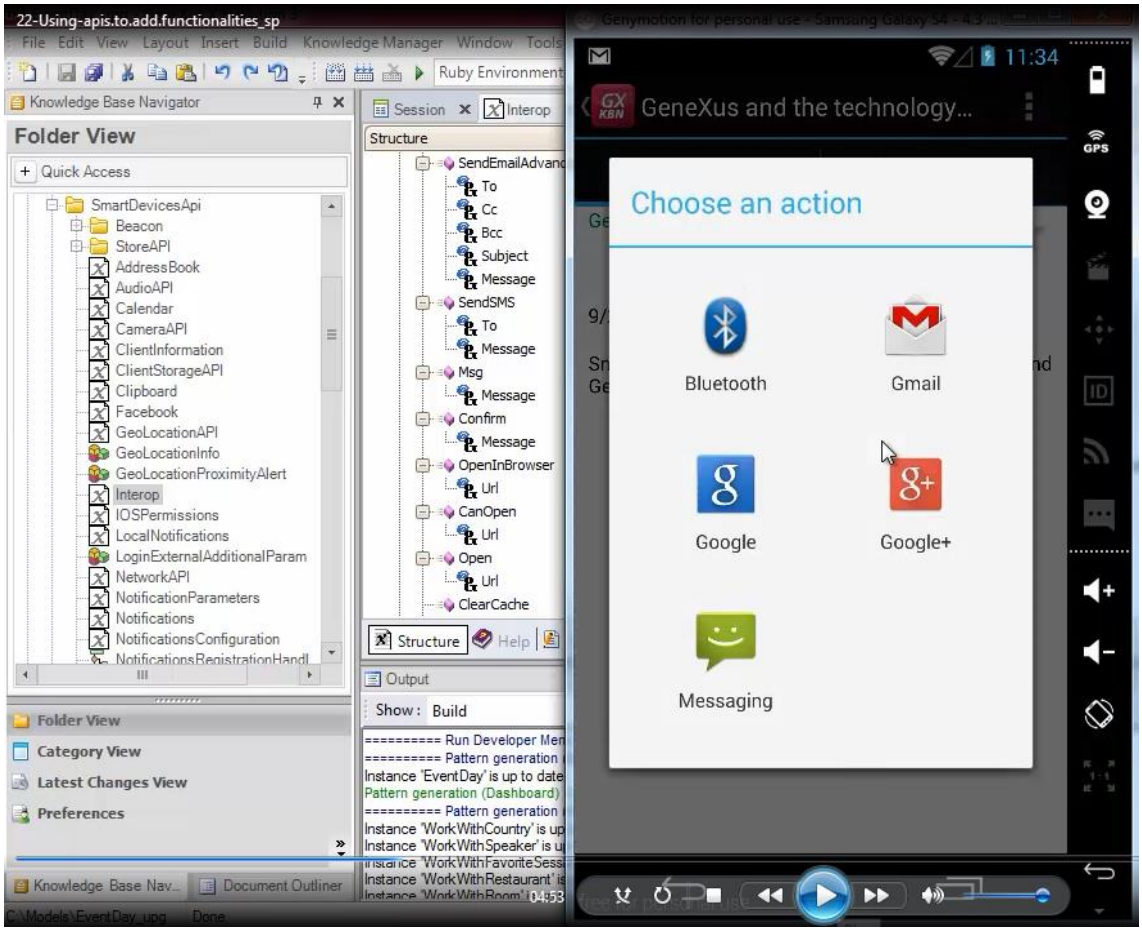
si vamos al detalle de una conferencia



vemos que podemos



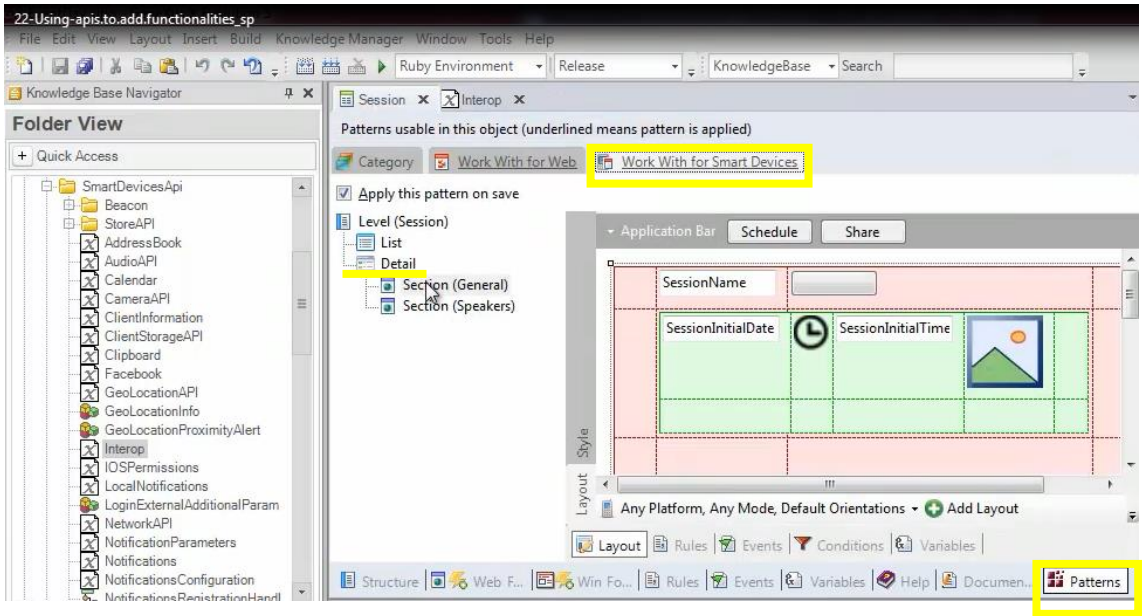
compartir parte de su información



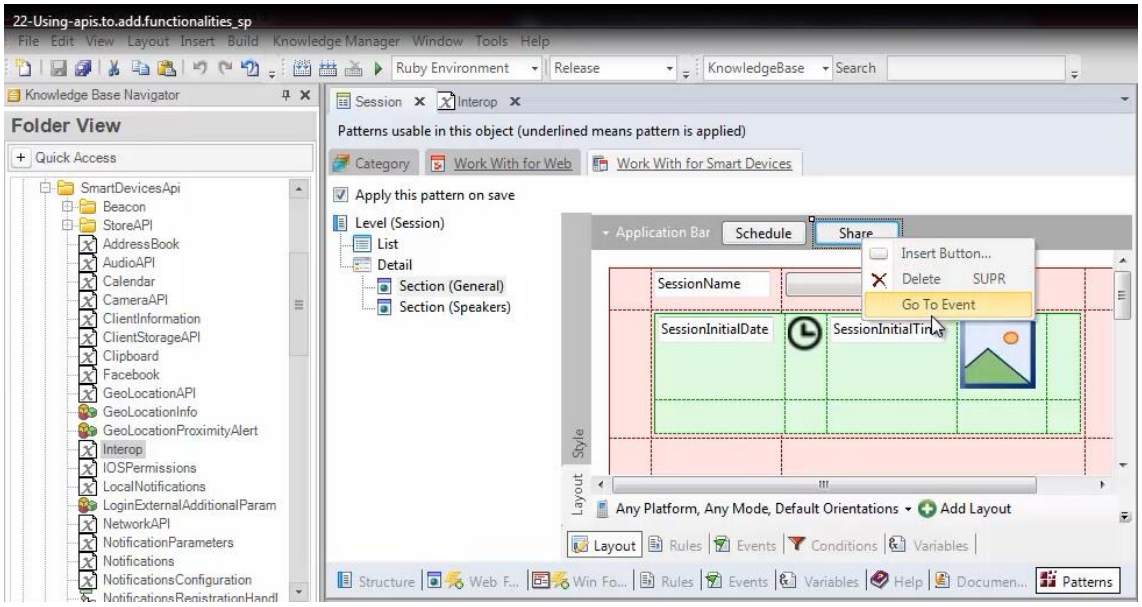
Observemos que se nos ofrece hacerlo a través de los programas que tenemos instalados en el dispositivo, que permiten enviar mensajes.

¿Cómo se implementó?

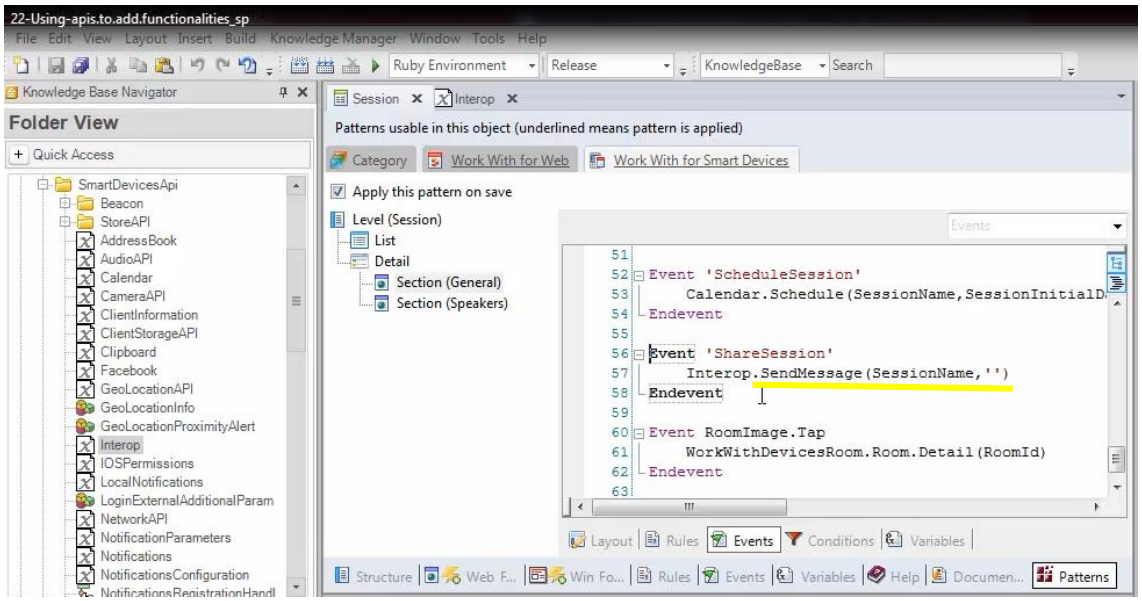
Si vamos al Detail del work with de Session



y buscamos el código del evento correspondiente al botón

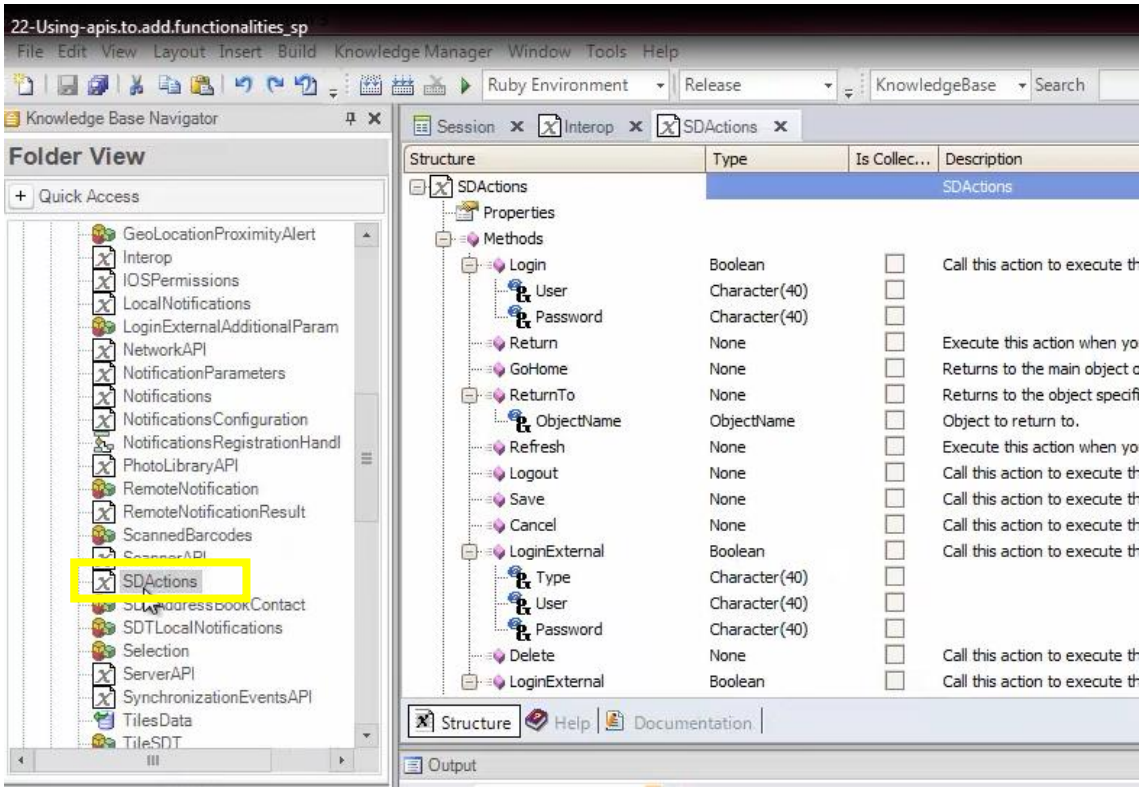


Share, en la Application Bar



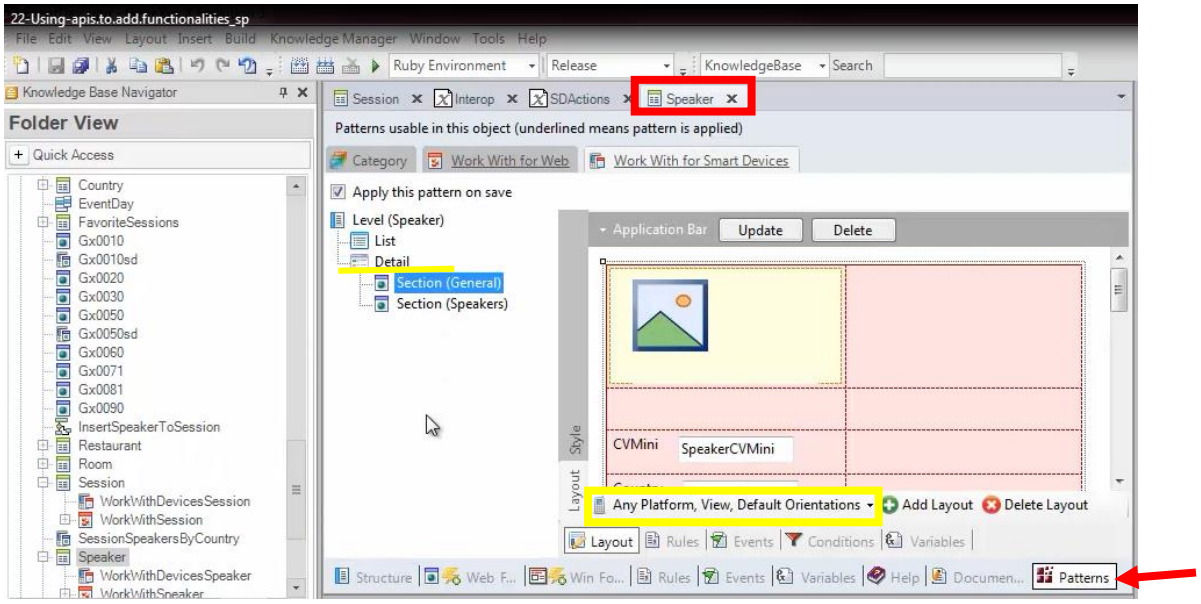
vemos que está utilizando la api Interop con su método SendMessage

Otra api que ya había aparecido anteriormente, era la SDActions

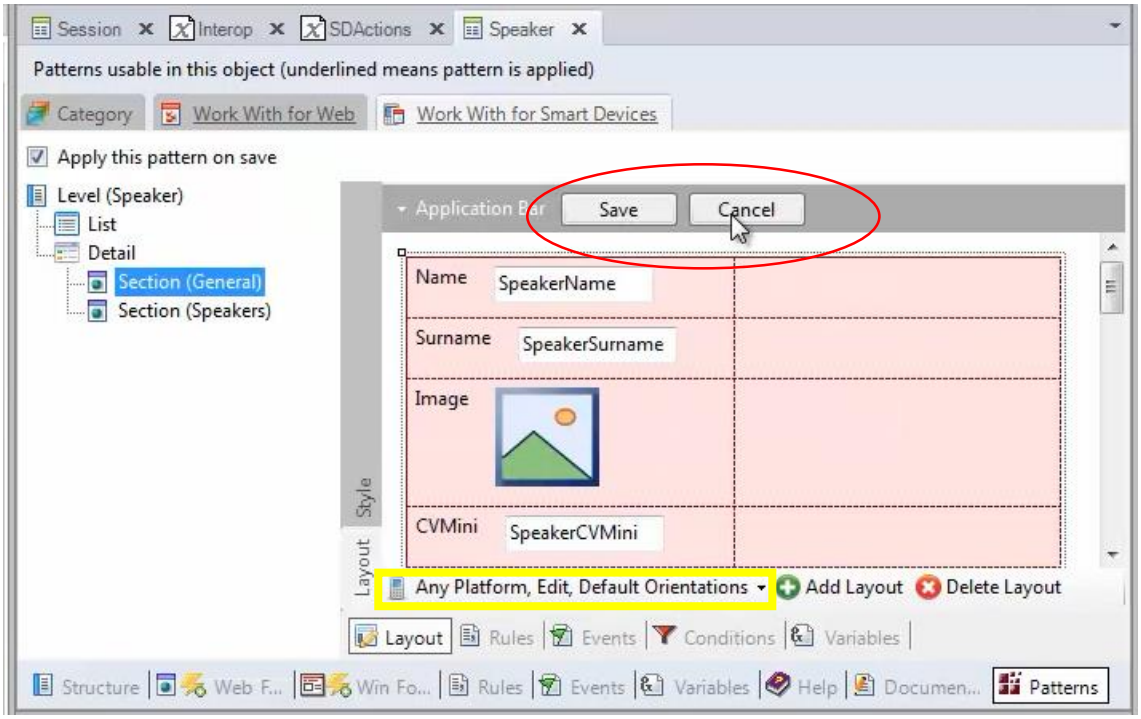


que permite entre otras cosas abstraer la funcionalidad de login, return, refresh, save, cancel… métodos estos últimos, que ya habíamos visto aparecer en los eventos que implementaba automáticamente el pattern work with a nivel del Detail.

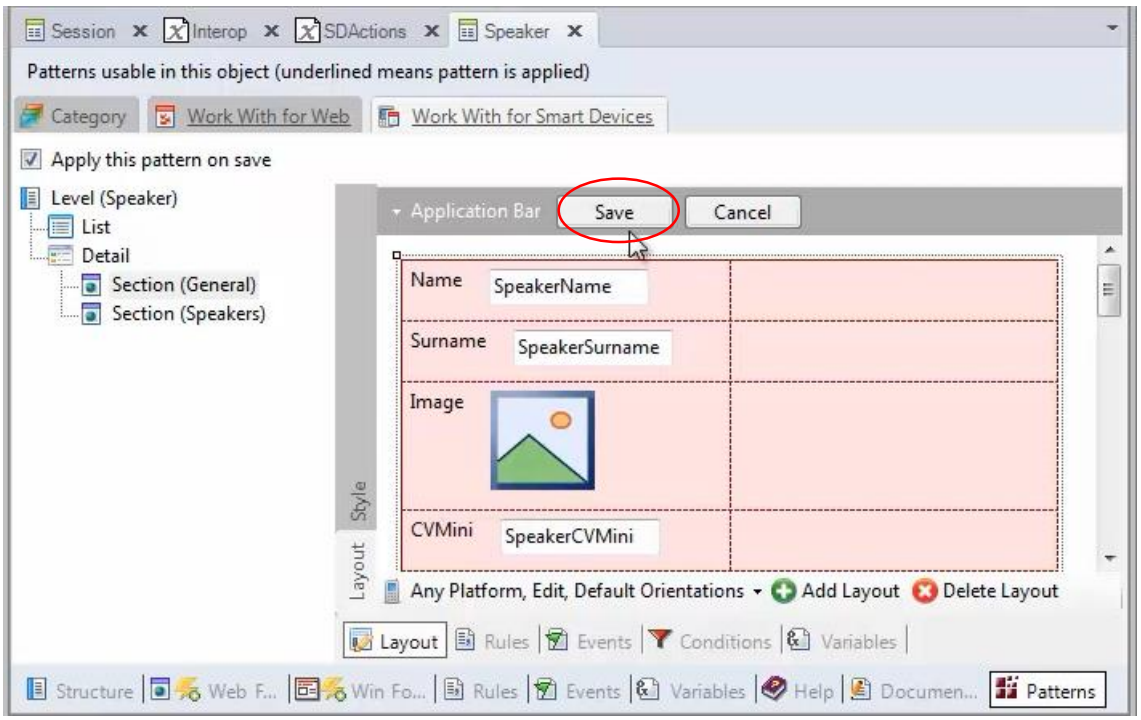
Por ejemplo, vamos a ver el Detail… la Section General de Speakers

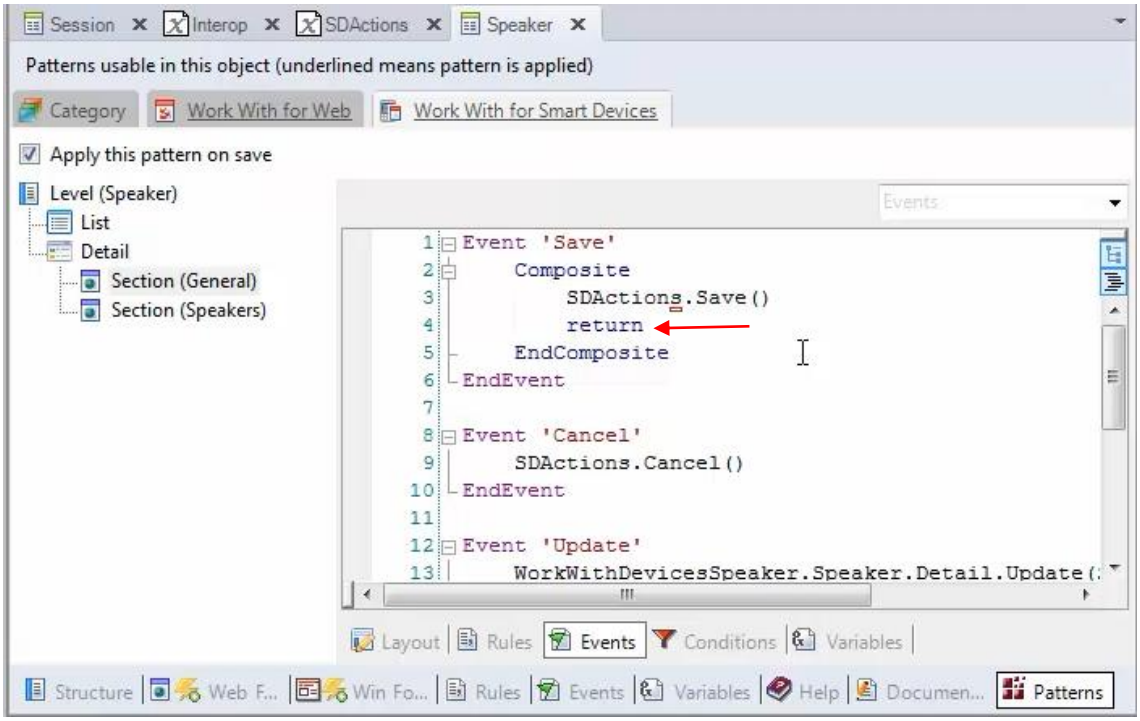


y vemos que tenemos los botones Update y Delete… para el modo View y para el Edit, los Save y Cancel



que si vamos a observar su programación

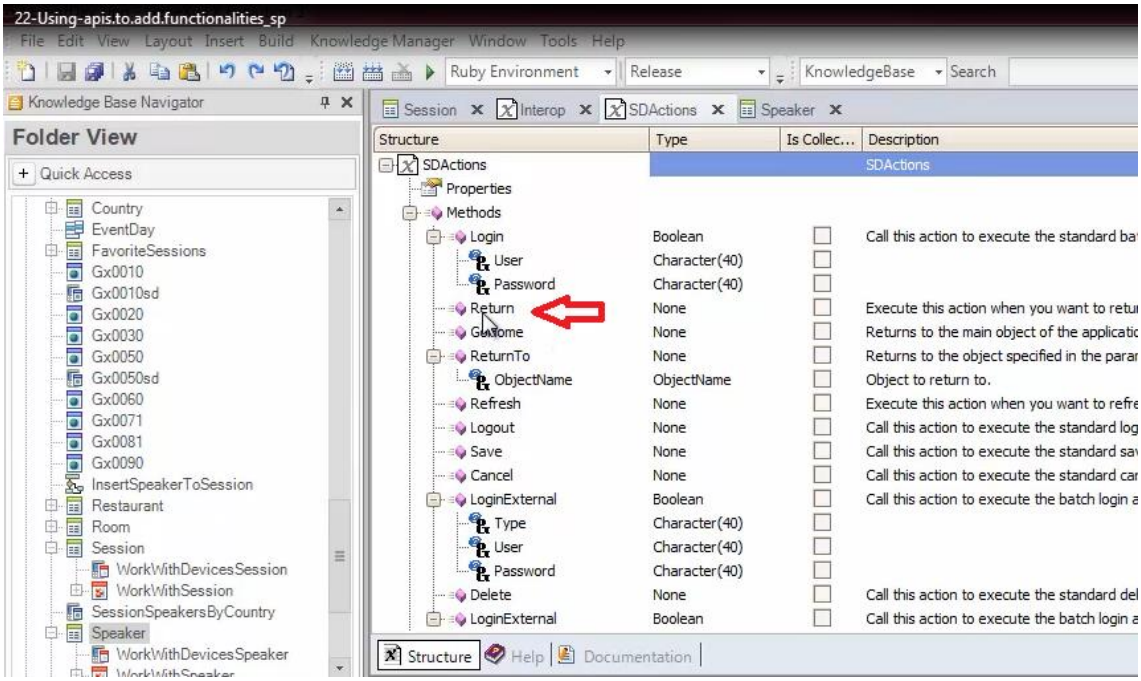




están utilizando la api que veíamos.

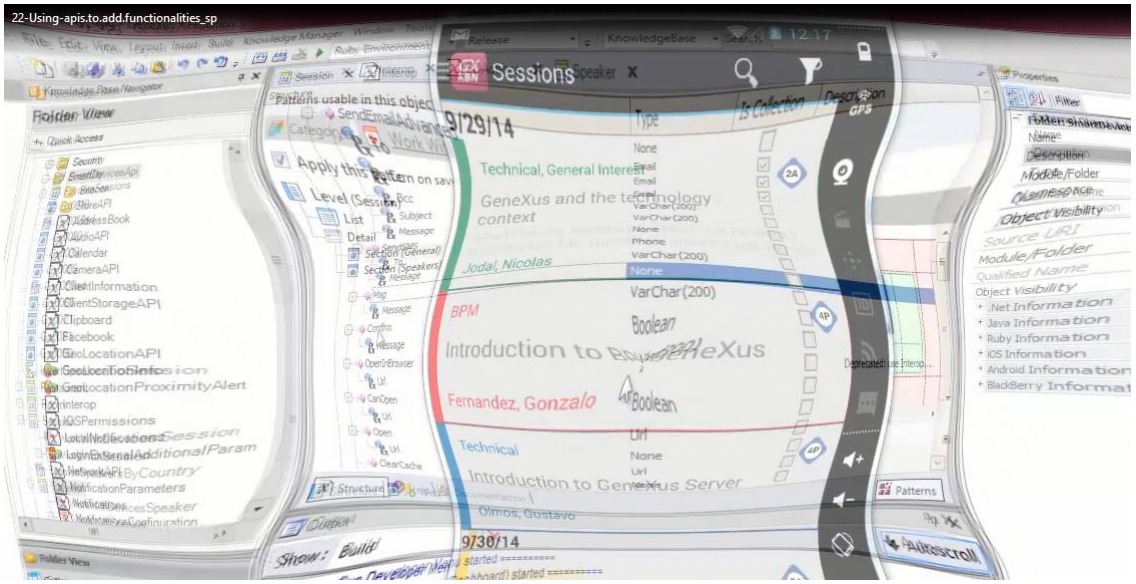
Este evento Save, corresponde al caso en el que estamos editando la información de un orador y estamos queriendo grabar esa información en la base de datos.

El método Save de esta api, lo que hace es encapsular la invocación al business component correspondiente. Luego está retornando al llamador. Este **return** corresponde al método: Return de la api

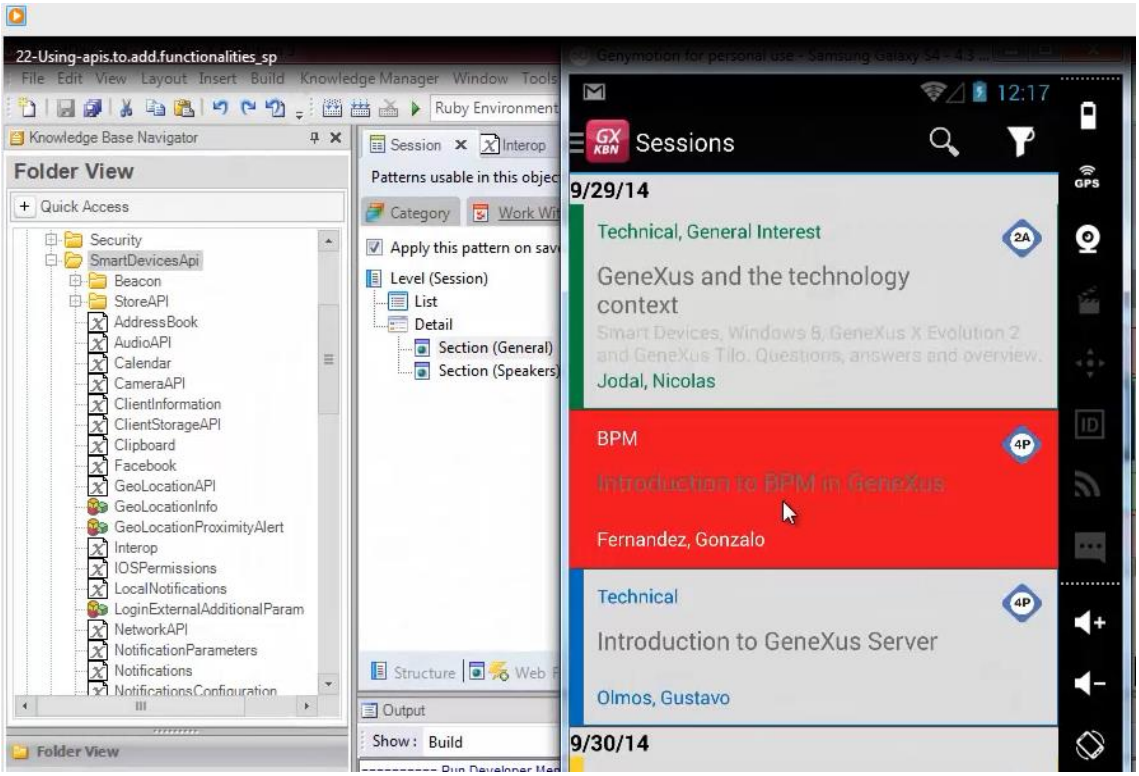


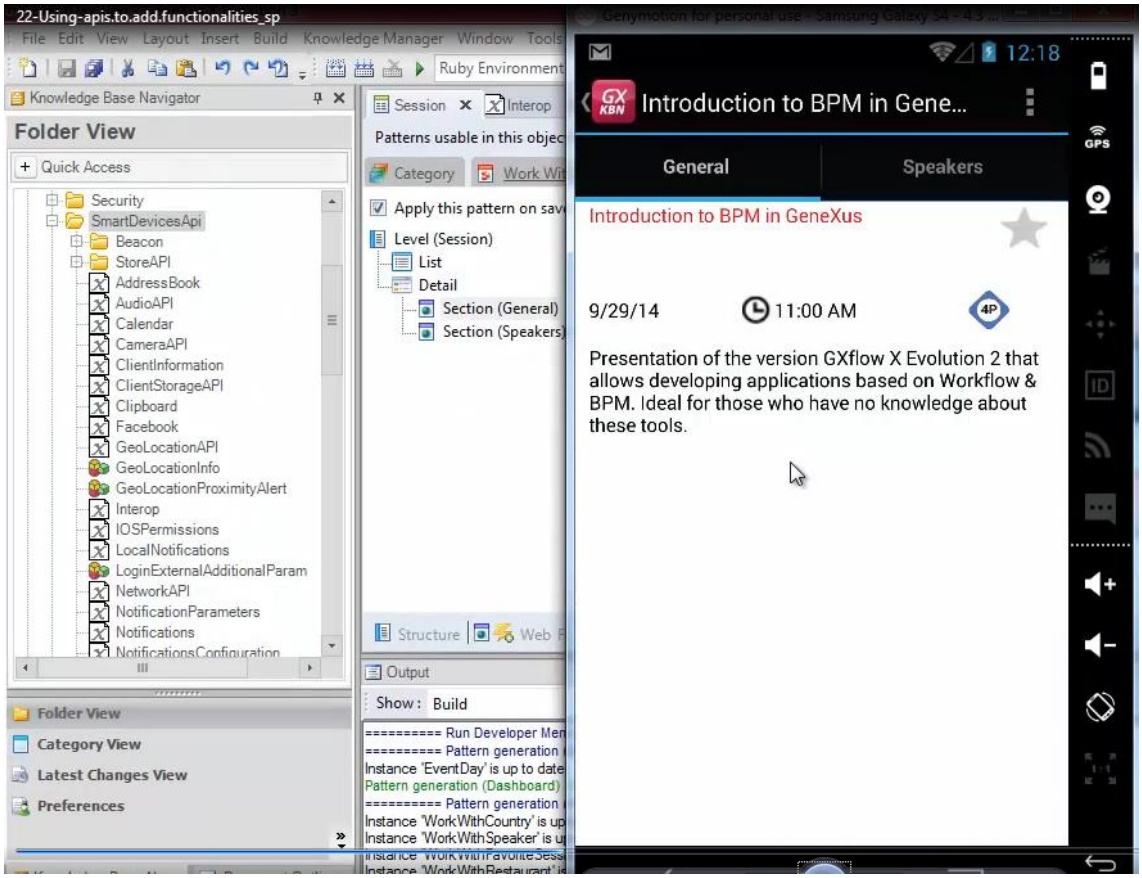
Aunque no se haya especificado como prefijo, el nombre de la api.

Algunos métodos de esta api y de la Interop, como el msg o el confirm, admiten omitir entonces el nombre de la api debido a la frecuencia con la que se utilizan.

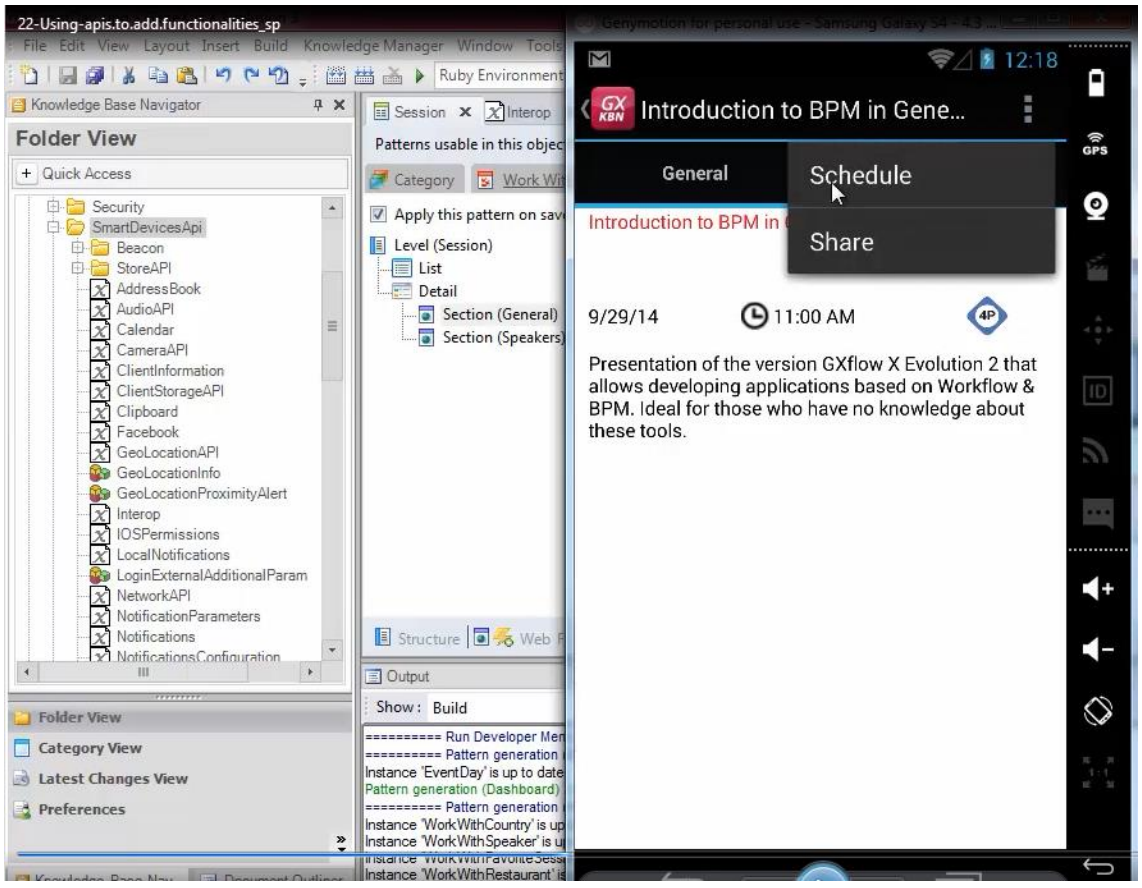


Volvamos a visualizar el detalle de una conferencia

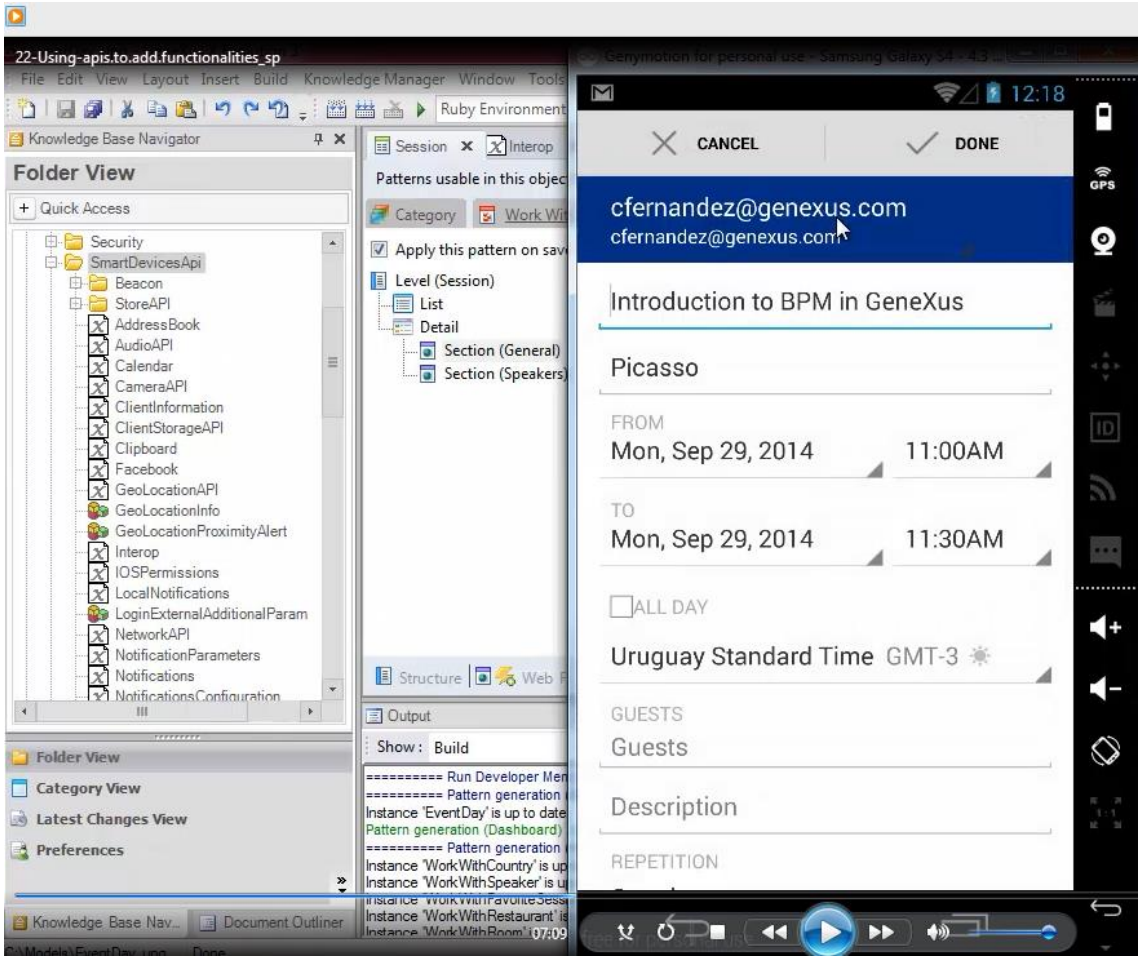




Tenemos también la posibilidad



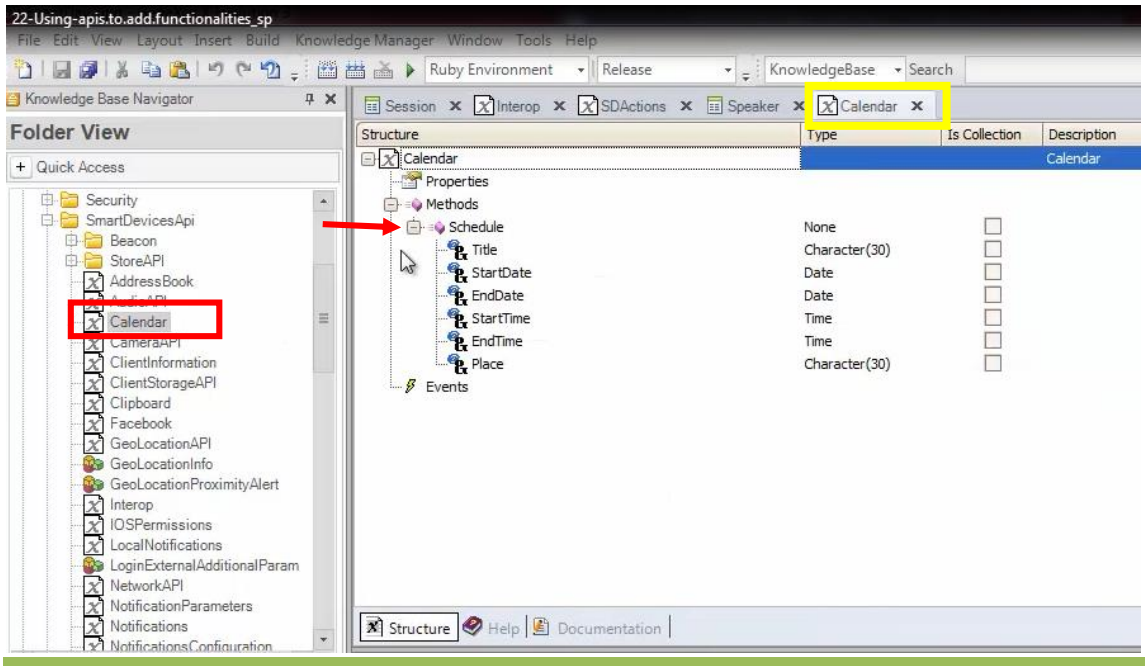
de agendarla en el caledario de nuestro dispositivo



Aquí vemos el título o nombre de la conferencia.. la sala donde se va a realizar.. y las fechas.

¿Cómo logramos esta integración?

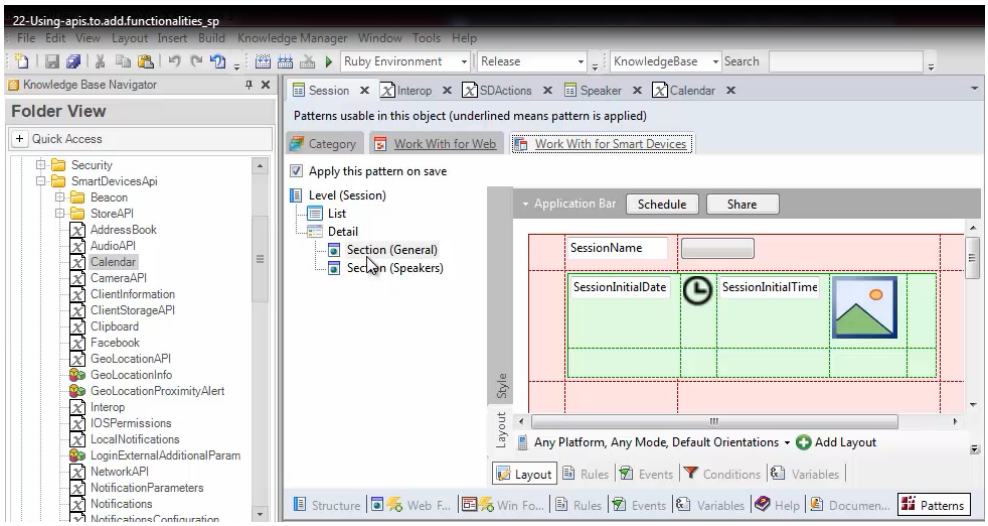
A través de la api: calendar



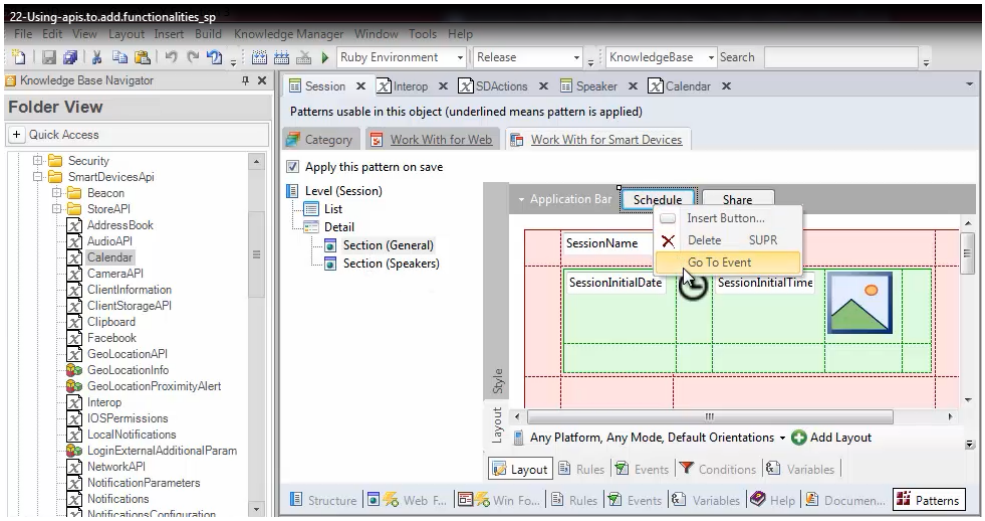
Video filmado con GeneXus X Evolution 3

que ofrece el método: Schedule … al que hay que pasarle estos parámetros que aquí aparecen

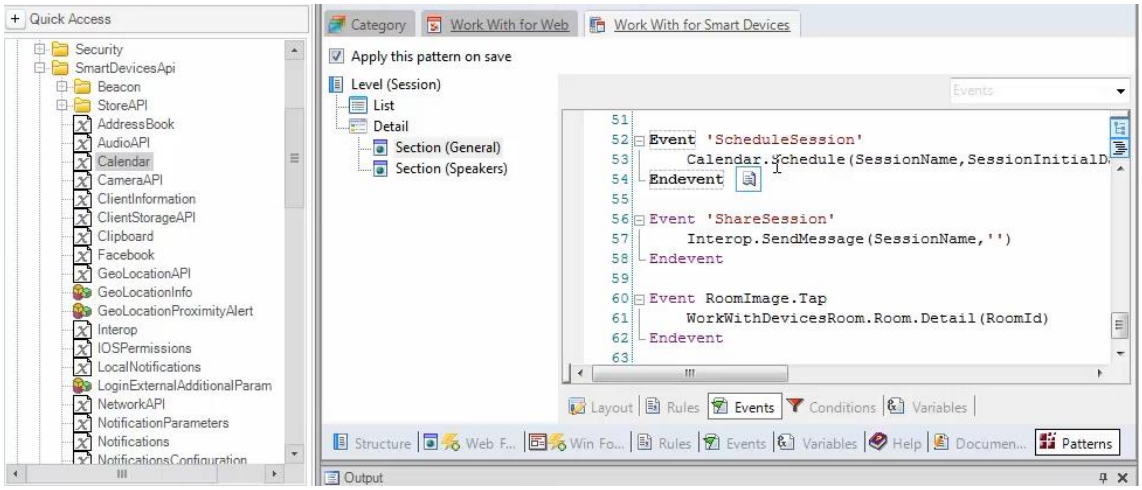
Si vamos al Detail de Session



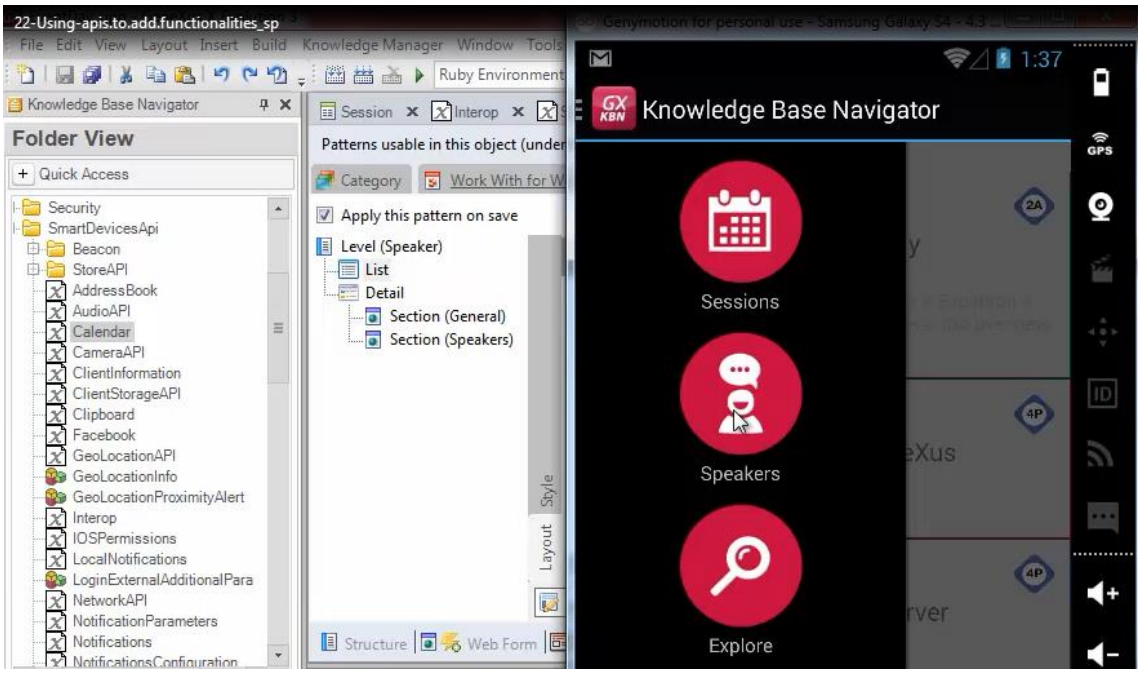
vemos el botón correspondiente



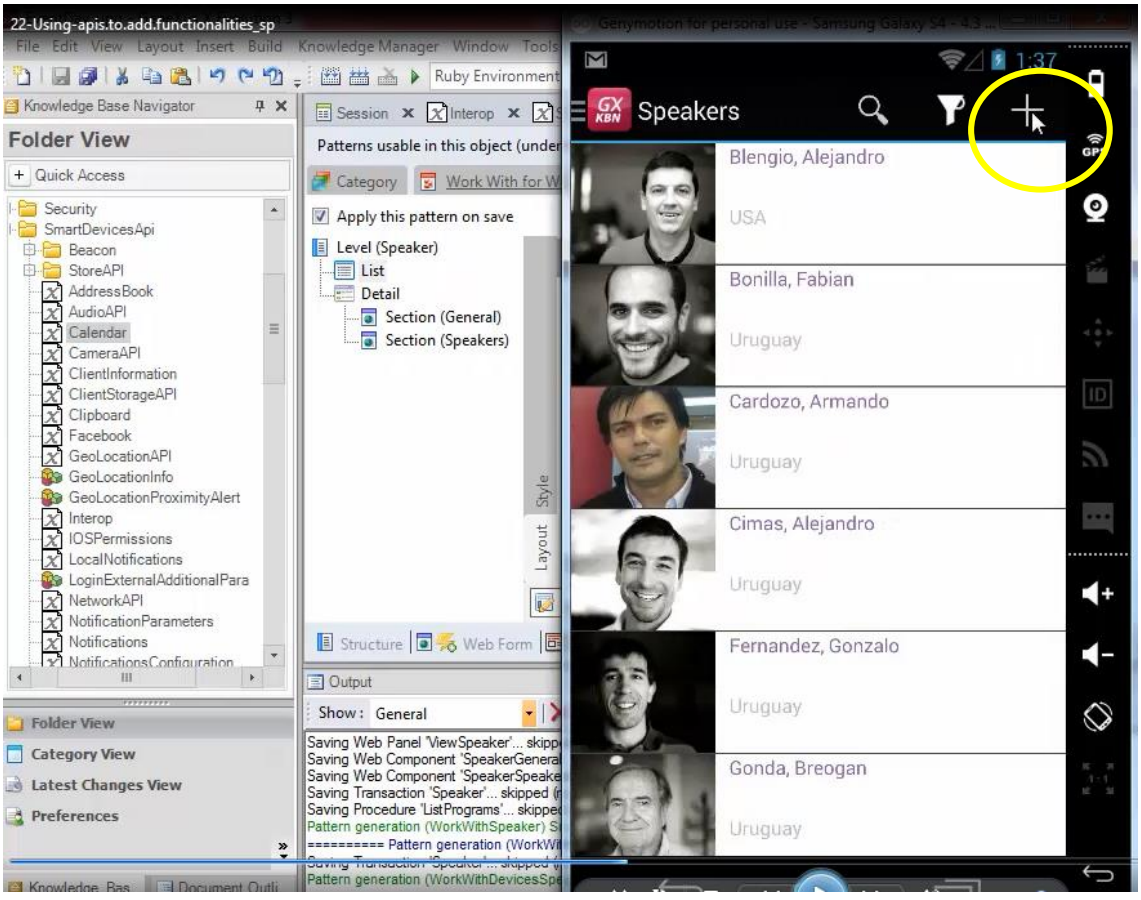
y vamos al código del evento



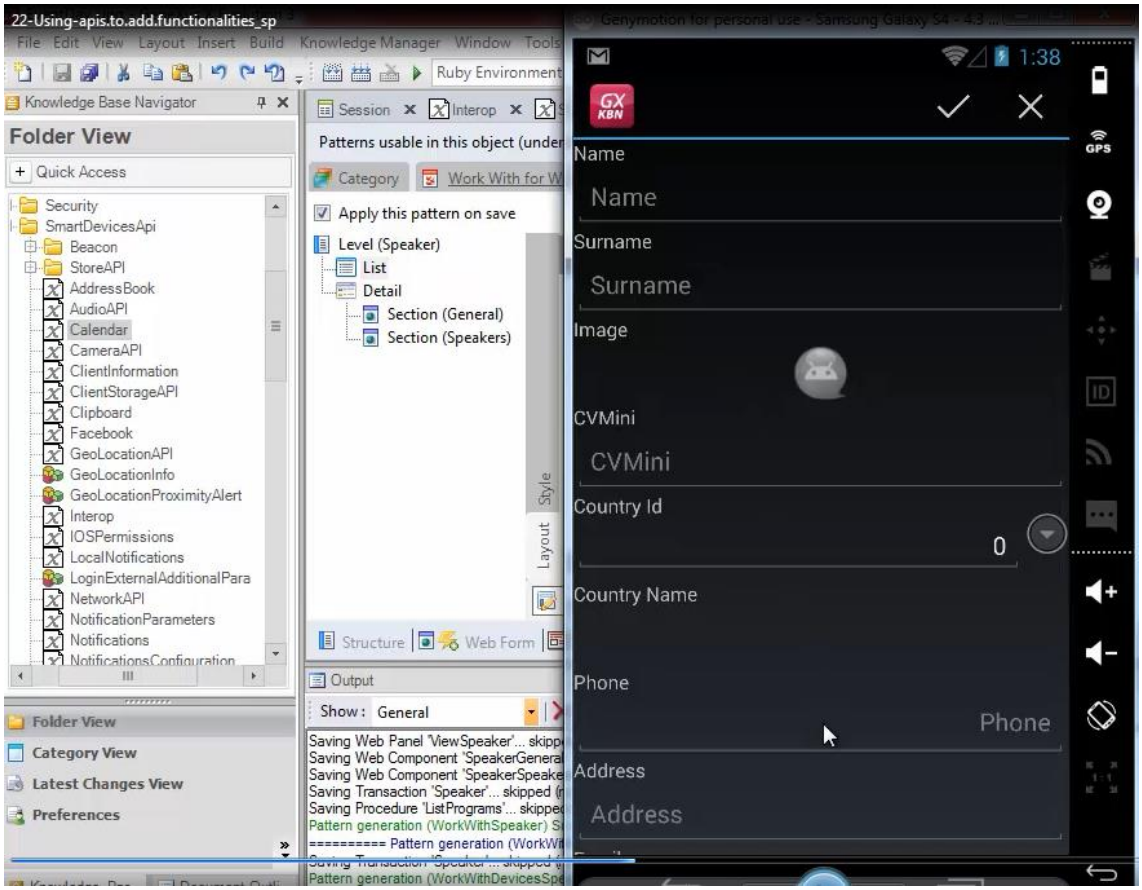
y efectivamente estamos llamando entonces a la api: Calendar, con el método Schedule (pasándole la información: SessionName, SessionInitialDate, FinalDate, InitialTime, EndTime y el nombre de la sala).



Supongamos ahora que desde el List de Speakers, queremos

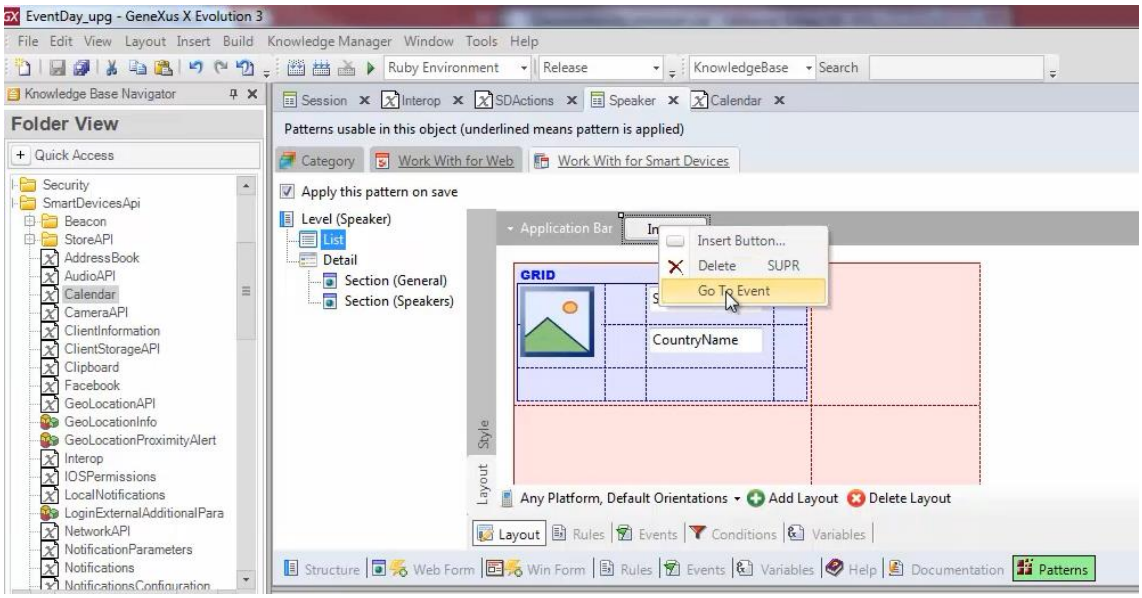


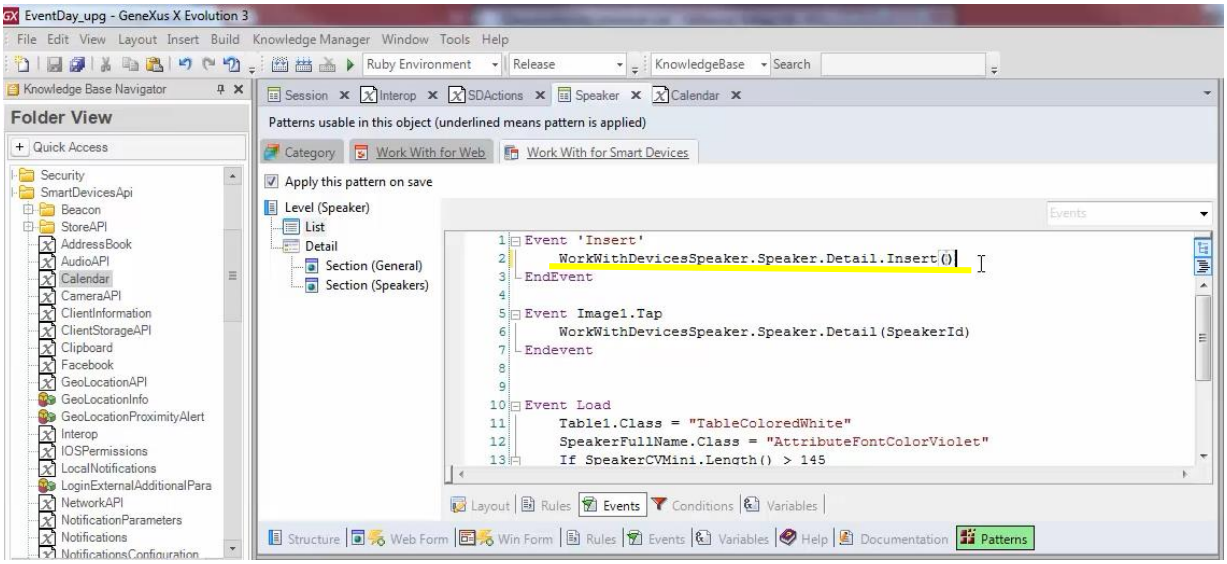
que tras ingresar un nuevo orador en la base de datos



se lo agregue también como contacto en la libreta de direcciones del dispositivo.

Para ello

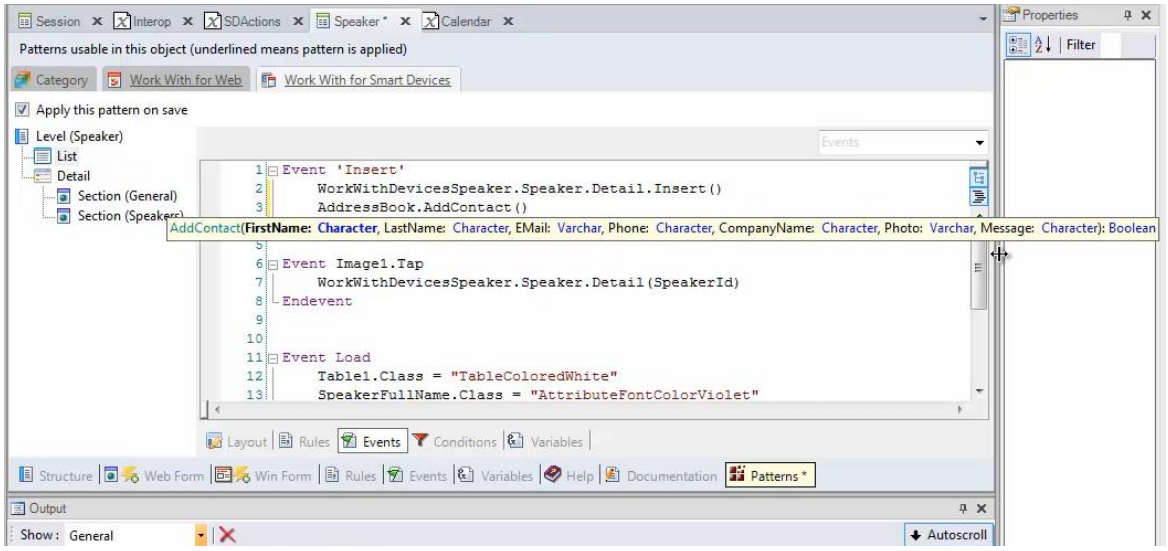




después de invocar al Detail en modo Insert, debemos invocar a la api AddressBook

Método: AddContact

Al que debemos de pasarle esta serie de parámetros

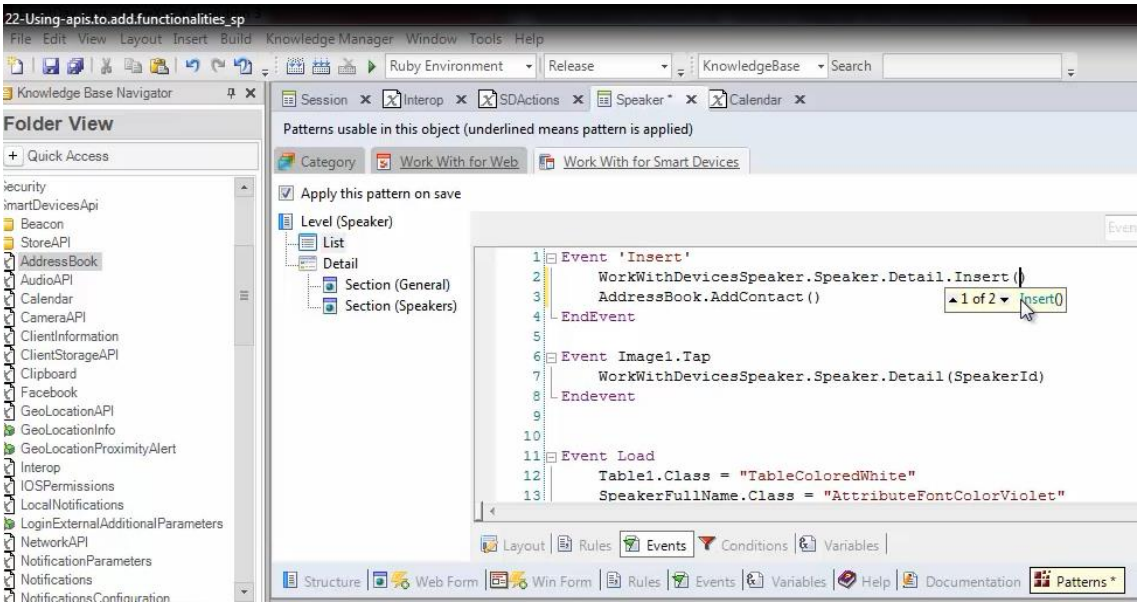


y en este orden

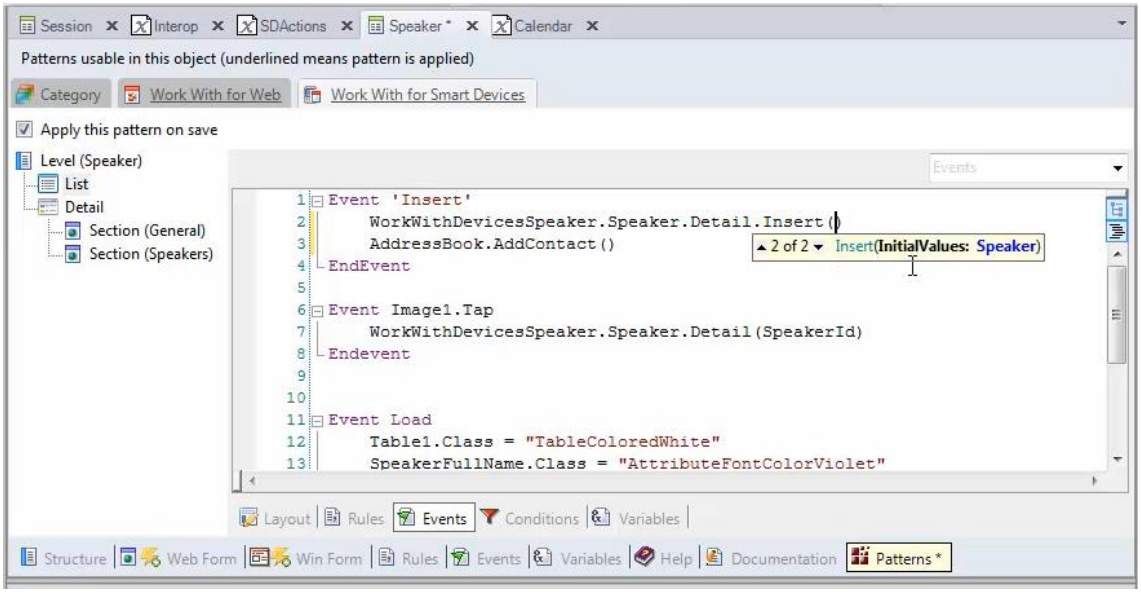
Estos parámetros van a corresponder a datos del orador insertado por el usuario, en este panel.

¿Cómo recuperamos esos datos para poder enviarlos como parámetro a este método?

Recordemos que el insert ofrecía 2 opciones. La primera era no pasar ningún parámetro.



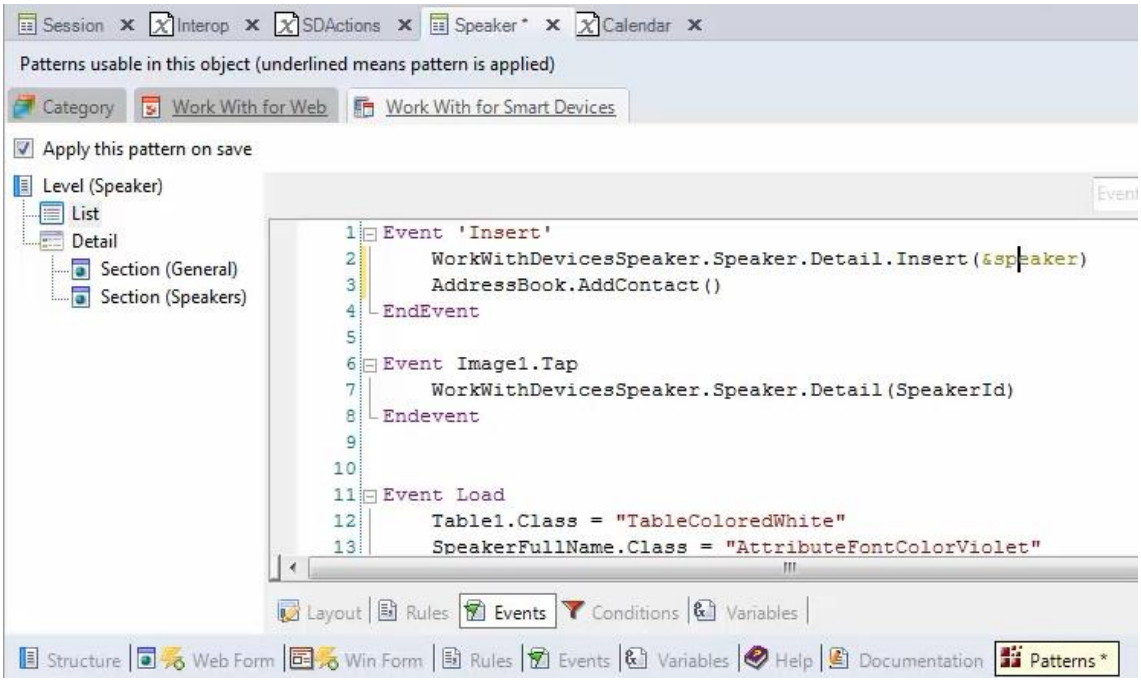
Y la segunda

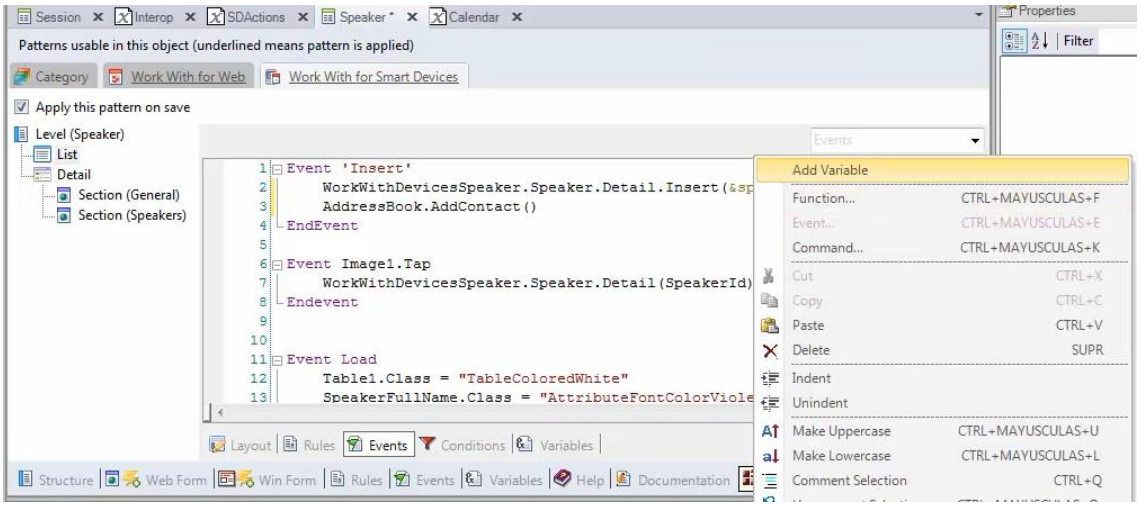


era pasar un único parámetro del tipo de datos el business component asociado al nivel en el que se está queriendo hacer la inserción.

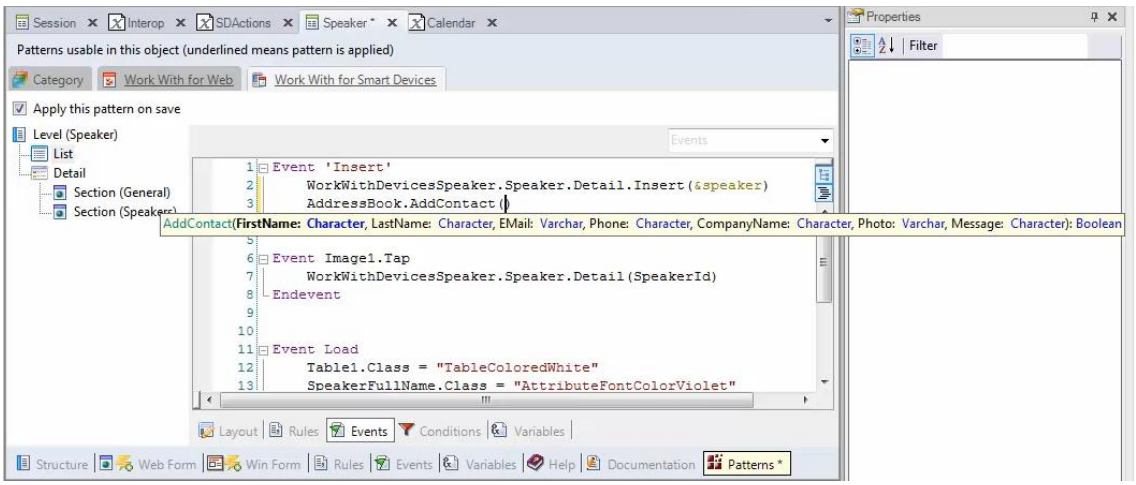
Este business component nos permitía inicializar valores en la pantalla a la que estamos invocando, y a la vez, al finalizar la inserción, va a venir cargado con todos los valores insertados; y esa es la manera que vamos a tener para poder recuperar esos datos para enviárselos al método AddContact.

Por tanto vamos a definir una variable &speaker





que por defecto, como se llama igual que el business component, va a quedar inicializada con ese tipo de datos y esa variable es la que vamos a utilizar, para en el método AddContact



ir agregando los parámetros:

El primero -> FirstName, va a corresponder a → &Speaker – punto - Name

El segundo, LastName, va a corresponder a → &Speaker – punto - SpeakerSurname

El tercero nos está advirtiendo que es el email → &Speaker – punto – SpeakerEmail

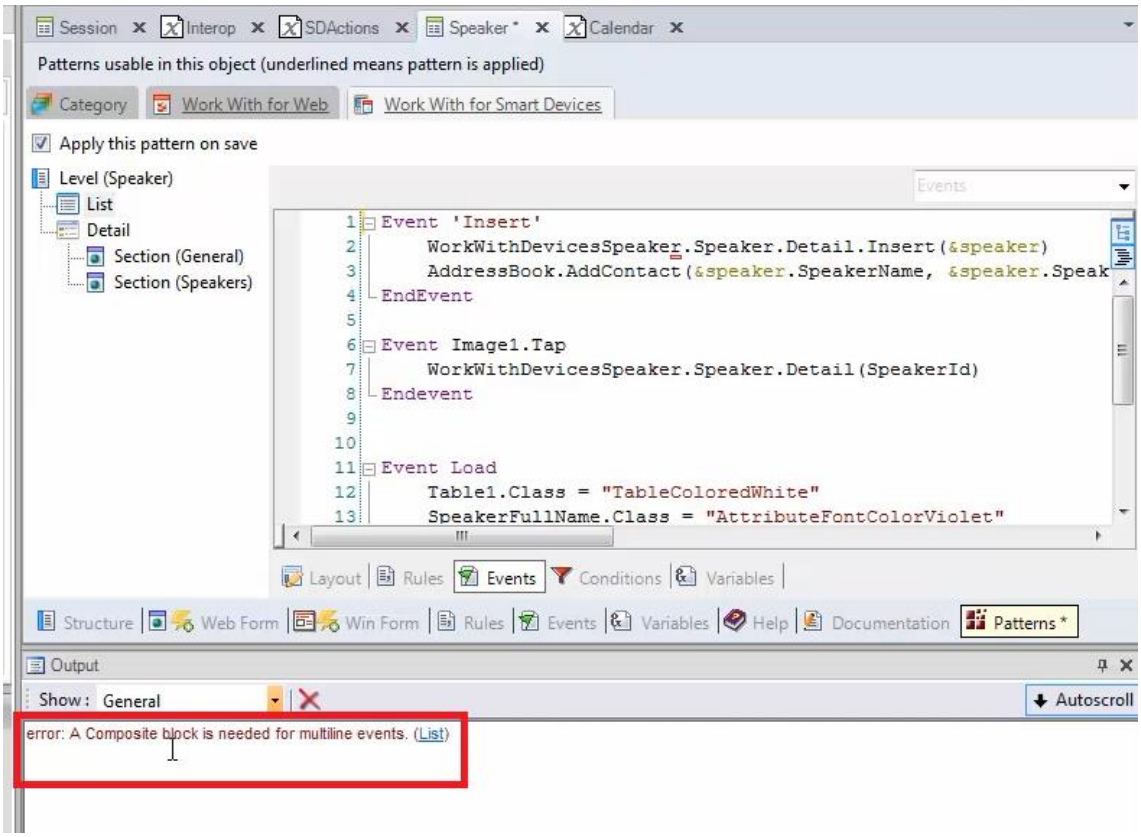
Luego el teléfono..

El que le sigue.. CompanyName.. y como no tenemos ese dato, vamos a pasarle la cadena vacía, puesto que hay que pasar el parámetro aunque no lo tengamos

La foto, será &Speaker.SpeakerImage

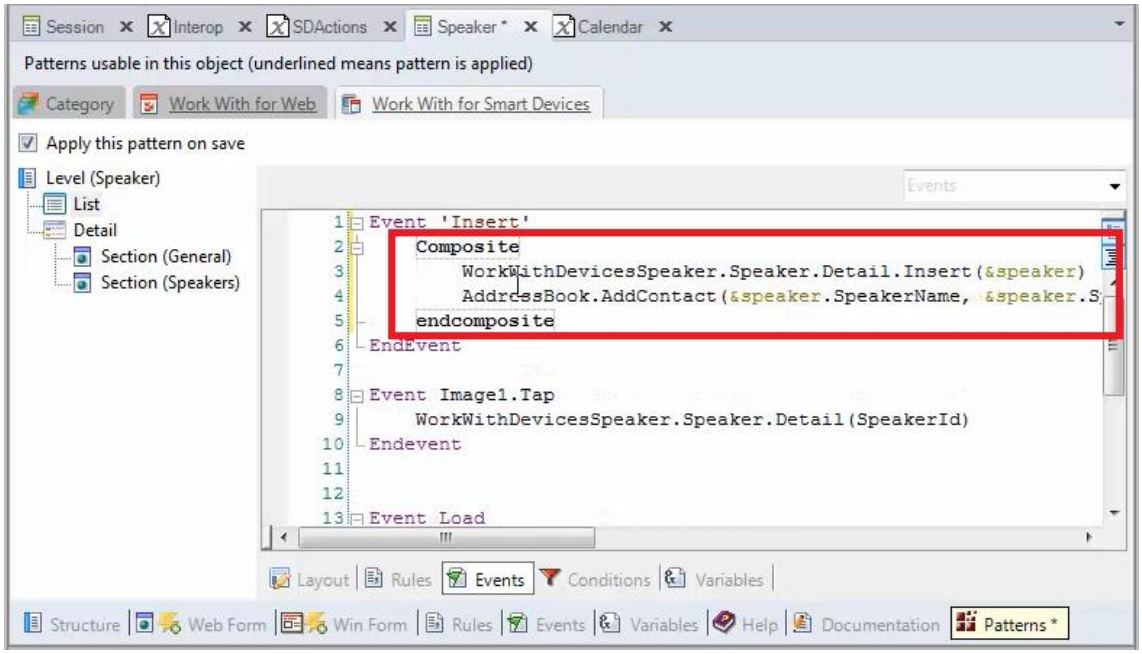
Y el mensaje: cadena vacía

Si ahora intentamos grabar, advirtiéndonos



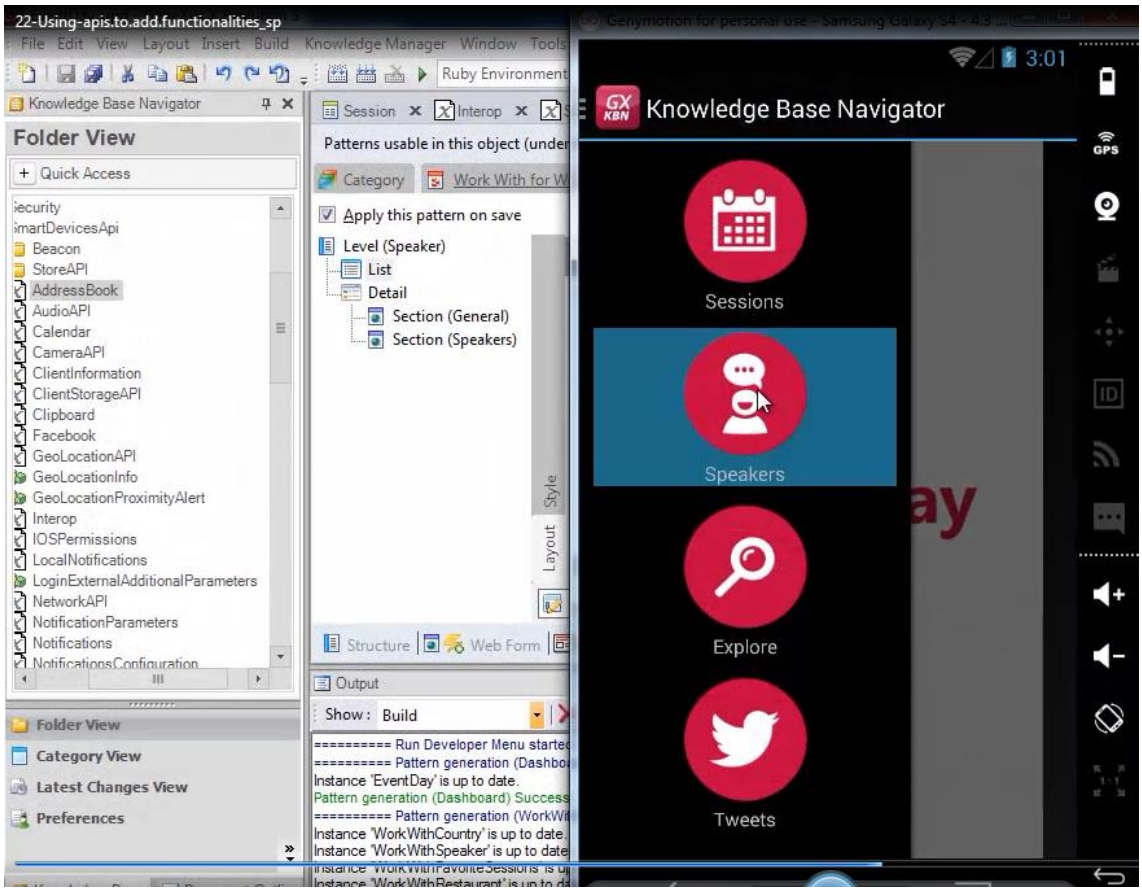
que nos está faltando el bloque Composit.

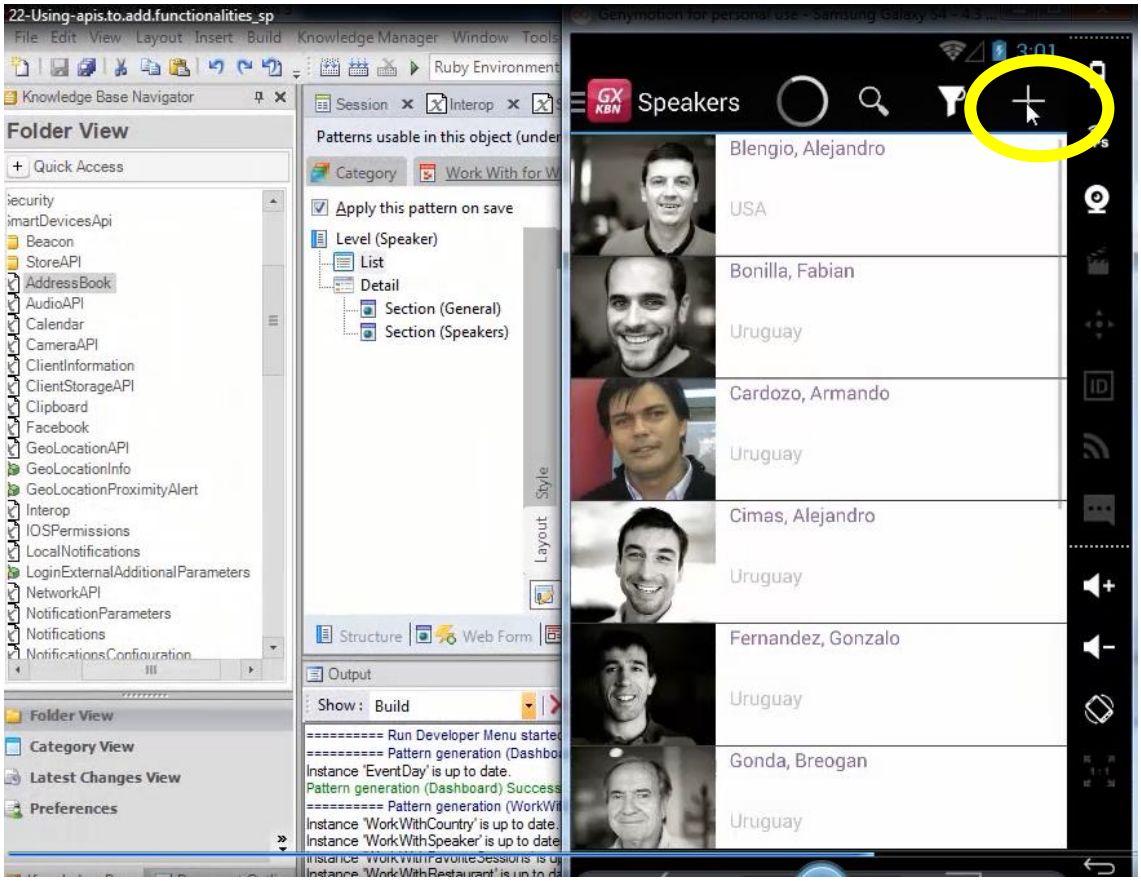
Veremos este comando en otro video, pero ya podemos advertir que toda vez que tengamos 2 invocaciones dentro de un mismo evento, debemos encapsularlas dentro de ese comando.



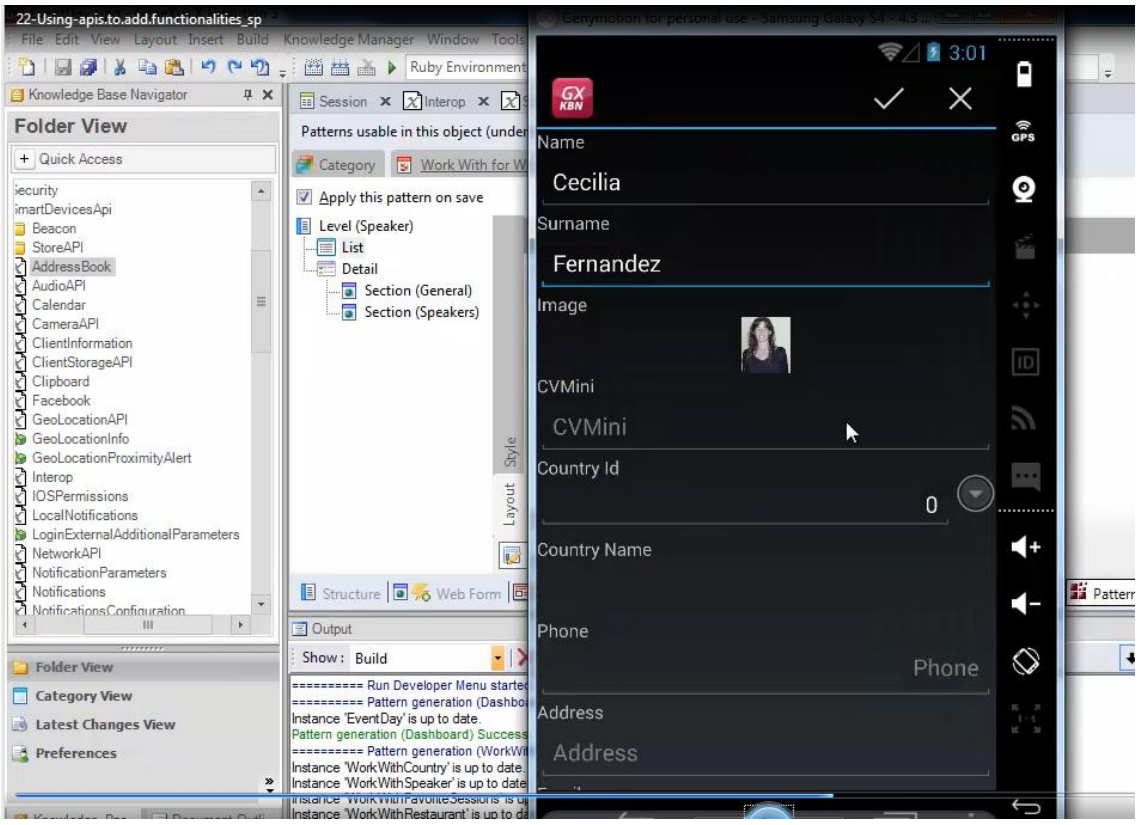
Si ahora grabamos, no vamos a tener problema.

Hagamos ahora un F5 y probemos.





Agregamos un nuevo orador..



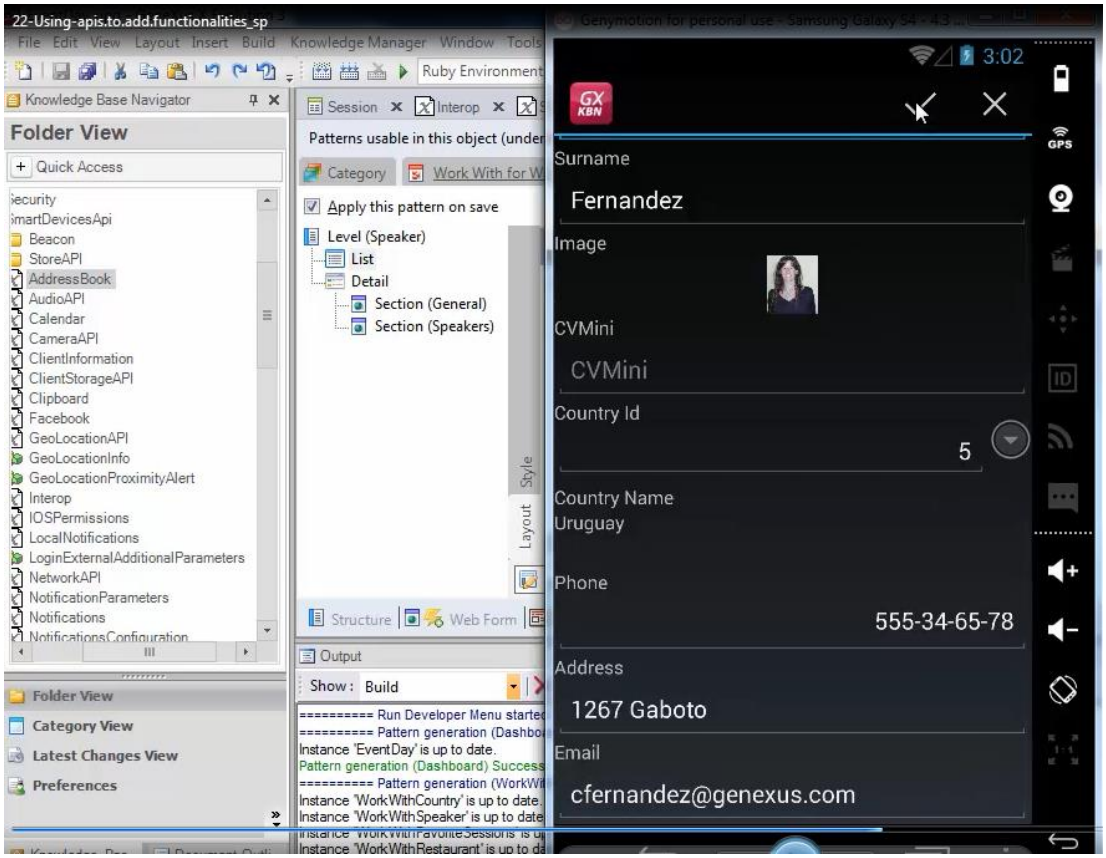
seleccionamos la imagen de la galería de imágenes que tenemos..

vamos a dejar vacío el currículum vitae

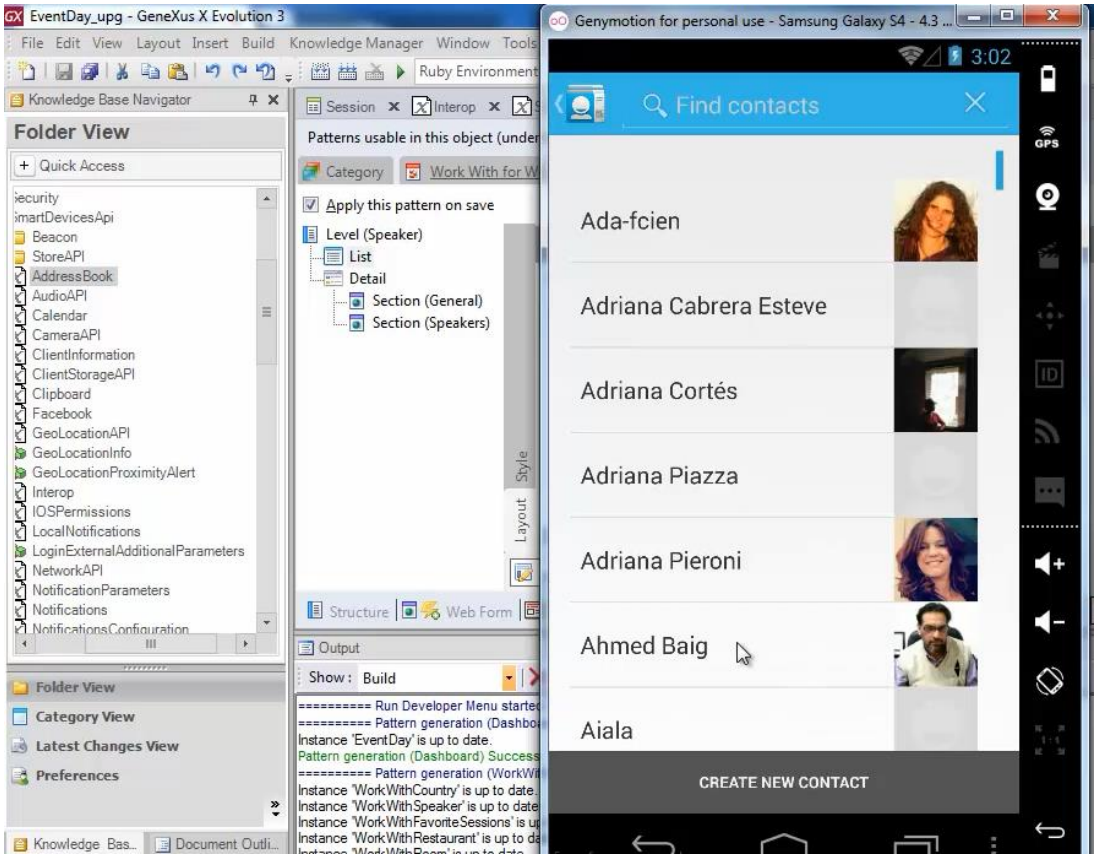
agregamos el país: Uruguay

teléfono.. dirección.. email..

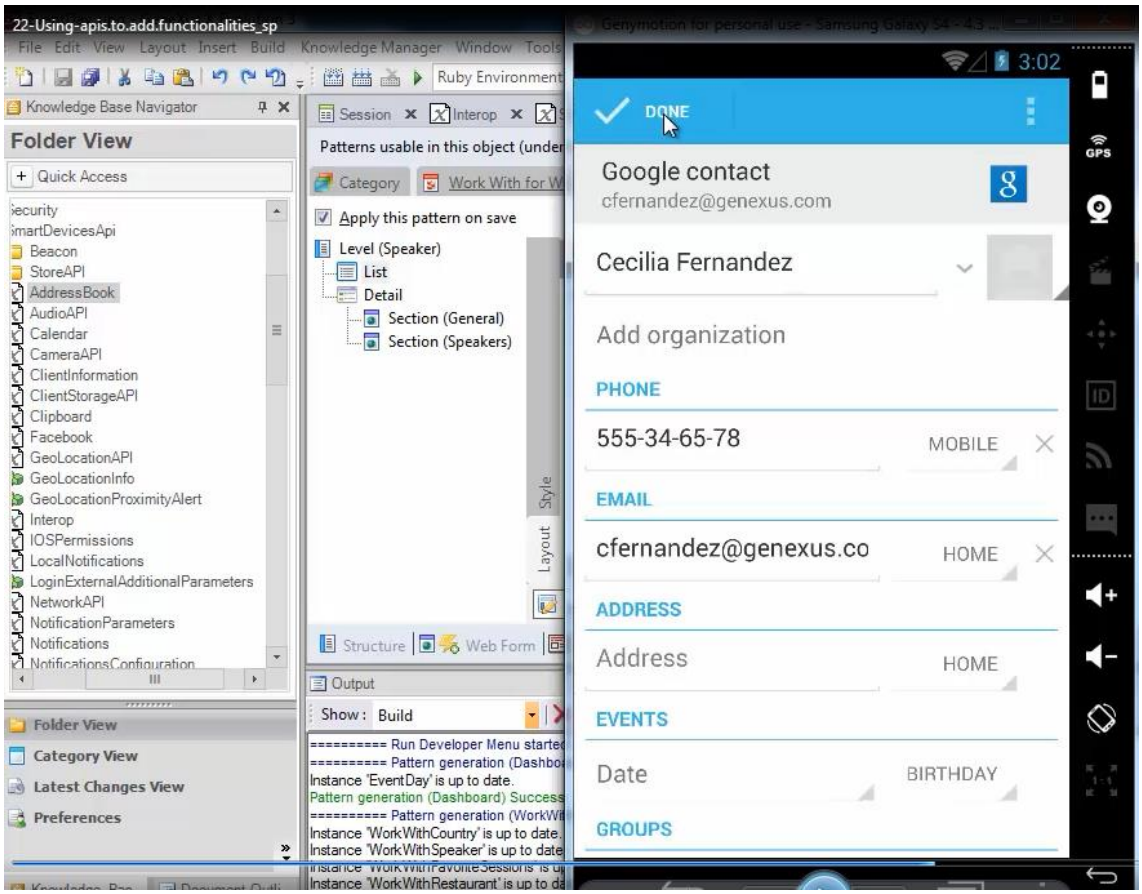
y grabamos



Vemos que fue insertada la información en la base de datos y se nos abre la libreta de direcciones

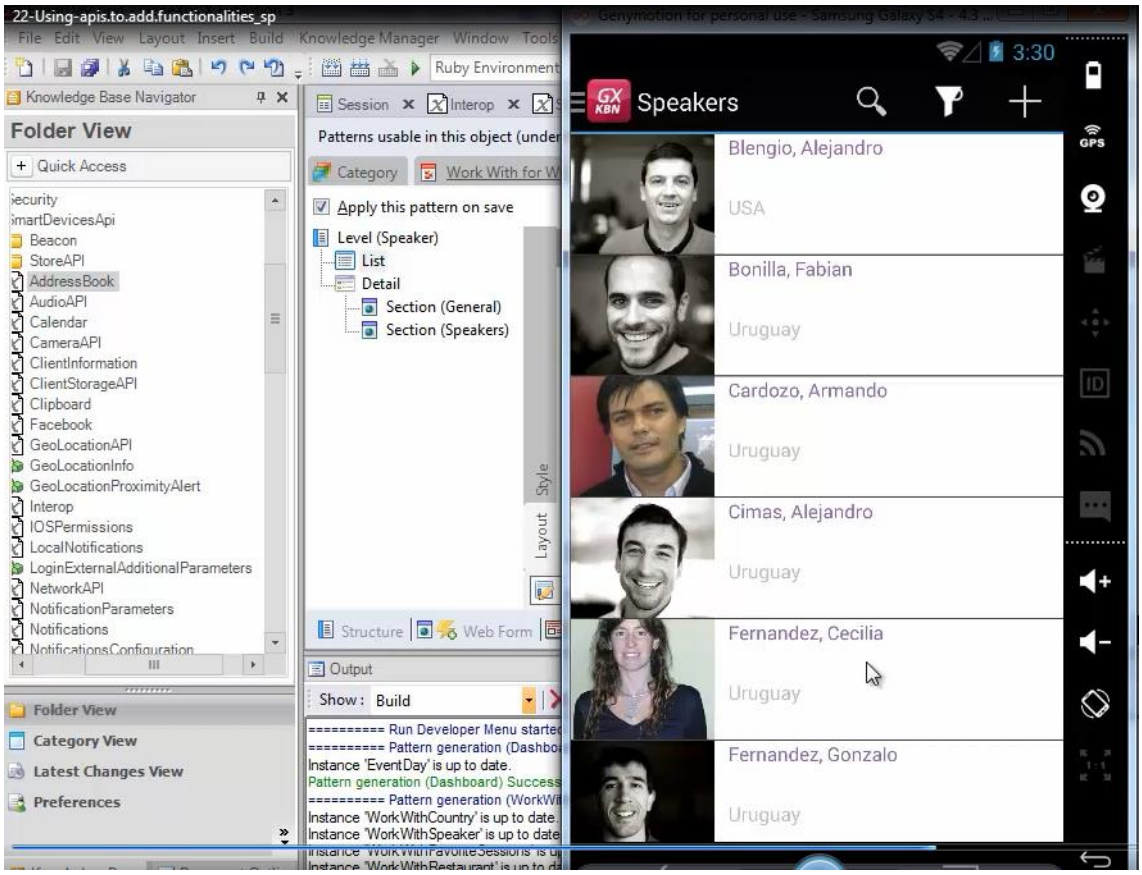


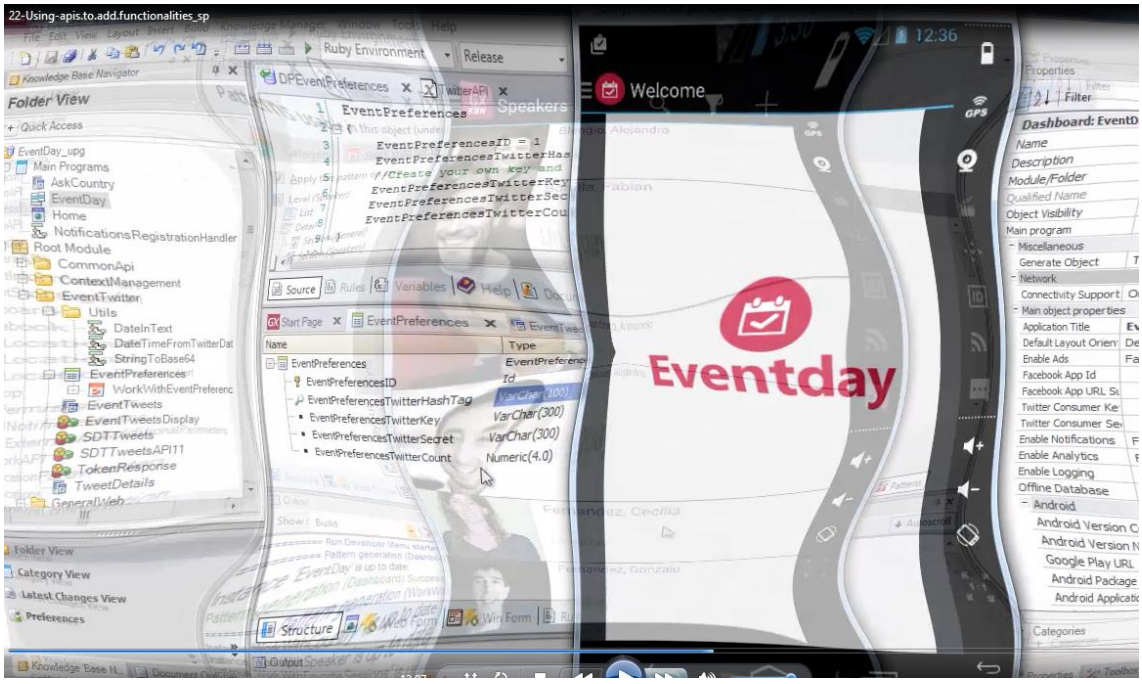
para crear el nuevo contacto con los datos correspondientes



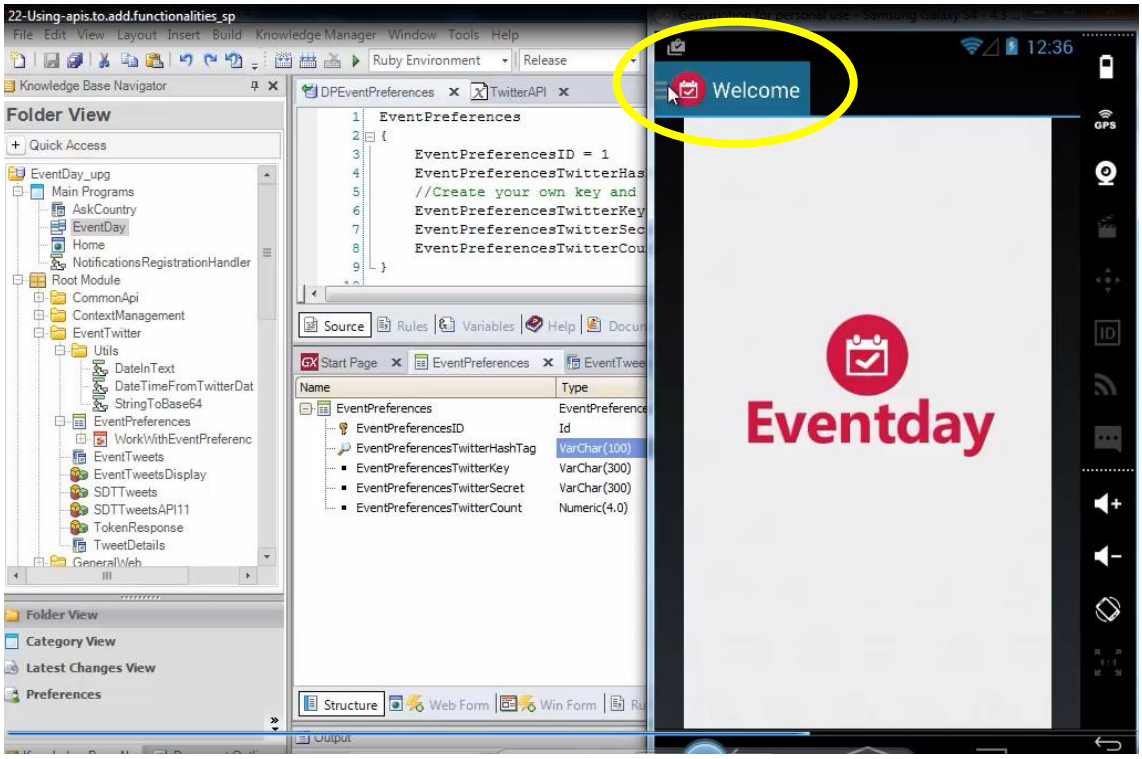
Video filmado con GeneXus X Evolution 3

Vemos que se ha agregado el nuevo orador.

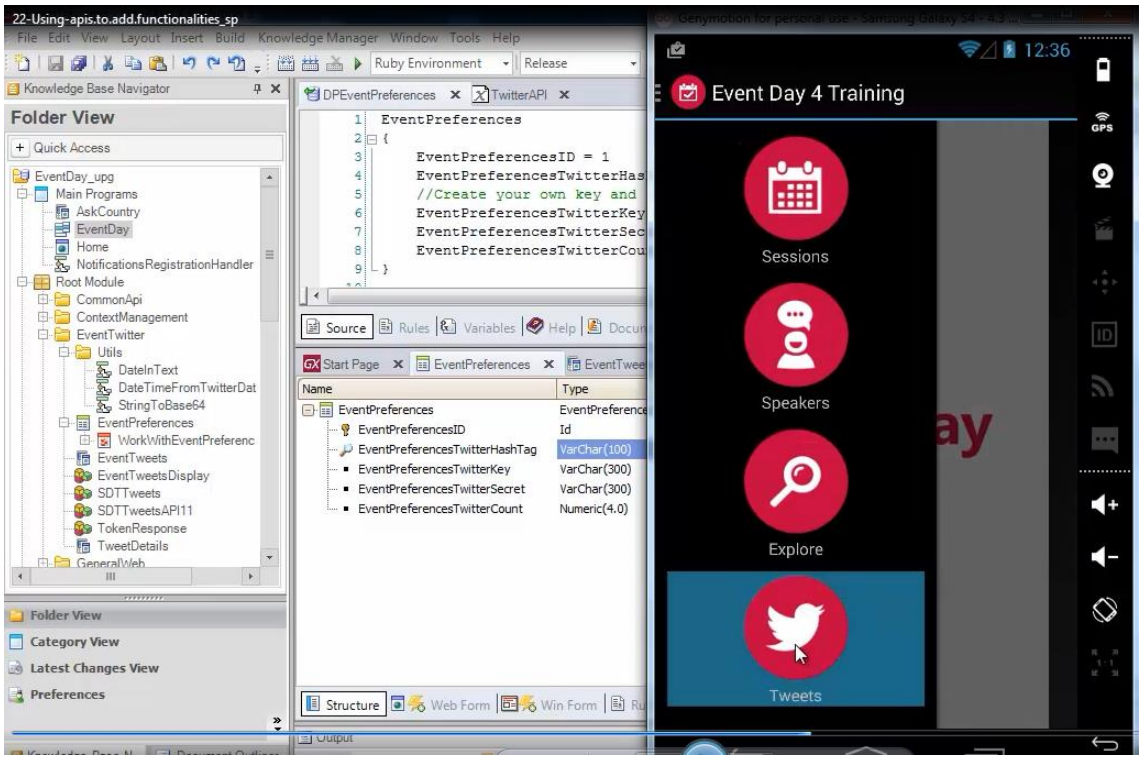




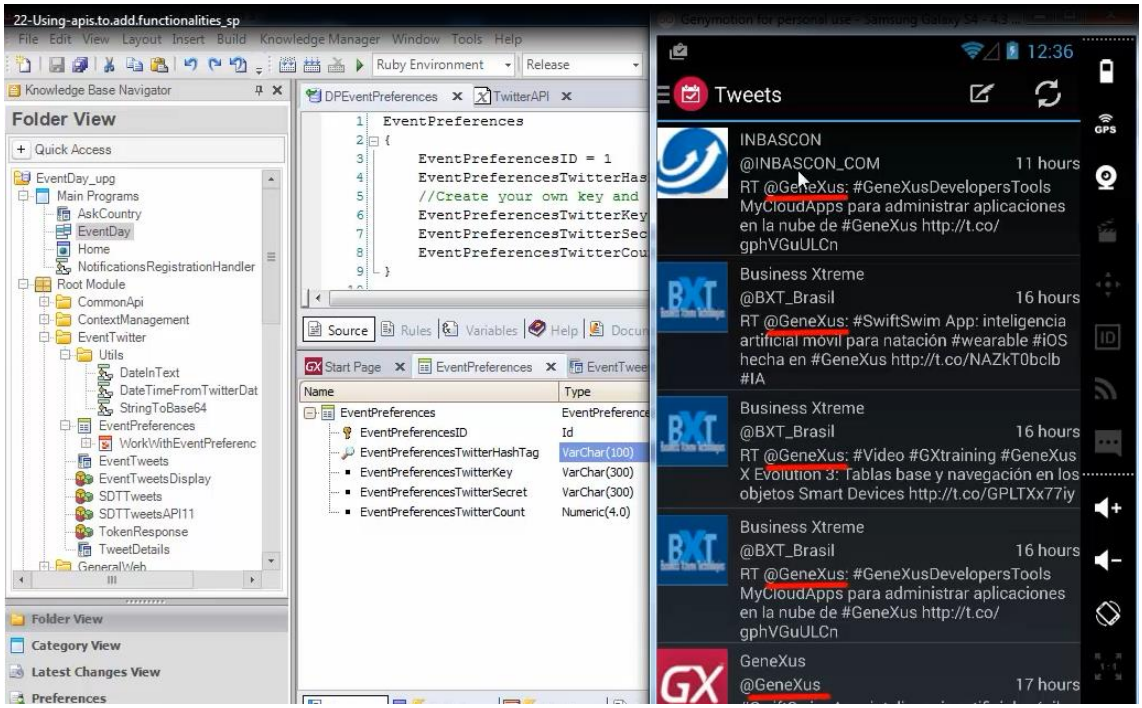
Hemos agregado en nuestra aplicación..



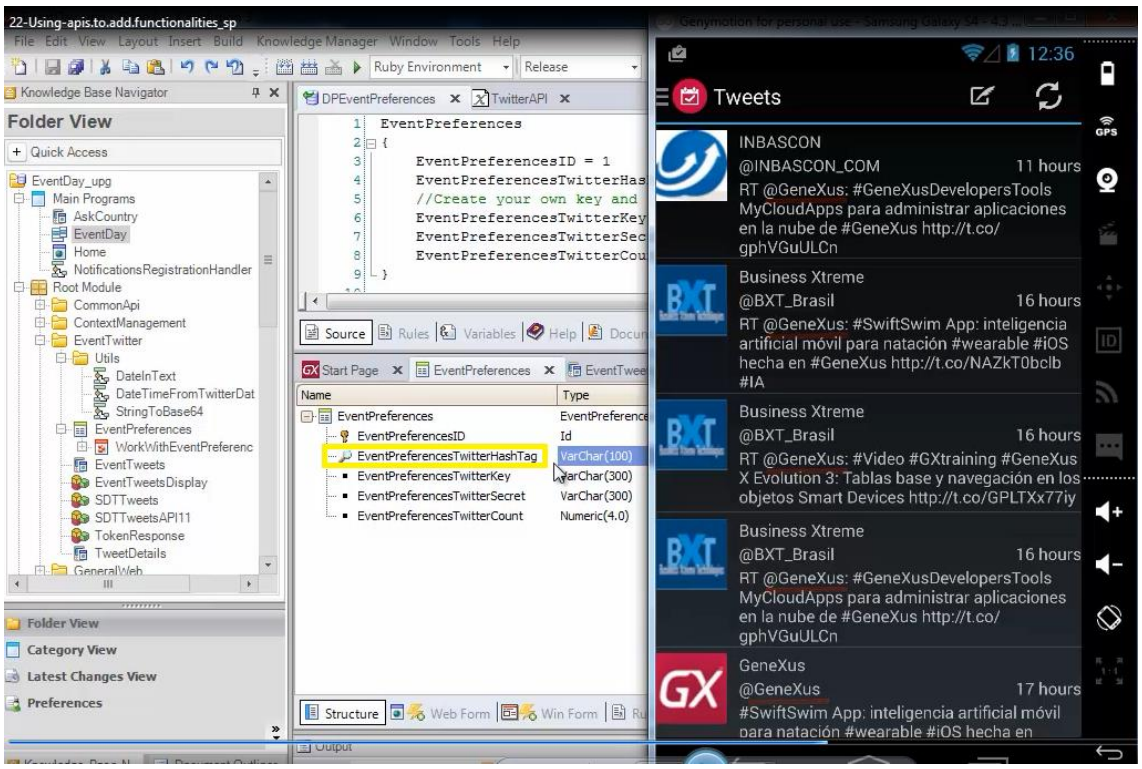
un panel



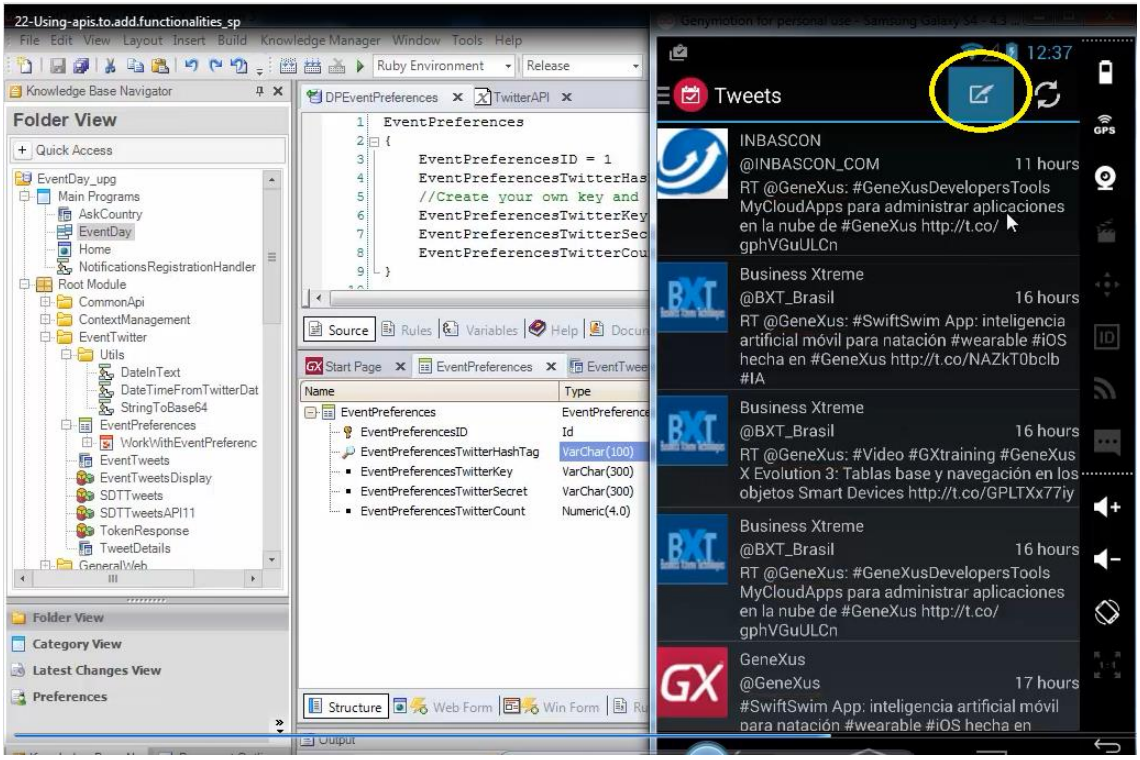
que muestra los últimos tweets



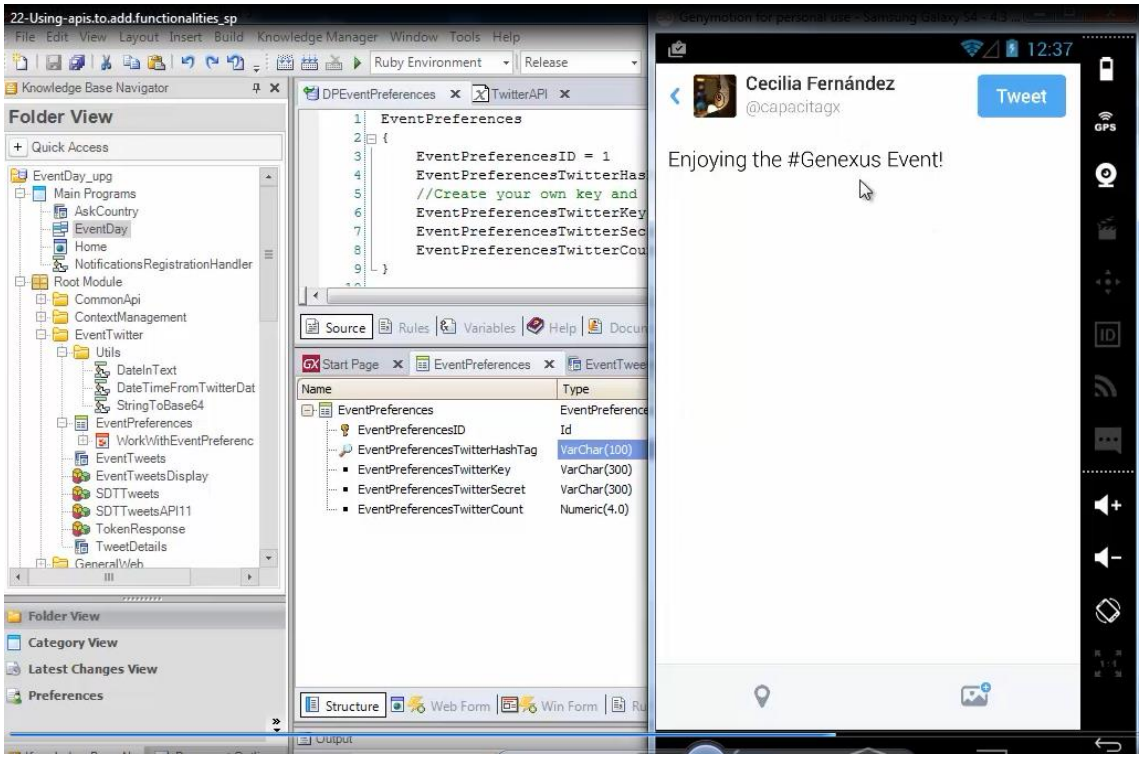
del HashTag que hemos definido en una transacción Preferences de nuestro evento, en el atributo EventPreferencesTwitterHashTag



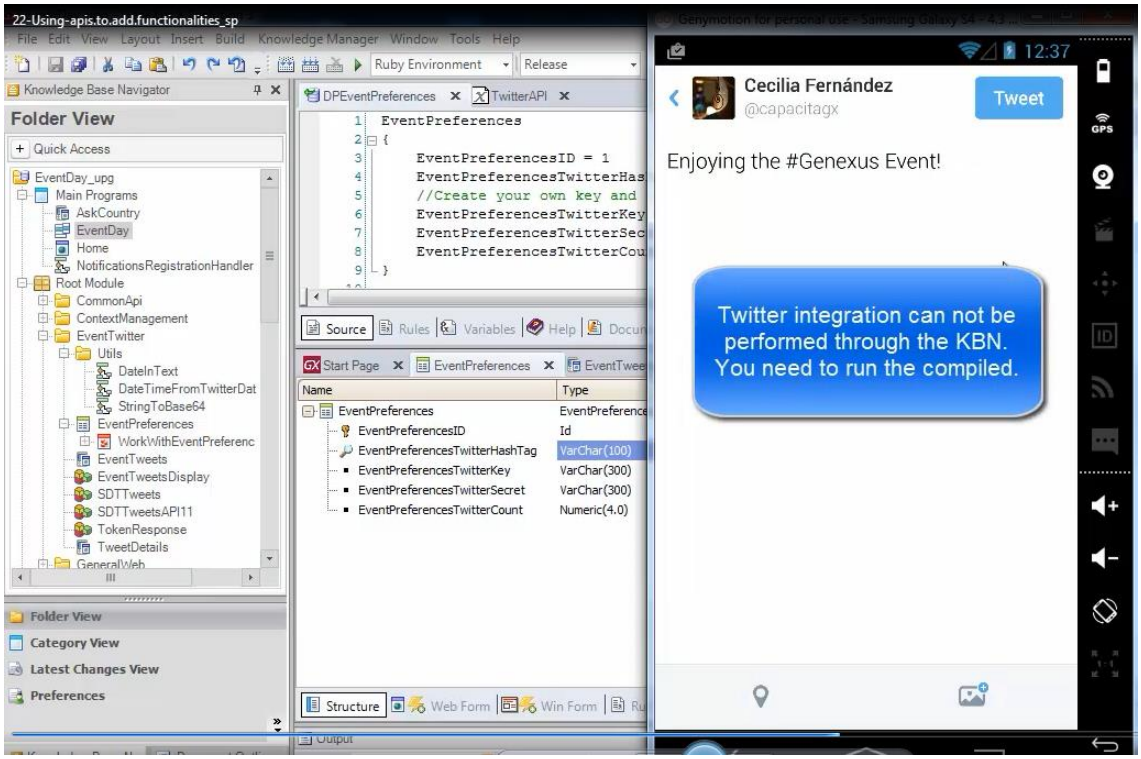
Desde este panel vamos a poder ingresar



un nuevo tweet con ese HashTag definido

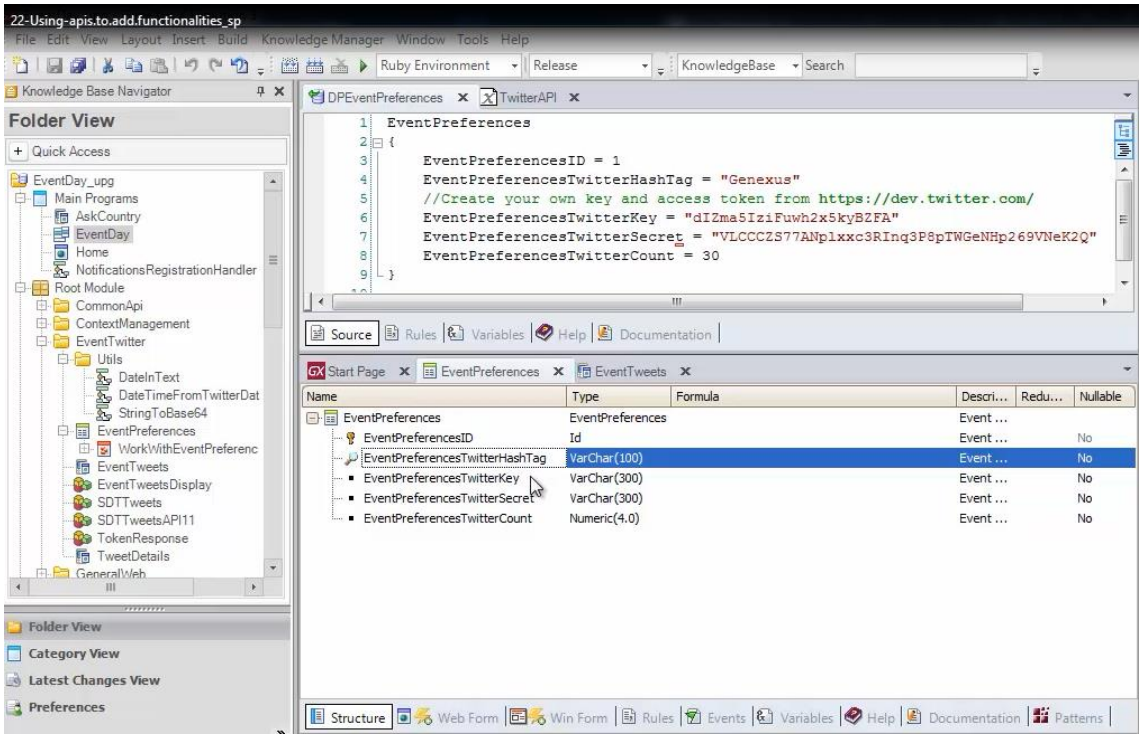


Observemos por tanto que nos estamos integrando con la aplicación de Twitter instalada en el dispositivo.

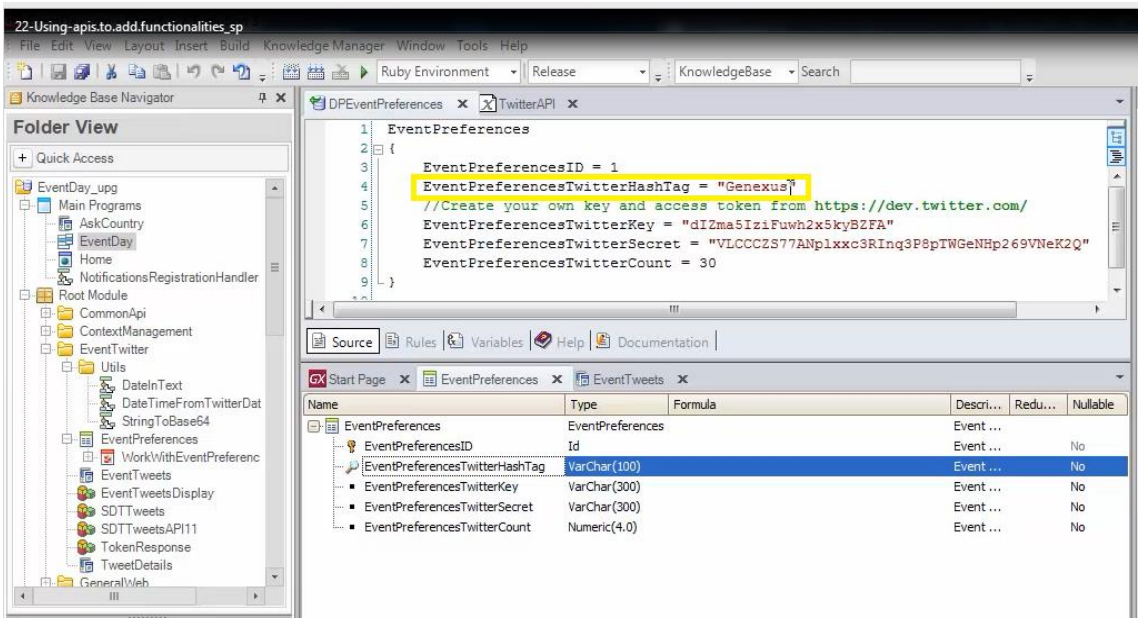


¿Cómo se implementó todo esto?

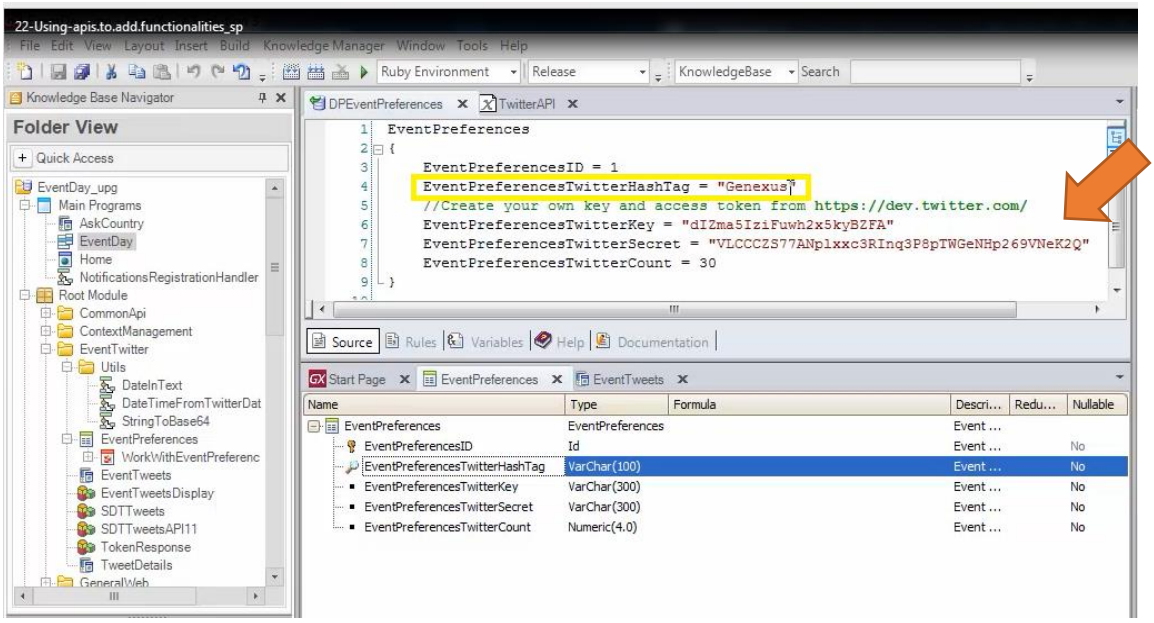
Vayamos a GeneXus.. Veamos entonces que tenemos la transacción donde debemos ingresar estos datos



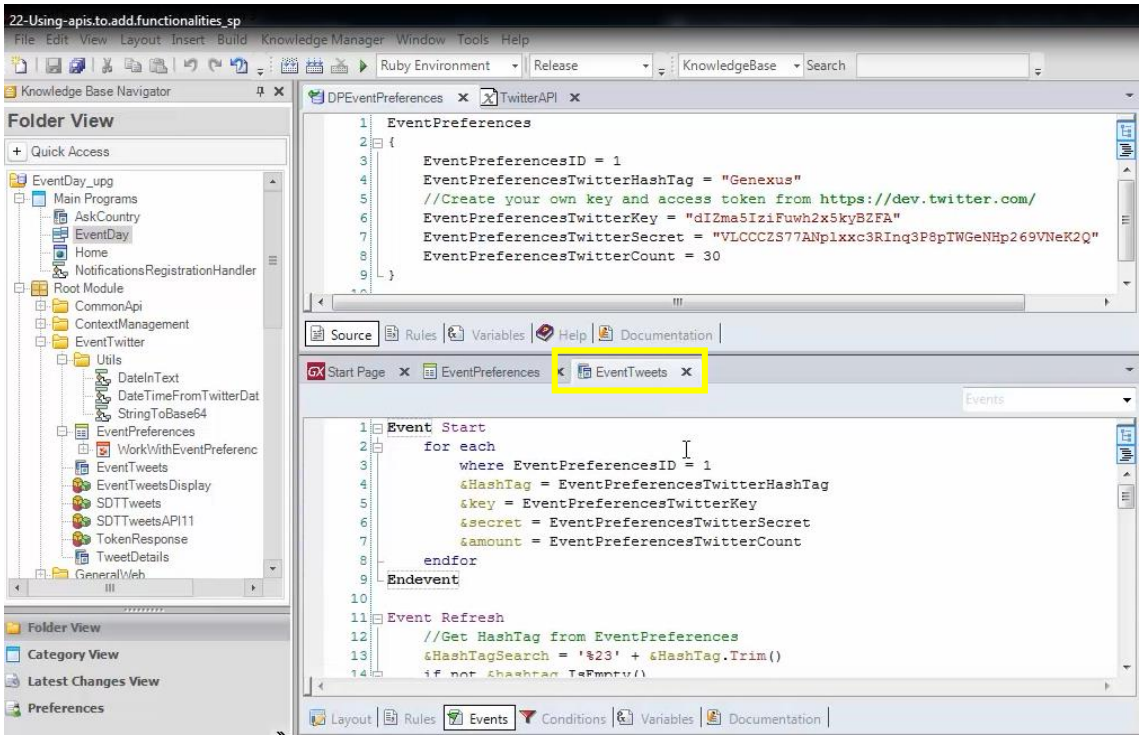
Lo hacemos a través de un data provider

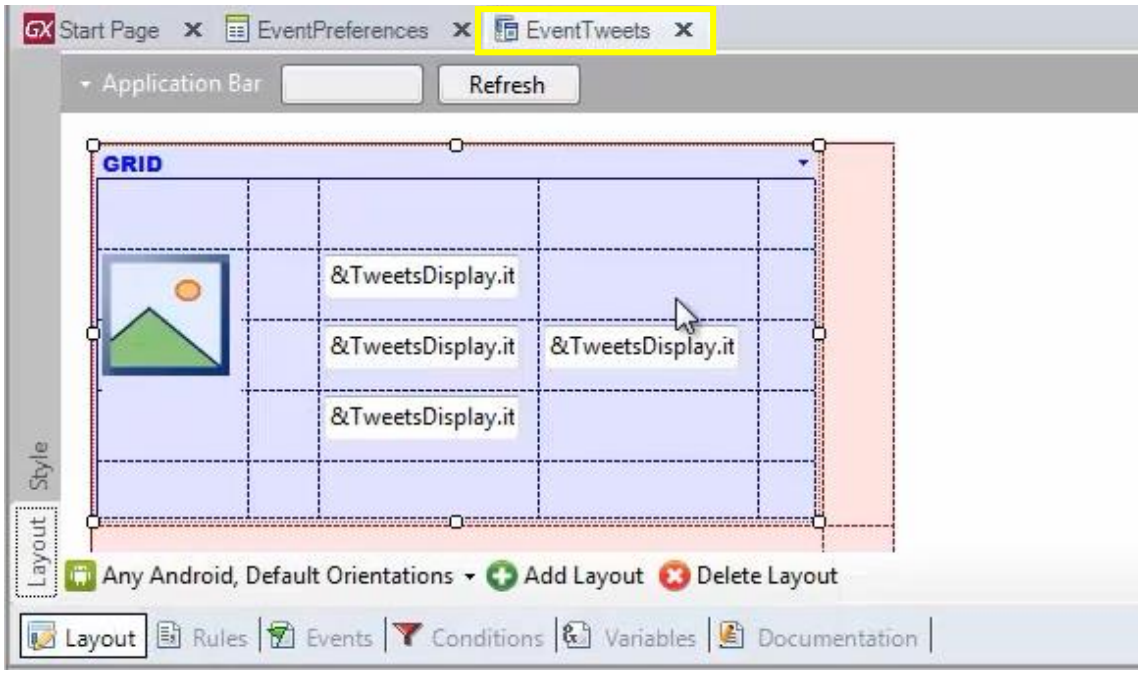


donde definimos como HashTag, GeneXus.. pero aquí sería el HashTag correspondiente al evento que se esté realizando.. y vamos a tener que ir a este sitio de Twitter para los desarrolladores, a pedir una clave y un token para poder comunicarnos con la aplicación. Aquí estamos usando el de GeneXus.

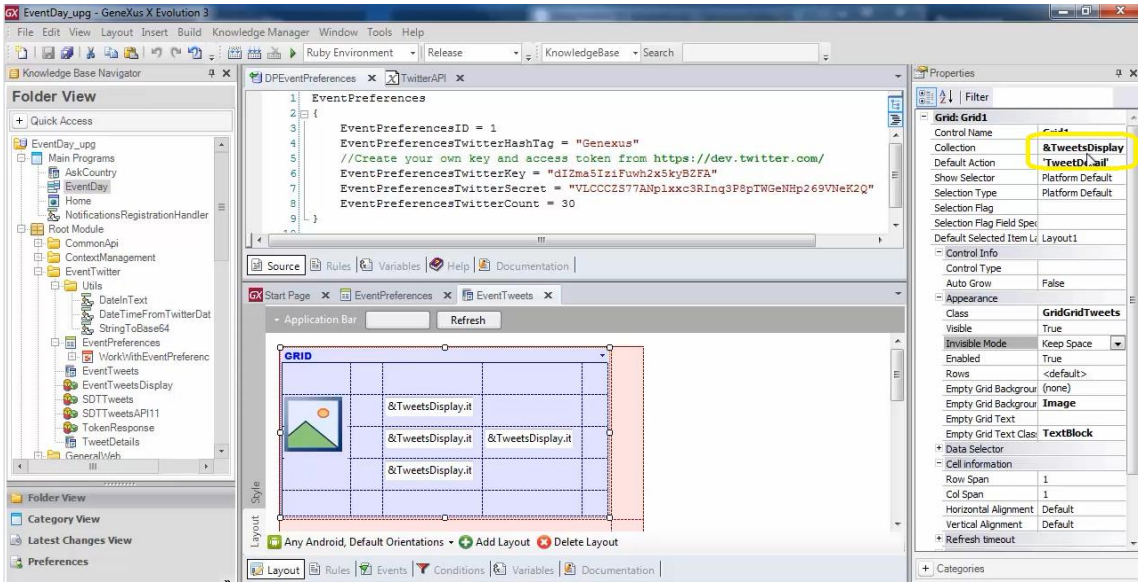


Luego, vamos al panel que implementa todo esto



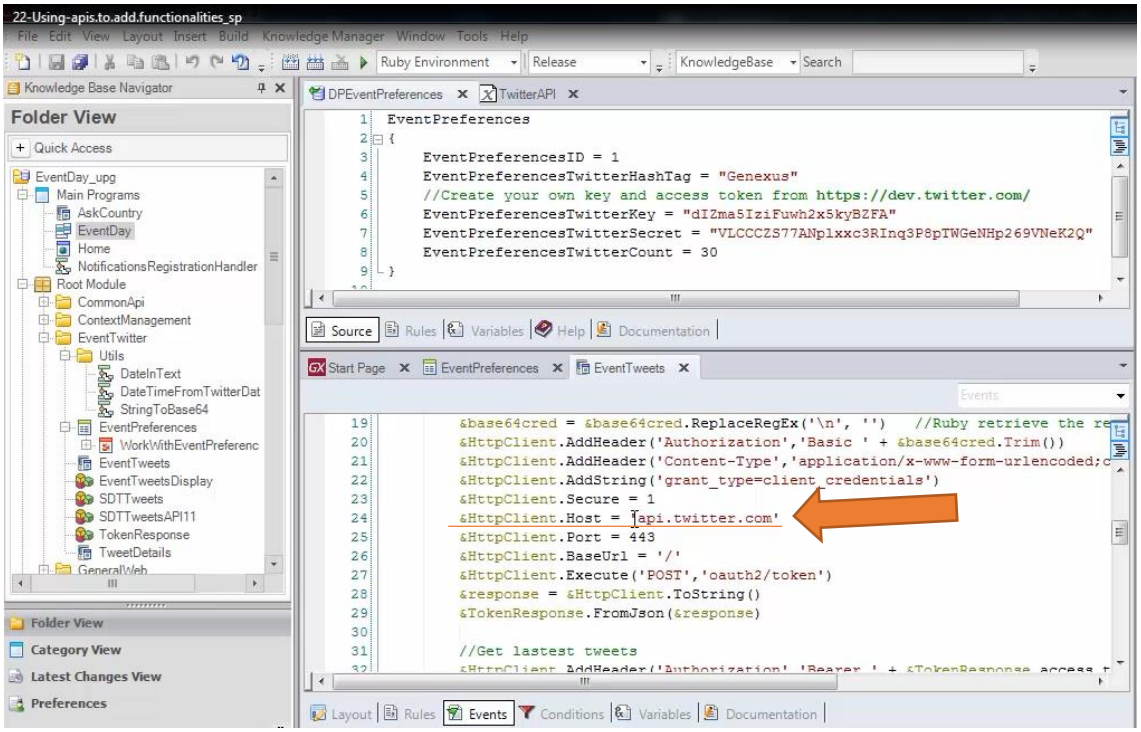


que hemos definido con un grid basado en una variable de tipo colección &TweetsDisplay



que es la que va a tener que ser cargada entonces con los últimos 30 tweets de ese hashtag.

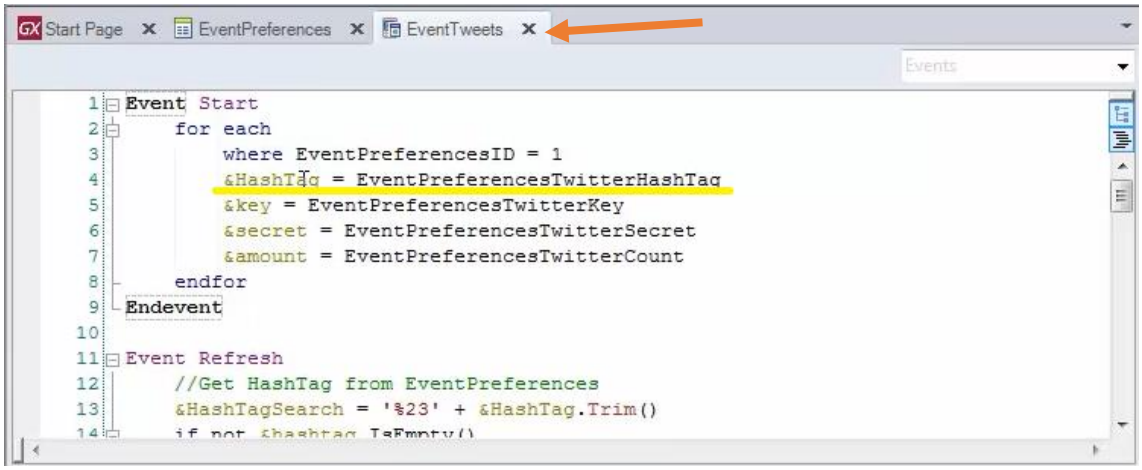
Eso se realiza en el evento Refresh, donde tenemos que comunicarnos



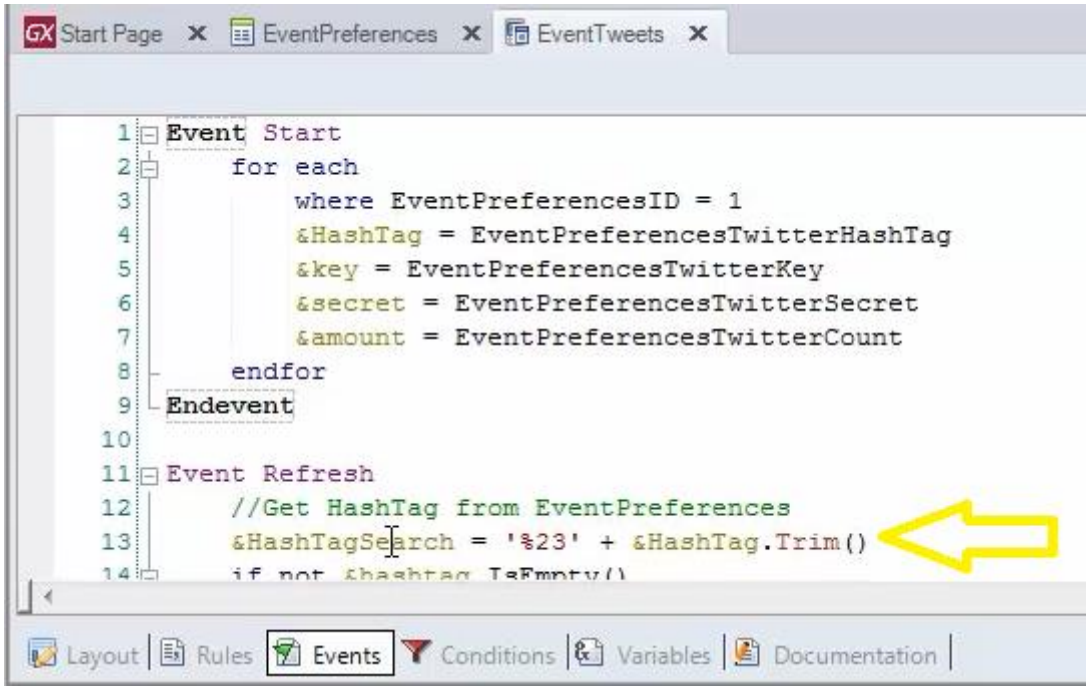
con este host --> api.twitter.com

para obtener el token

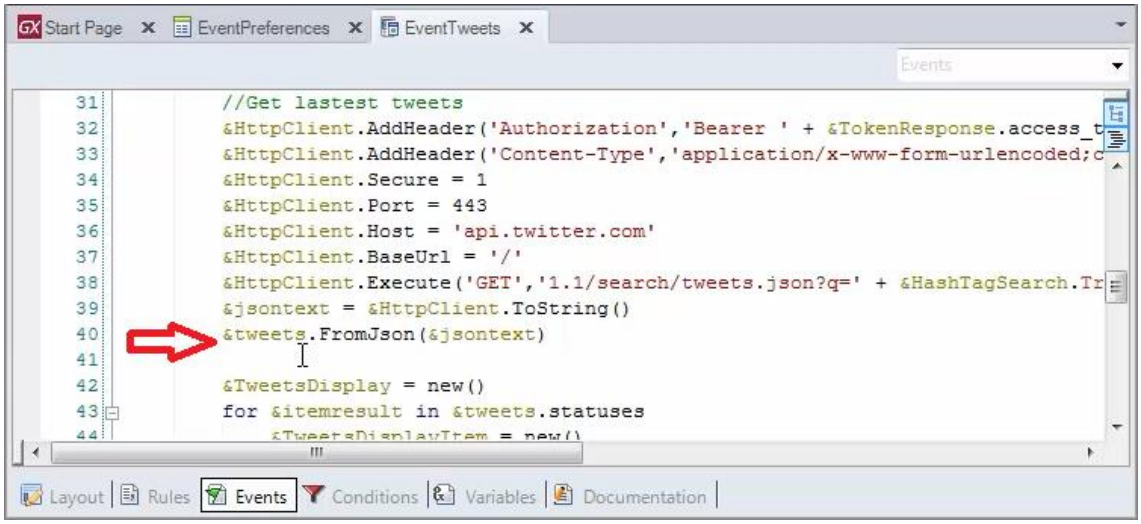
y luego, con el hashtag correspondiente, que fue obtenido en el evento start



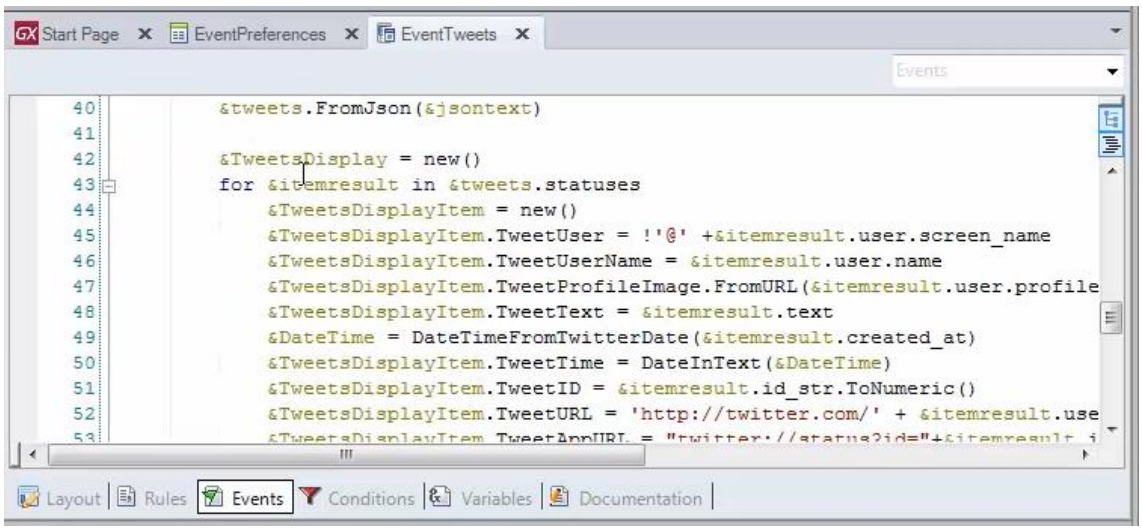
tomándolo del registro de la tabla correspondiente.. y aquí vemos que fue reformateado de esta manera:



Decíamos, con eso entonces: el token, el hashtag y la cantidad de tweets que queremos recuperar, lo hacemos en esta variable &tweets

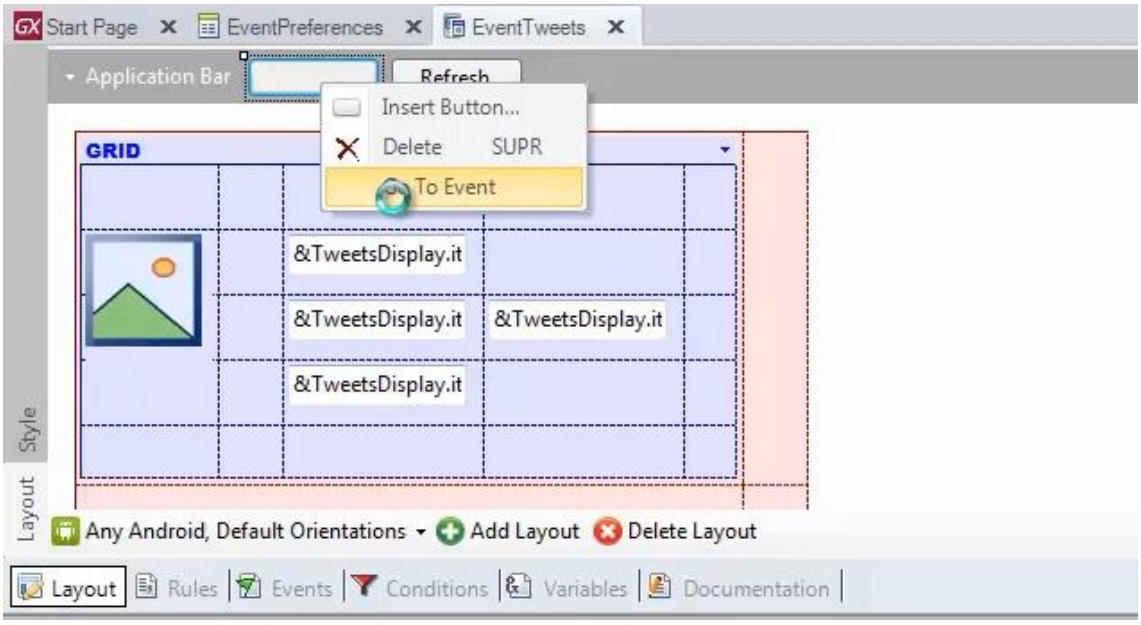


Variable que recorremos aquí

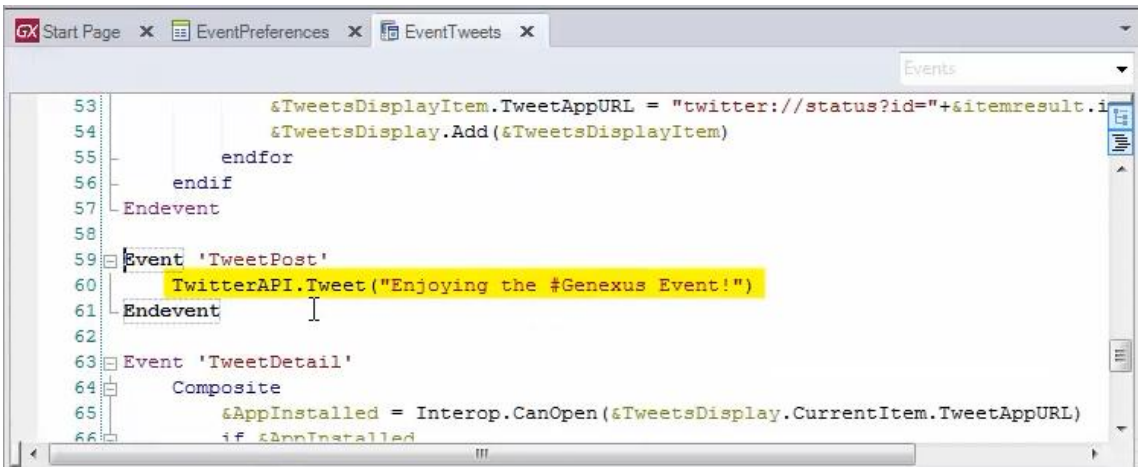


cargando entonces la variable colección que se va a desplegar en el grid.

Y luego, tenemos el botón

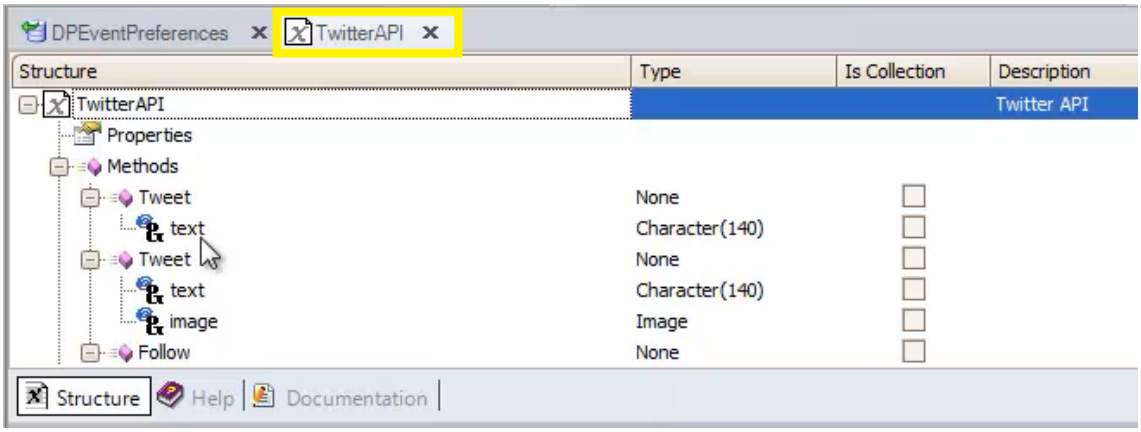


que permite que el usuario ingrese un tweet



Observemos que allí estamos usando la api TwitterAPI-método Tweet.

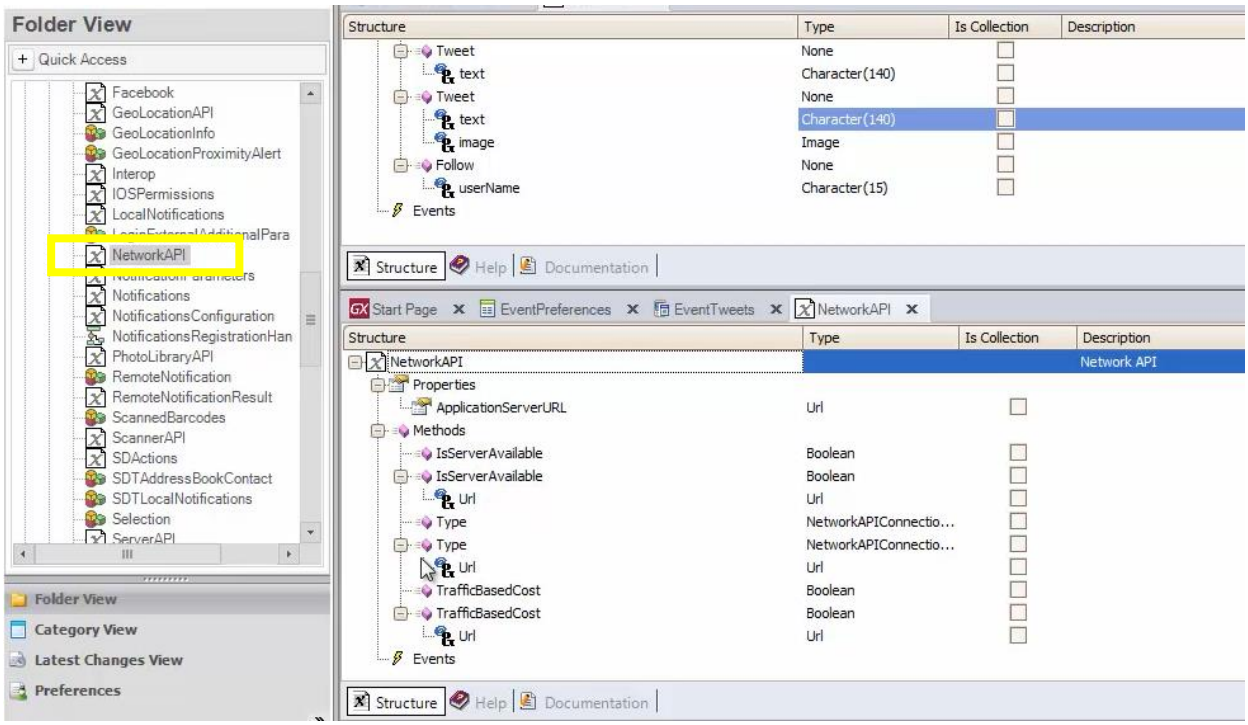
Si observamos la api, nos ofrece 2 métodos tweet



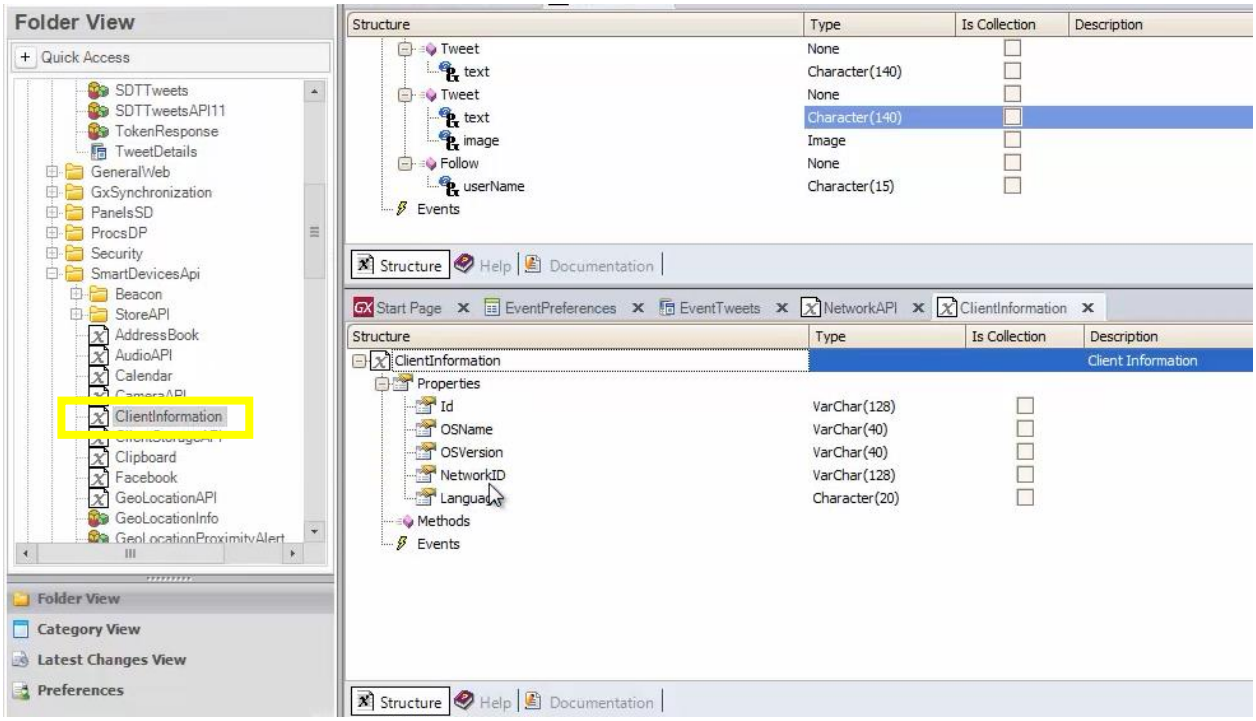
en uno solamente enviamos el mensaje a ser tweeteado y en el otro el texto y la imagen.

Y por otro lado tenemos el método Follow.

Tenemos unas cuantas apis más. Por ejemplo, una que nos permite saber si hay conexión con el servidor en un momento dado.. y el tipo de conexión..

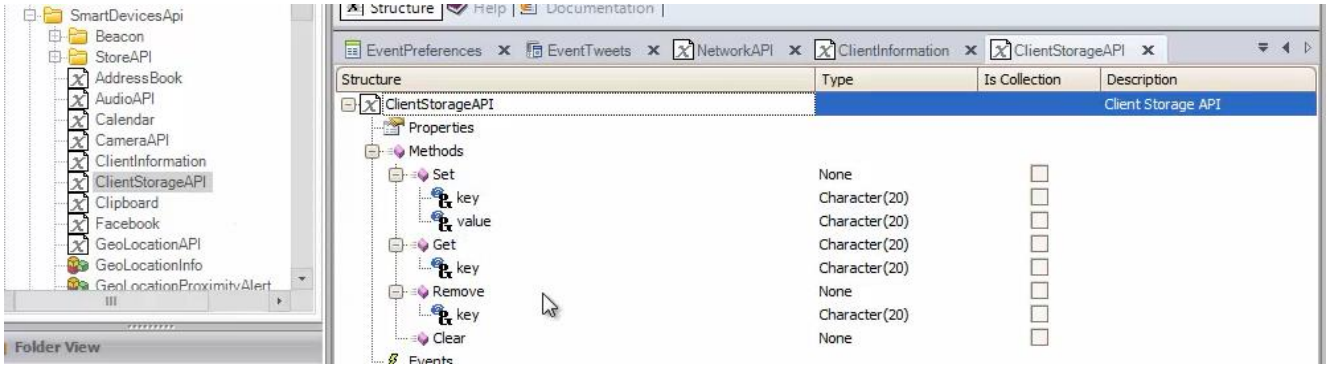


Otra para obtener la información del dispositivo que está ejecutando la aplicación



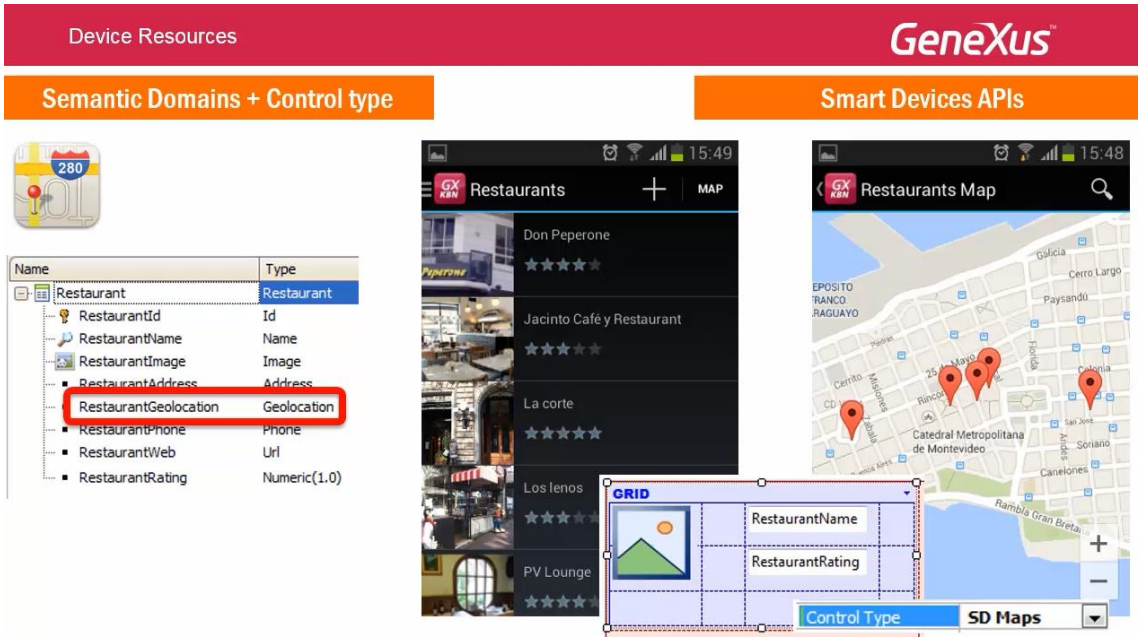
Su Id, sistema operativo, versión, lenguaje..

Una que permite definir variables globales para el dispositivo

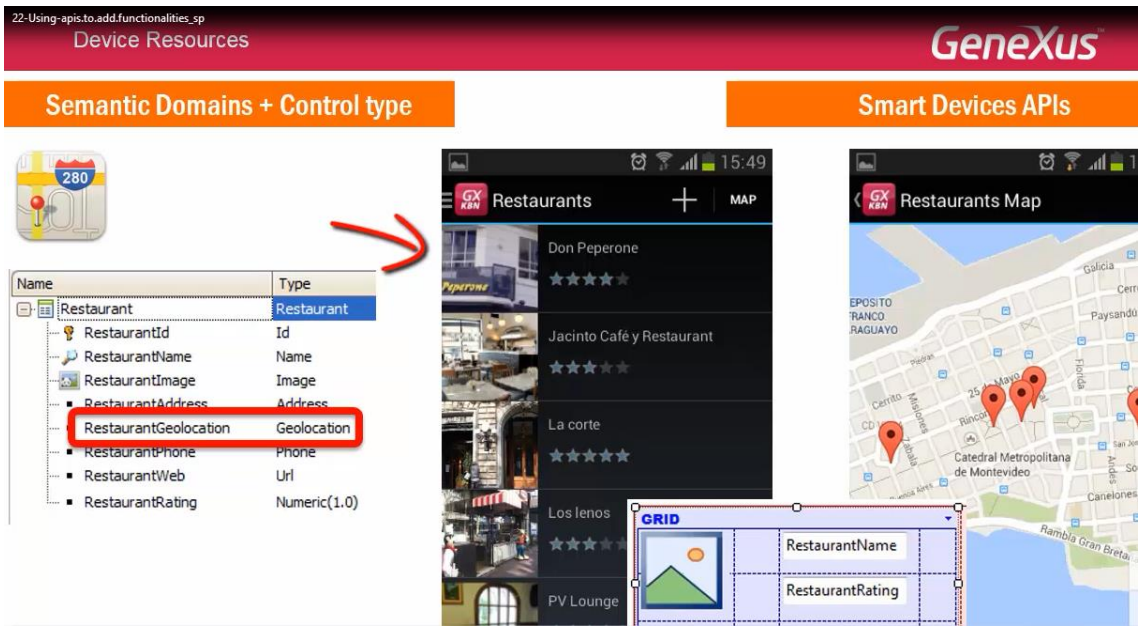


Una para integrarnos con Facebook, otra muy utilizada para trabajar con la geolocalización

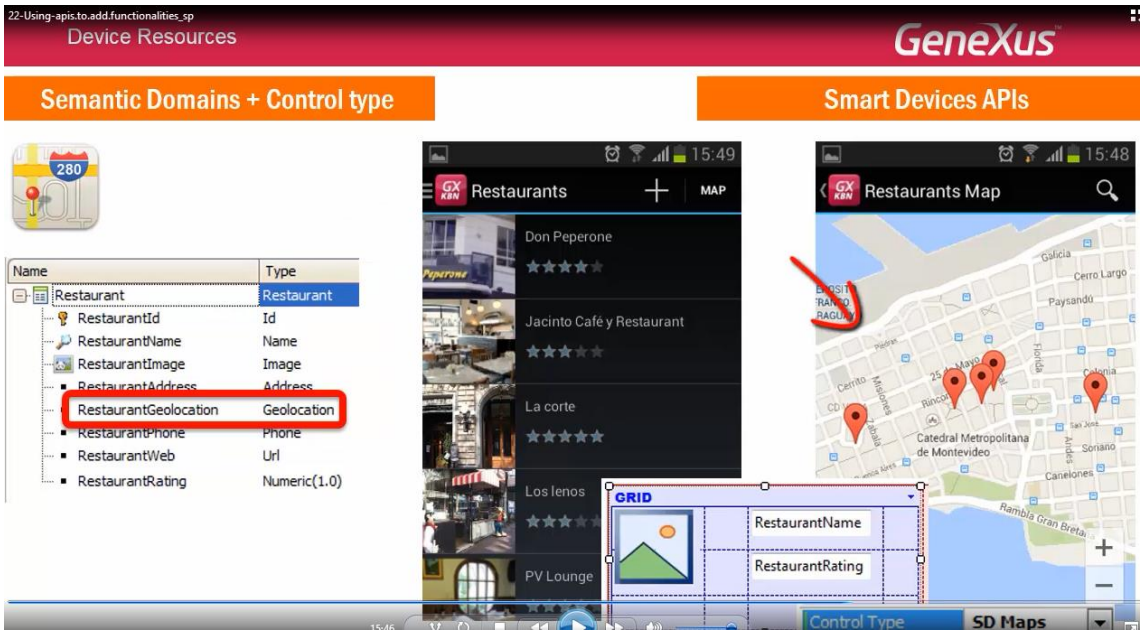
Ya usábamos el dominio semántico Geolocation



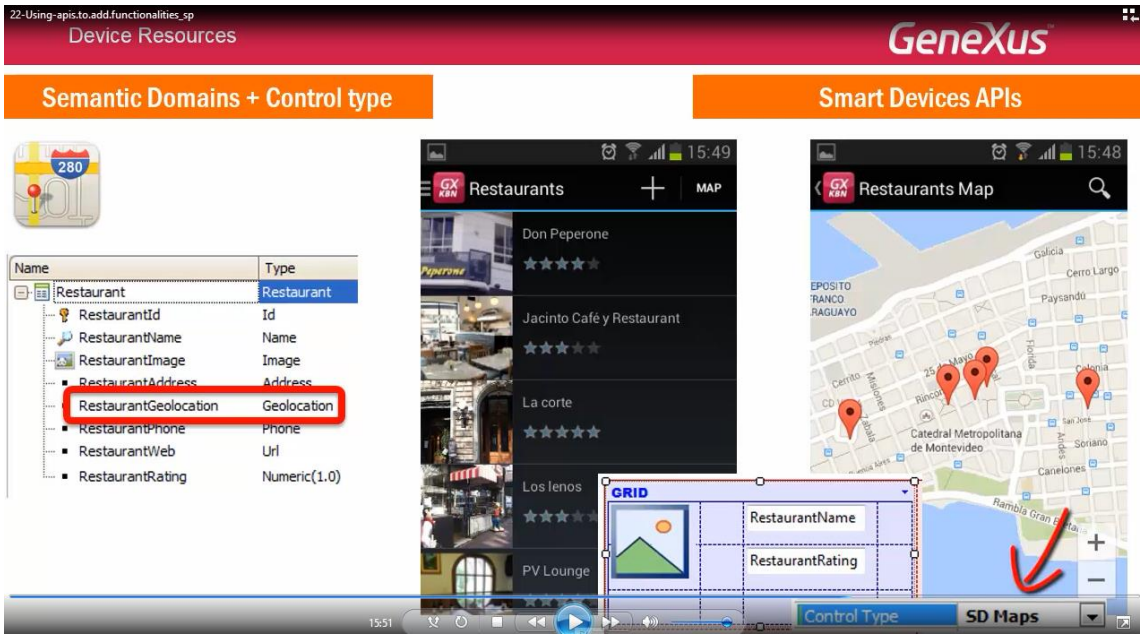
cuando queríamos presentar los restaurants, no como un List standard



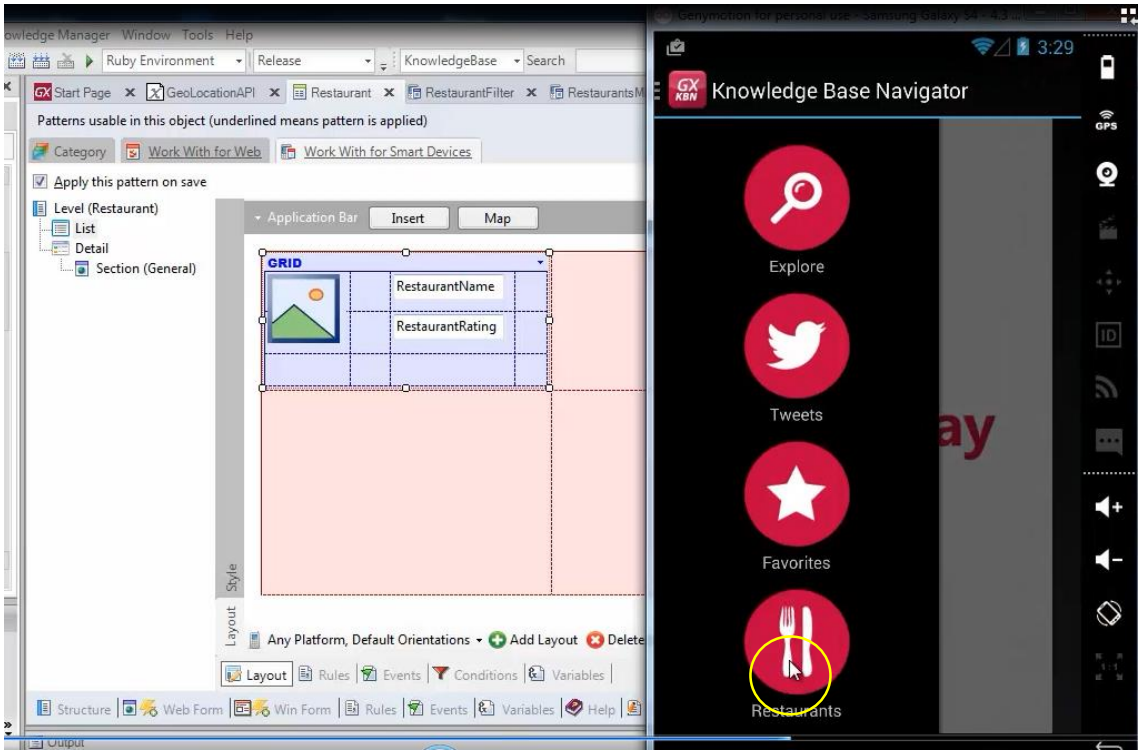
sino como puntos en un mapa



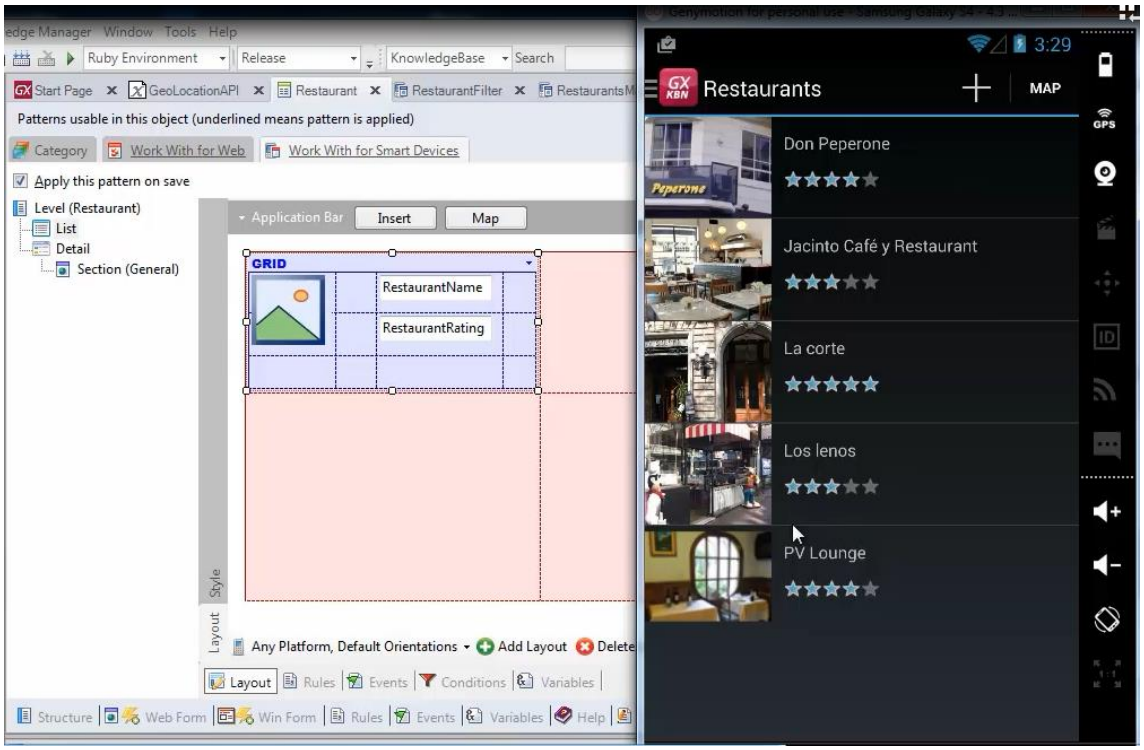
Para ello nos alcanzaba con cambiar el tipo de control del grid por: SDMaps



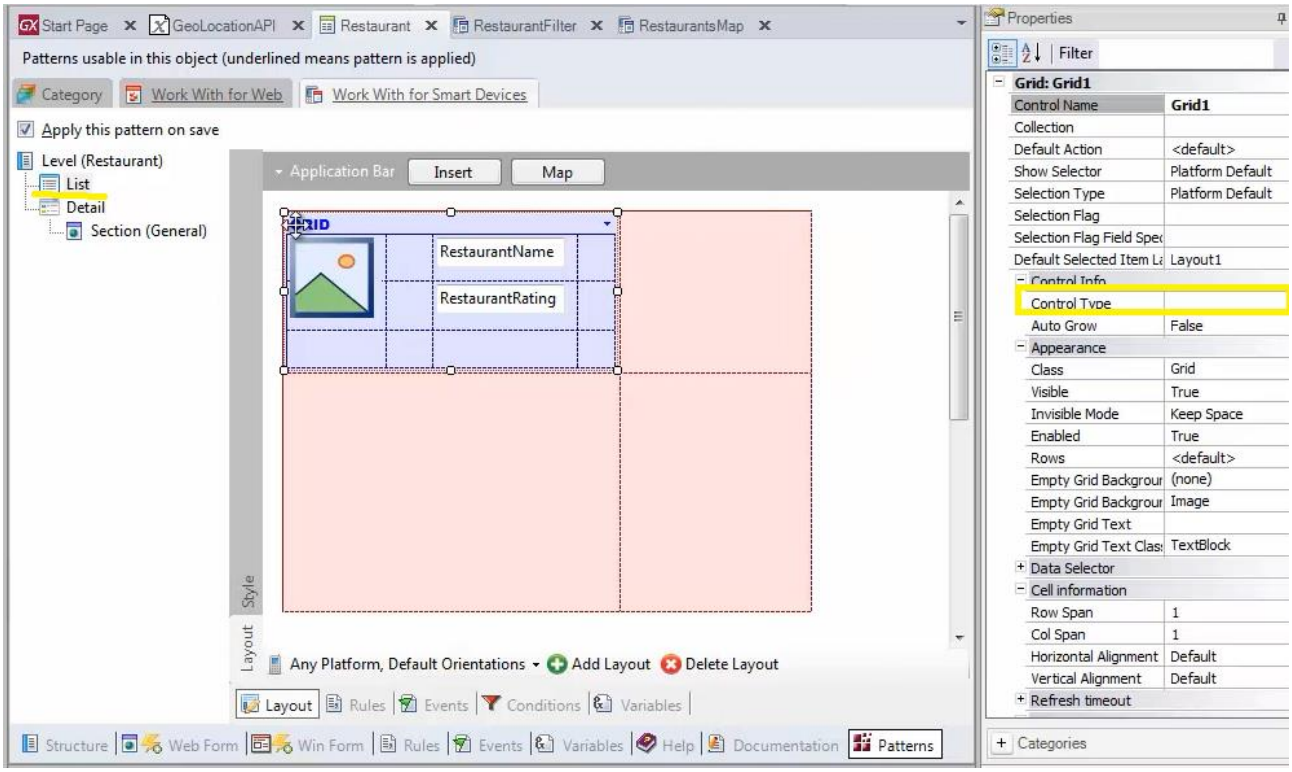
En nuestra aplicación, tenemos



Tenemos implementado el List de restaurants

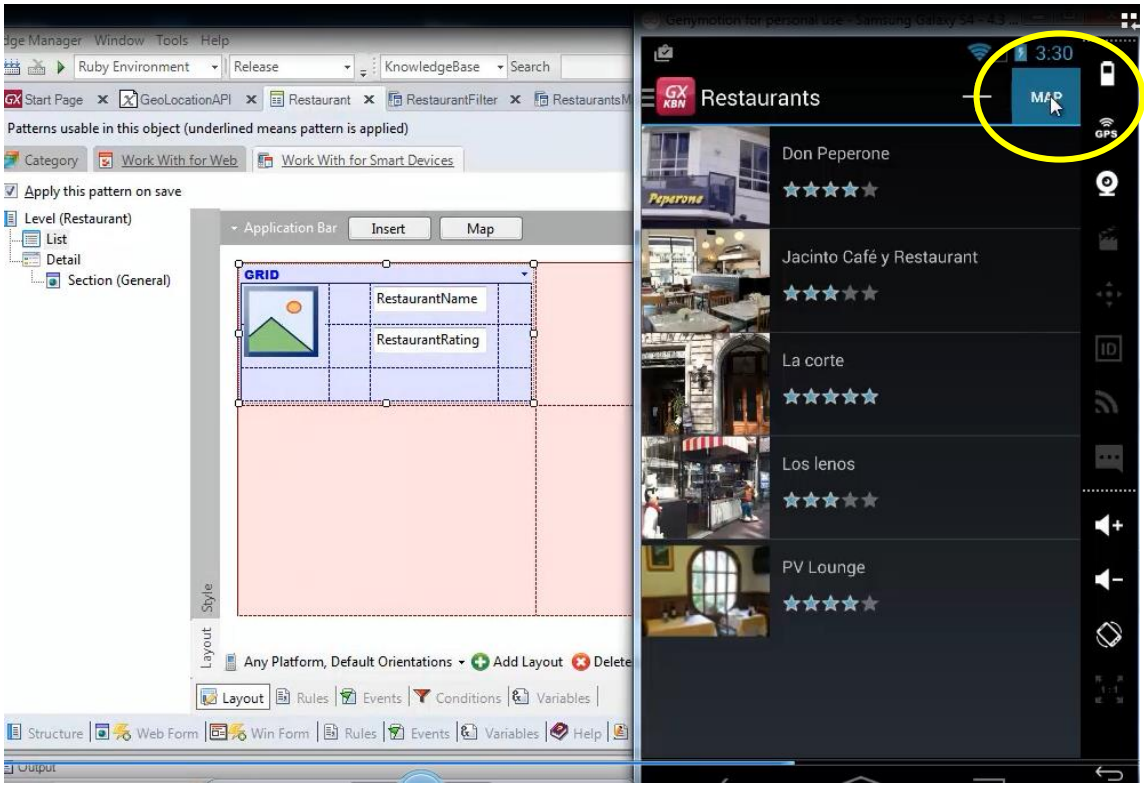


que corresponde al nodo del work with

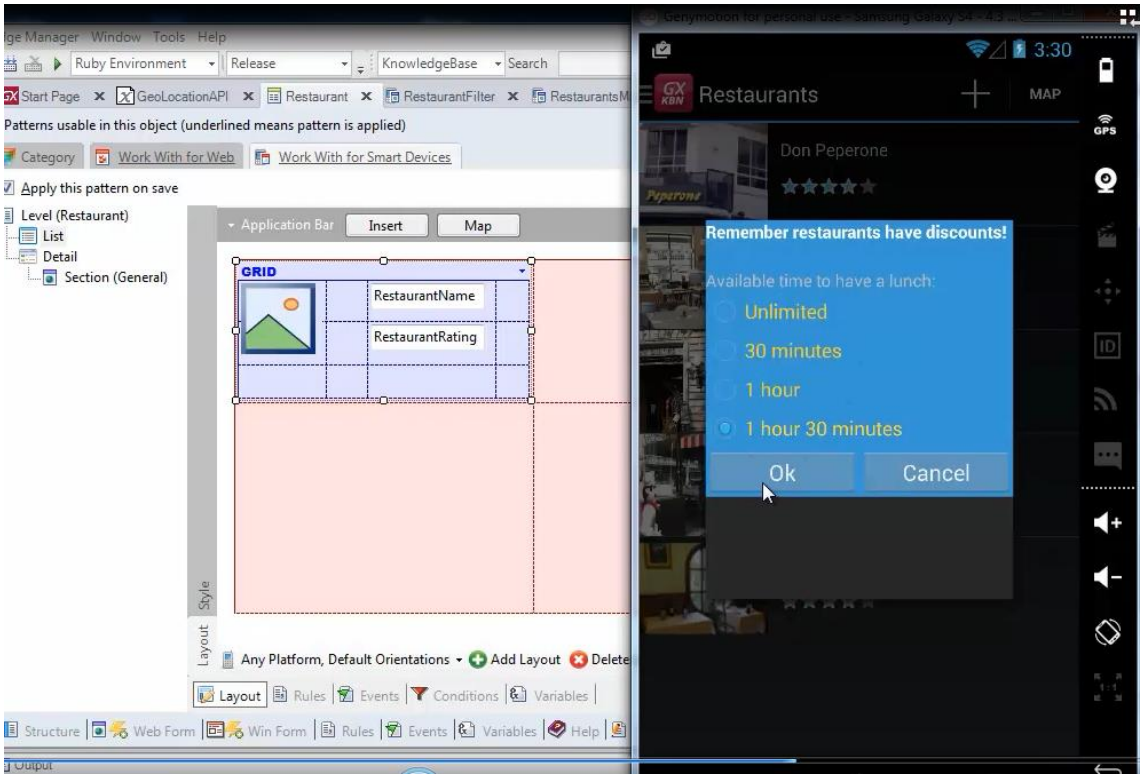


Donde el grid tiene el Control Type default

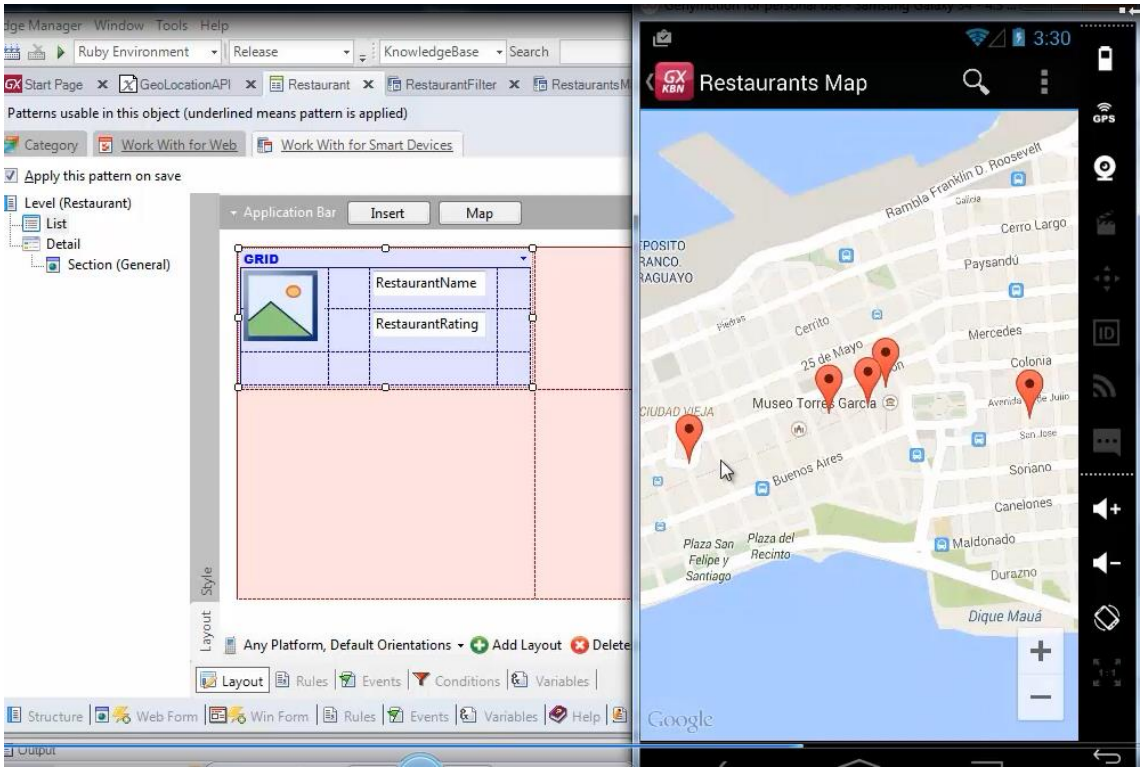
Y por otro lado, ofrecemos al usuario



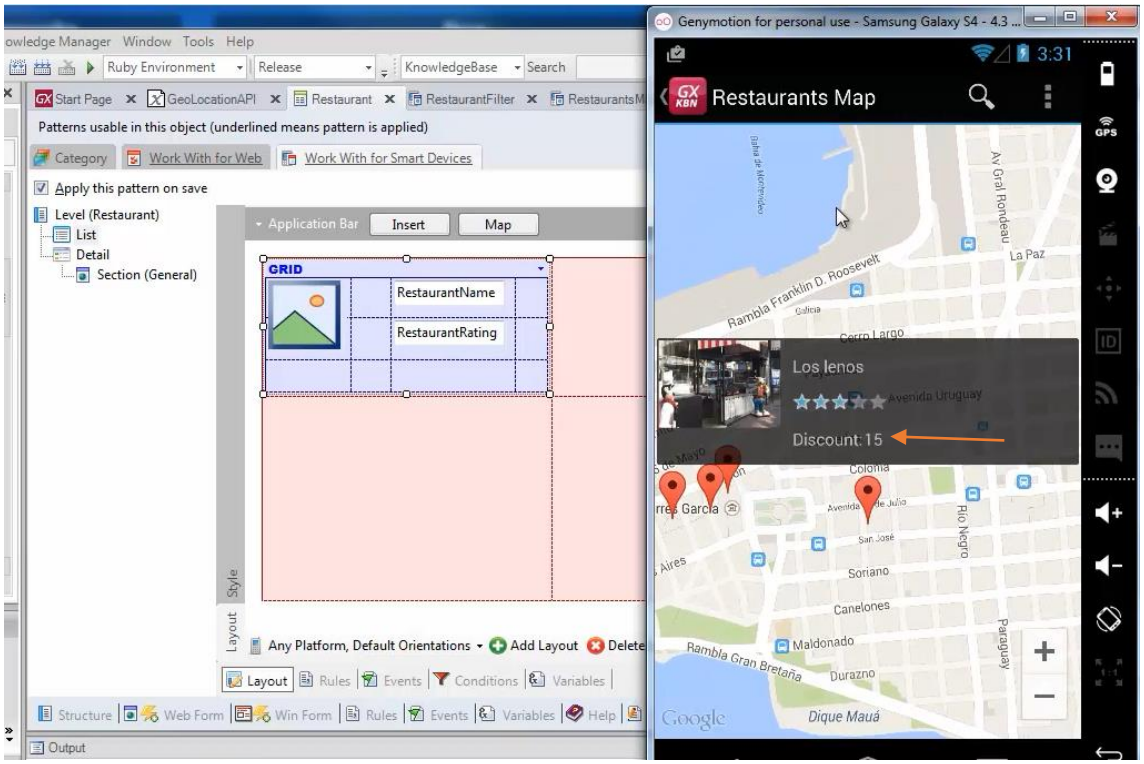
la posibilidad de eligiendo el tiempo del que dispone para almorzar



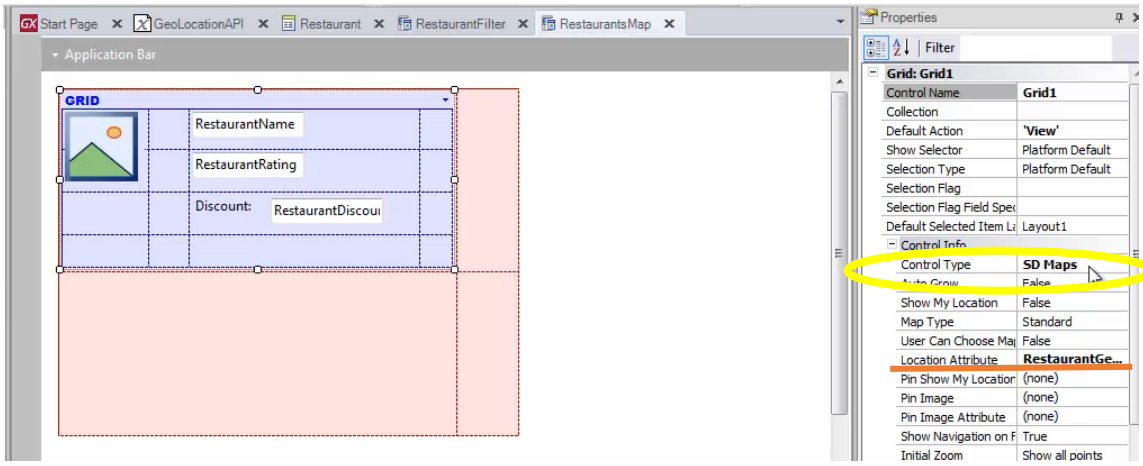
mostrarle los restaurants que se comprometen a brindarle el servicio dentro de ese tiempo



Este es un panel:

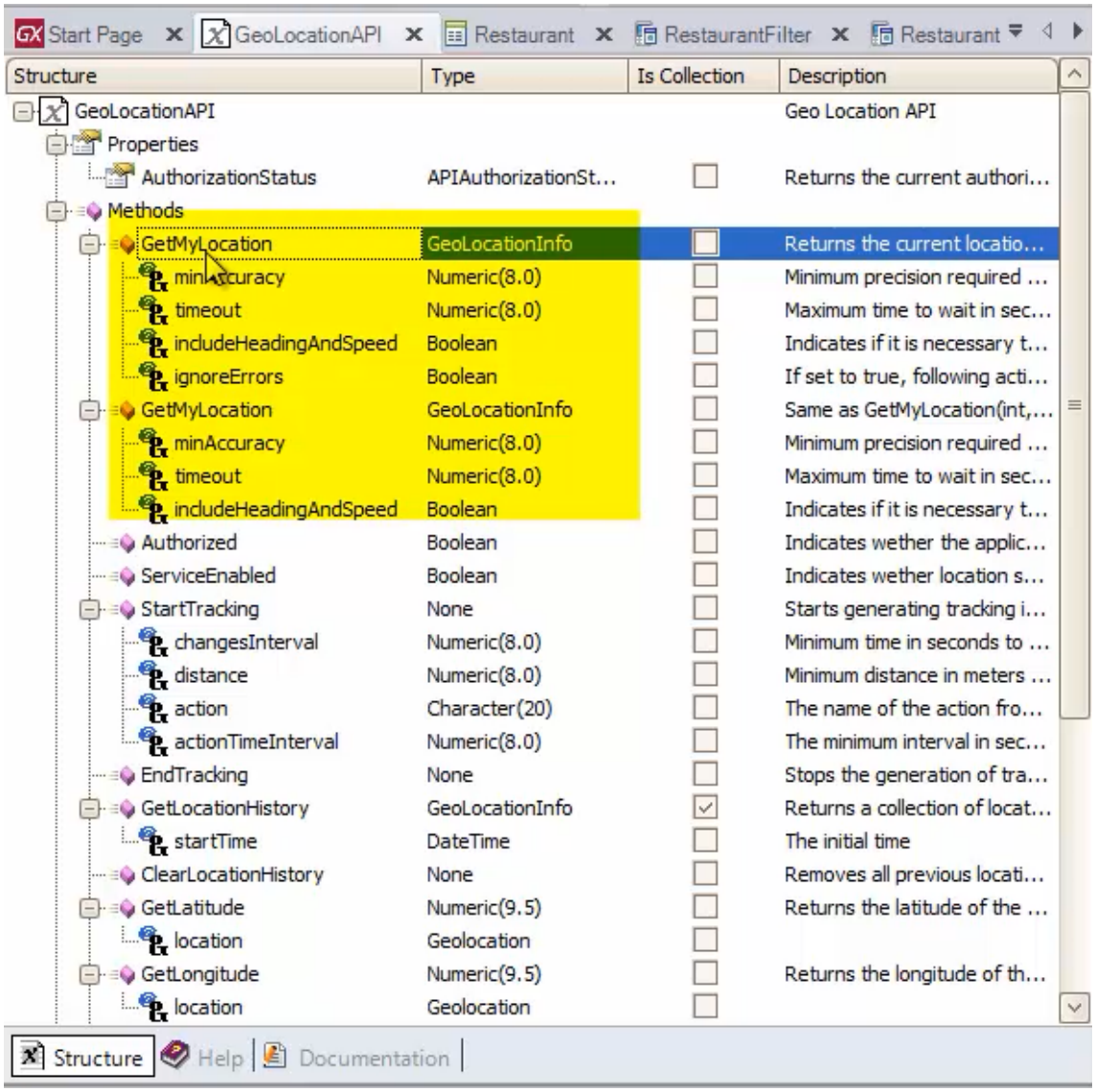


cuyo Layout es muy similar al del List del work with, con la diferencia de este atributo de aquí .. y de la manera de mostrar entonces la información del grid, que será como puntos en un mapa

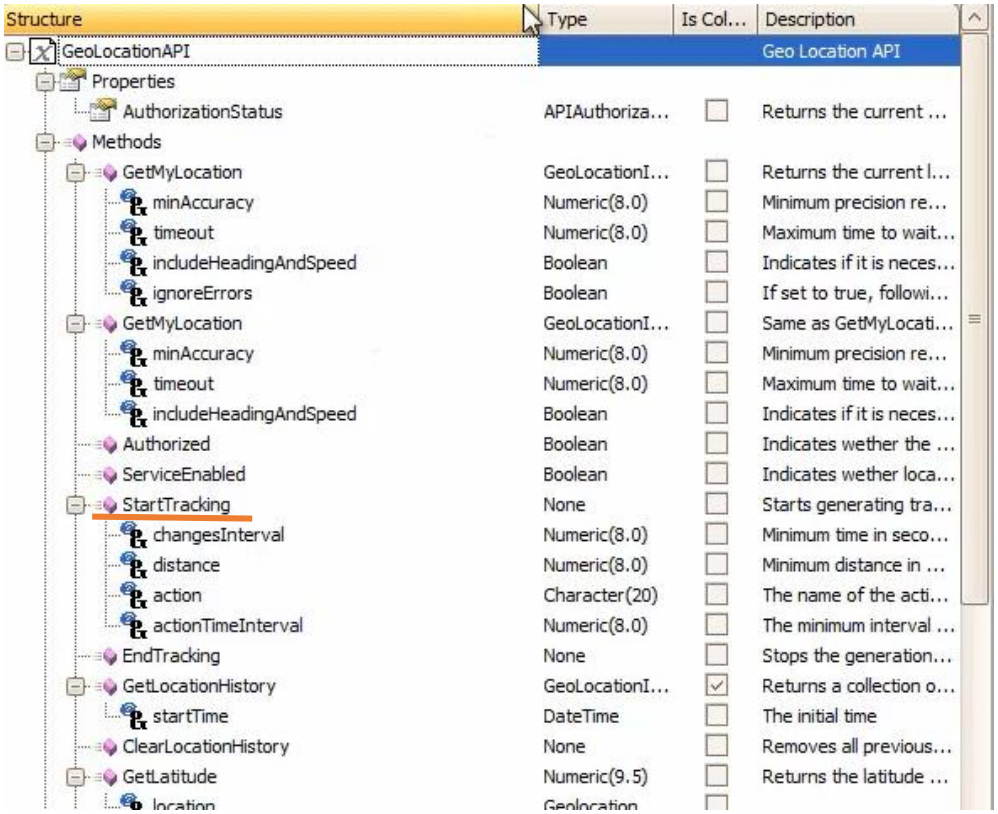


Debido a que tenemos el atributo: RestaurantGeolocation

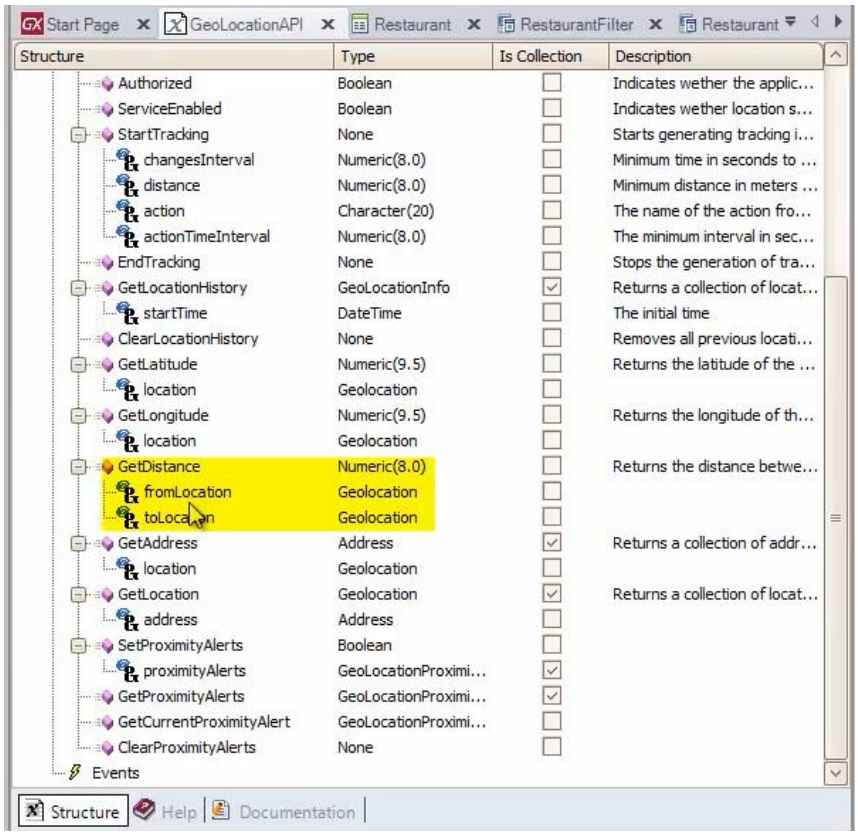
Ahora bien, podríamos solamente querer mostrar al usuario, los restaurants que se encuentran a menos de cierta distancia de la localización actual del dispositivo. Allí es donde interviene la GeoLocationAPI



que ofrece m⍰todos para por ejemplo obtener la Geolocation actual, para lo cual necesita el GPS del dispositivo..

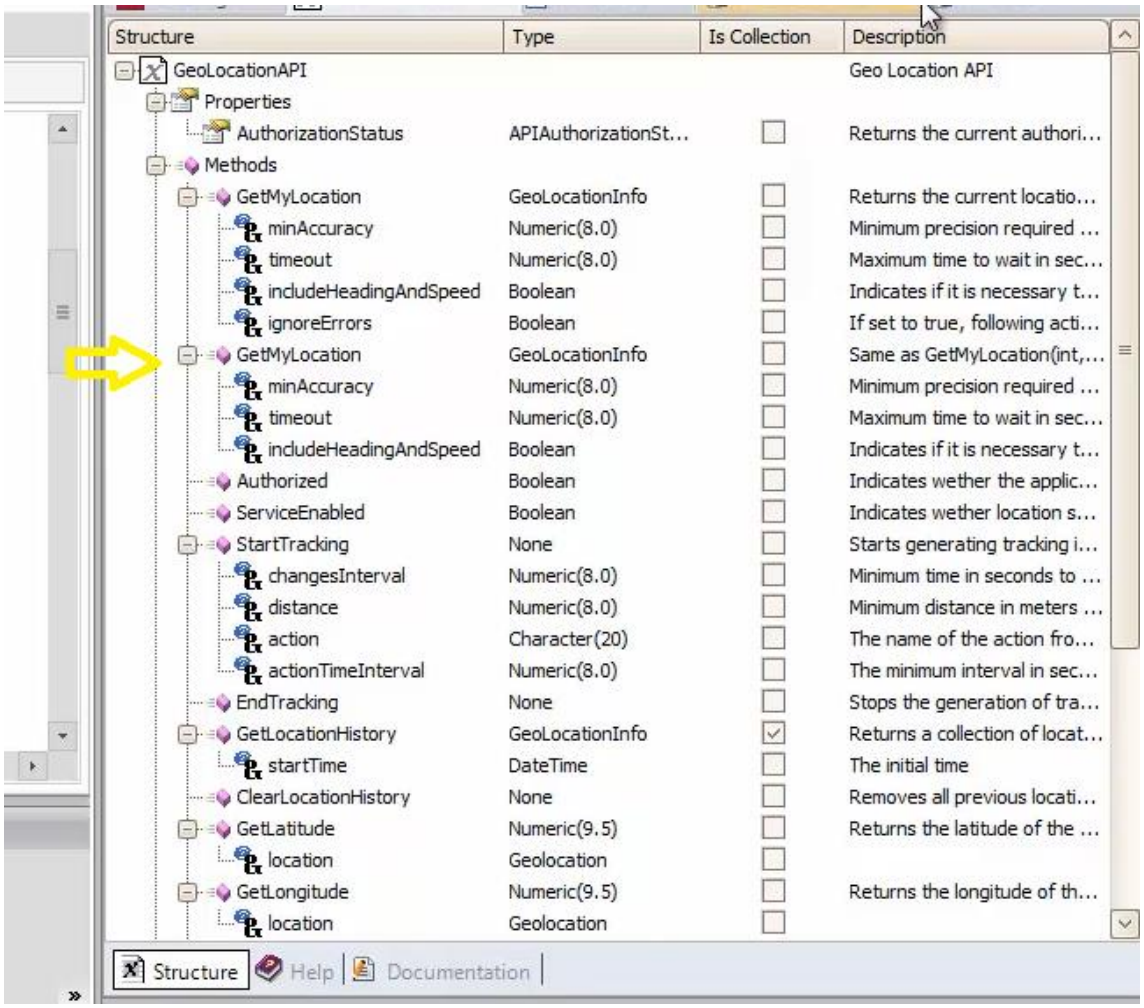


realizar un tracking del camino por el que va pasando el dispositivo a través del tiempo, finalizar el tracking, obtener la latitud a partir de una geolocalización, o la longitud, así como obtener la distancia entre 2 puntos

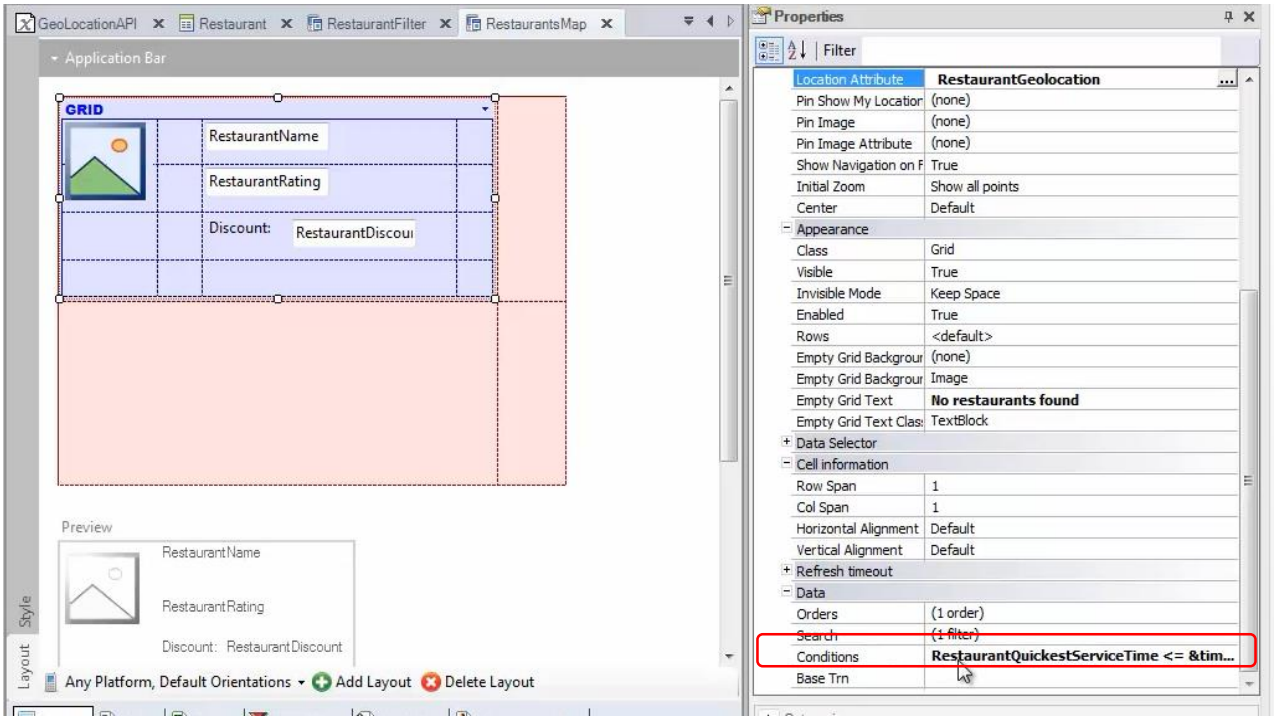


U obtener la dirección a partir de una geolocalización o viceversa.. configurar alertas de proximidad u obtenerlas.. etc.

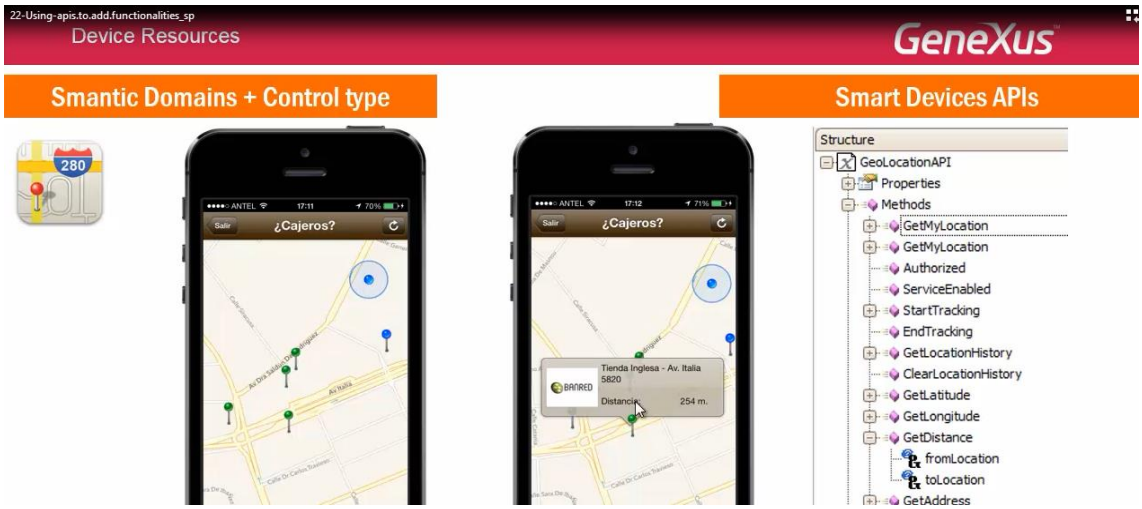
Por tanto, para mostrar solamente los restaurants a menos de –supongamos- 400 metros de nuestro dispositivo, tendremos que utilizar el método GetDistance y el método GetMyLocation



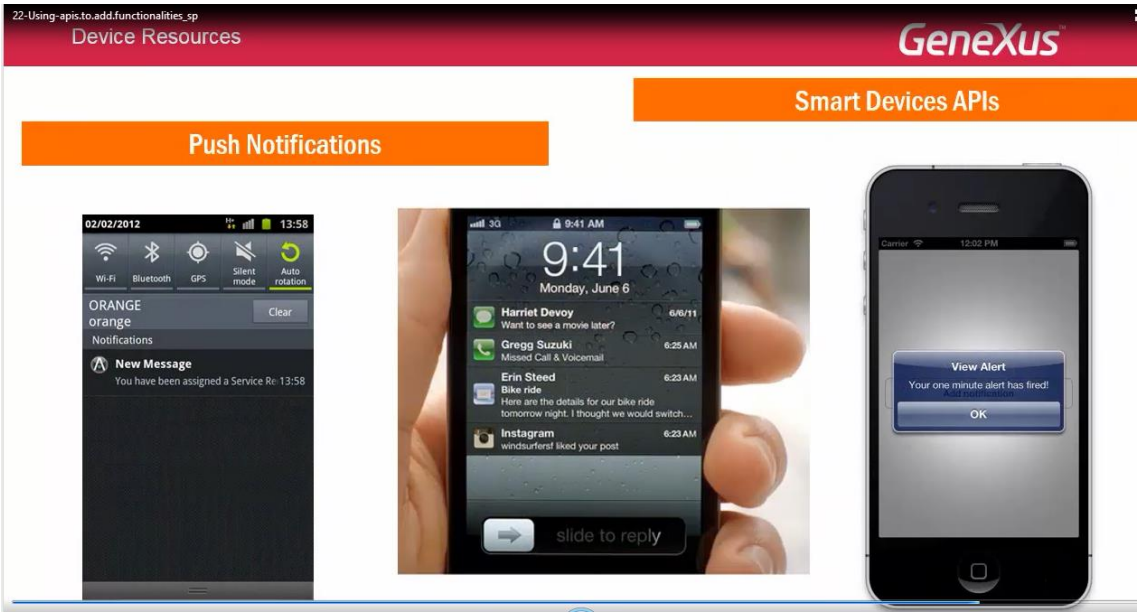
en el grid, a la hora de filtrar la información que mostramos



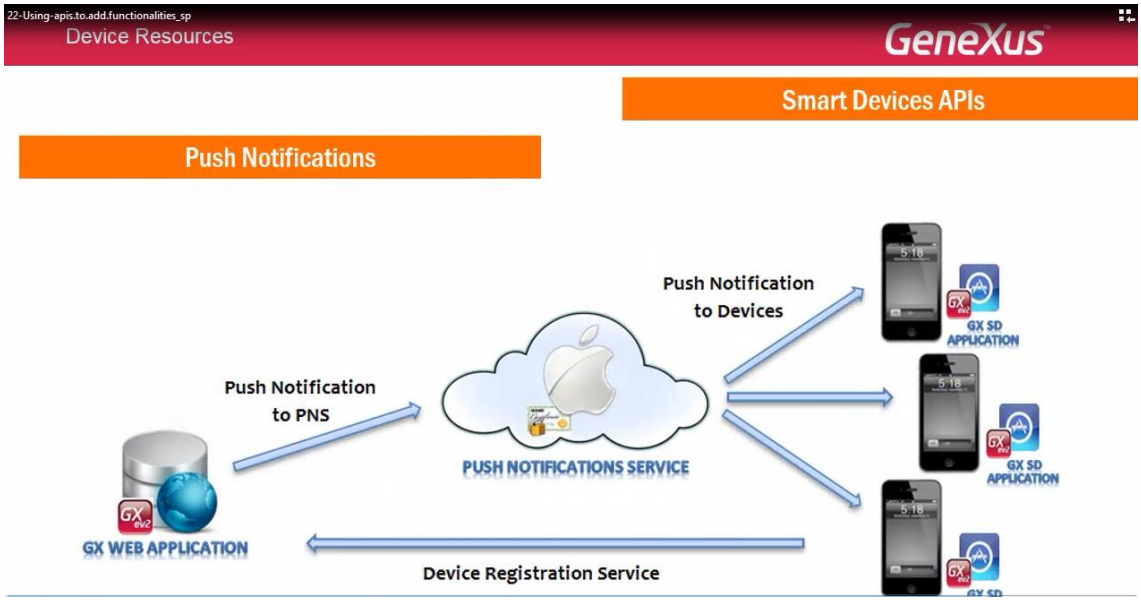
No es ni más ni menos que lo que hace una aplicación para obtener los cajeros automáticos cercanos al punto donde nos encontramos



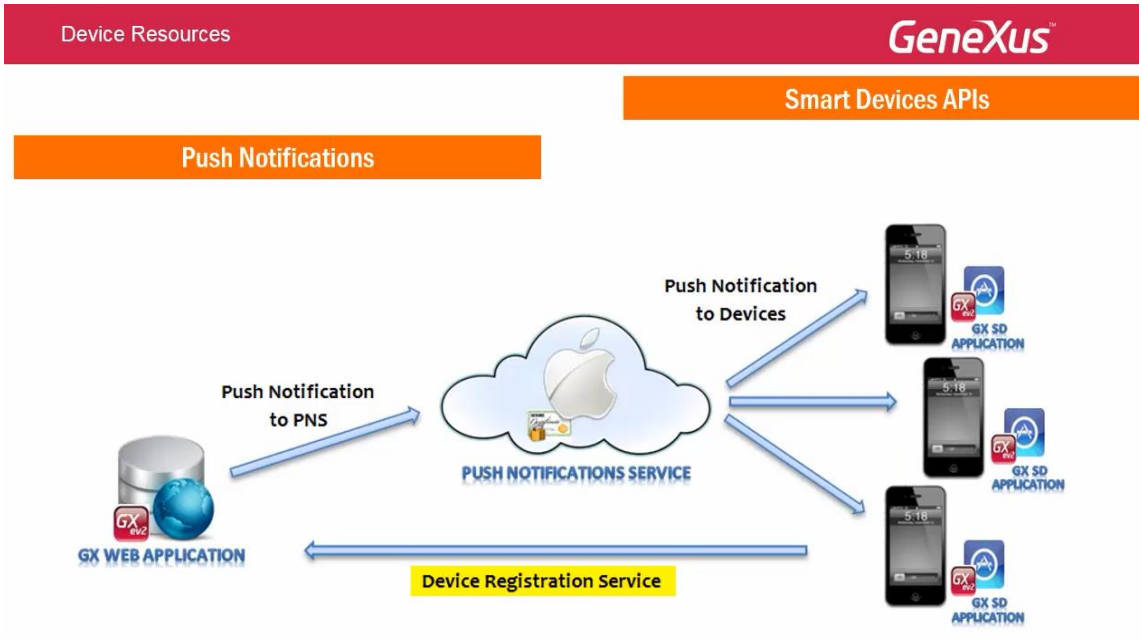
Otra funcionalidad muy utilizada corresponde a las push notifications, notificaciones que pueden enviarse desde el servidor web a los distintos dispositivos que tienen instalados nuestra aplicación móvil, incluso cuando esta no está corriendo



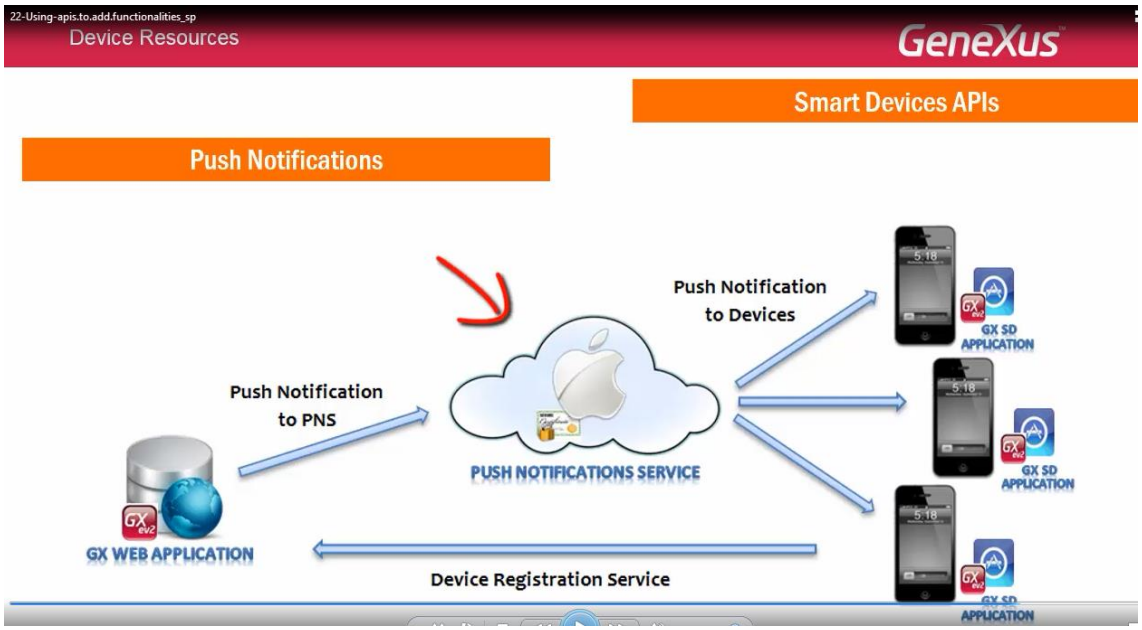
De esta forma, el usuario recibirá la notificación que le envía el servidor y podrá realizar la acción que corresponda.



La idea básica es que cuando el usuario se instala la aplicación en el dispositivo y la ejecuta, este se registra en una tabla en el server

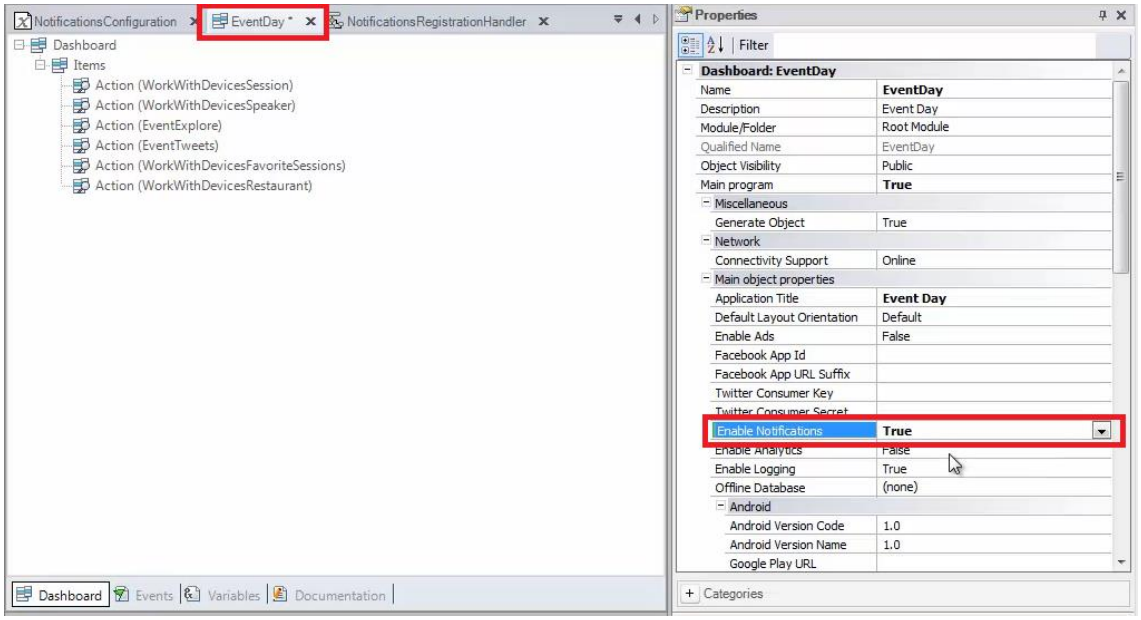


De esta manera, el servidor tiene identificados a los devices para poder mandarles notificaciones. Pero quien envía efectivamente las notificaciones, es un proveedor que depende de cada tecnología

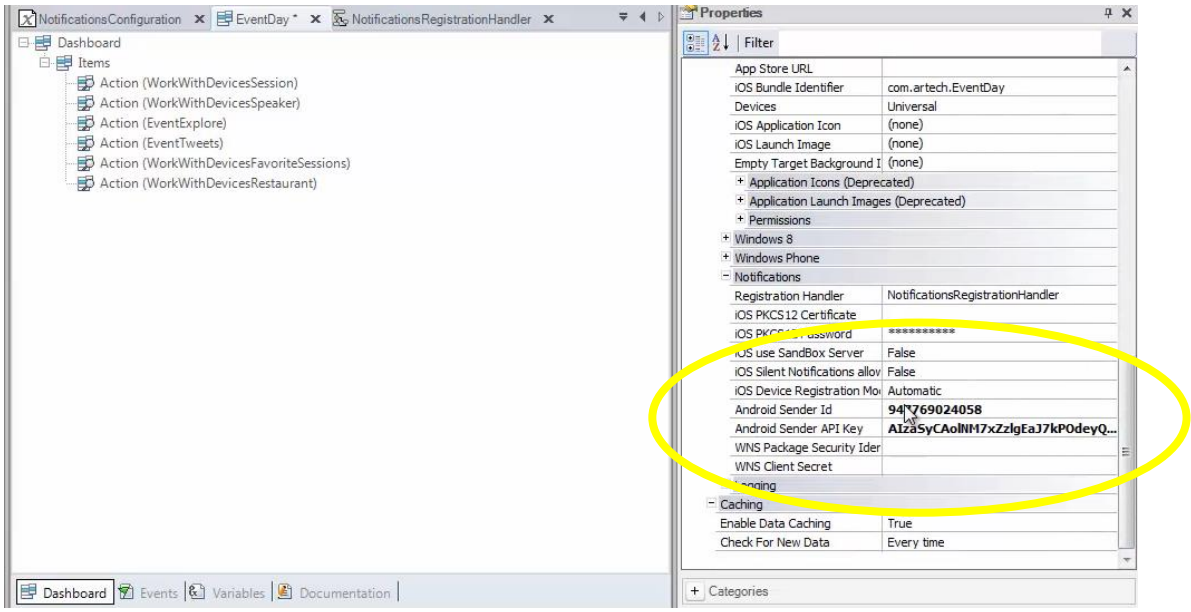


Así el de Apple, se llama Apple Push Notifications Service.. el de Google, Google Cloud Messaging.. el de Blackberry, Blackberry Pushing Service.. y el de Windows, Windows Push Notifications Services.

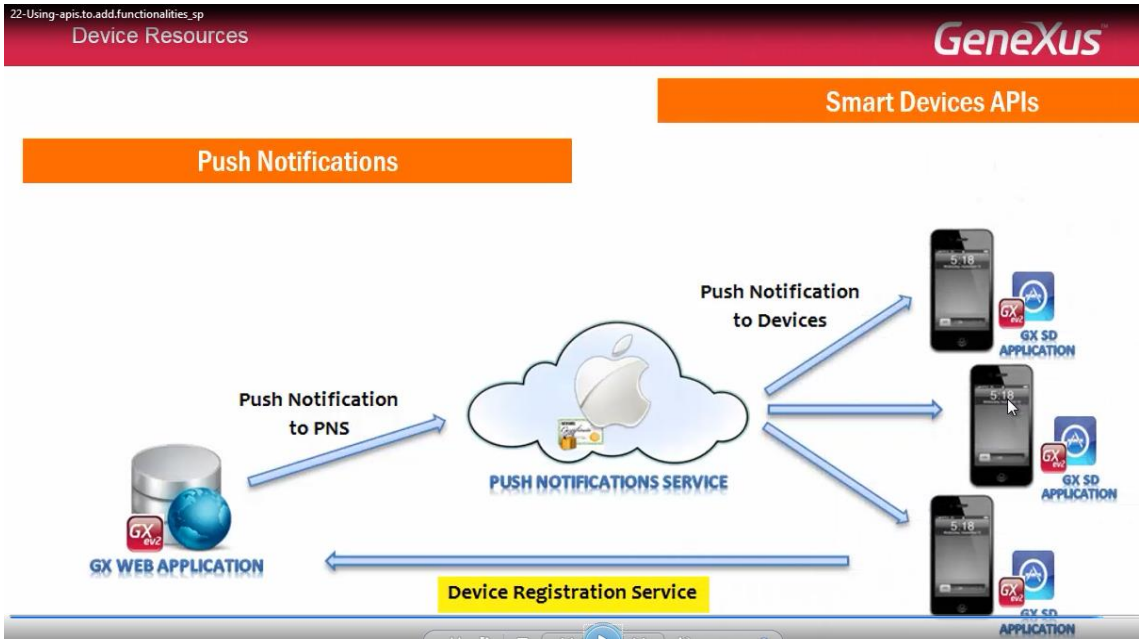
Lo que debe hacerse es ir a cada uno de estos servicios a crear un proyecto y obtener las credenciales, para luego ir al objeto main de nuestra aplicación móvil y prender –habilitar- las notificaciones



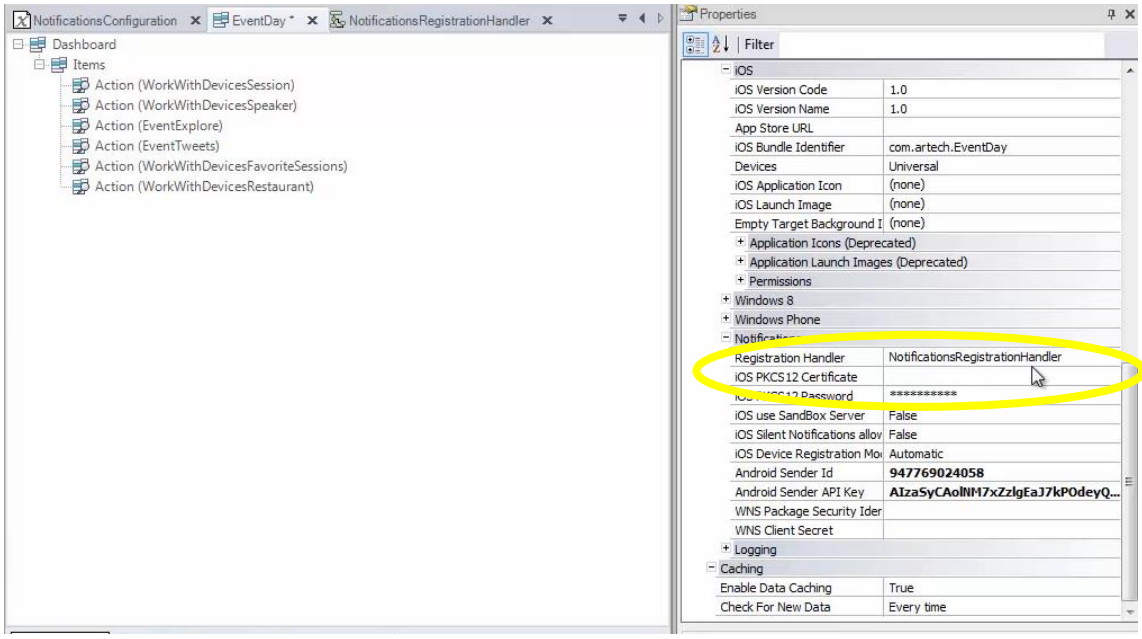
ingresando en las propiedades correspondientes, las credenciales obtenidas



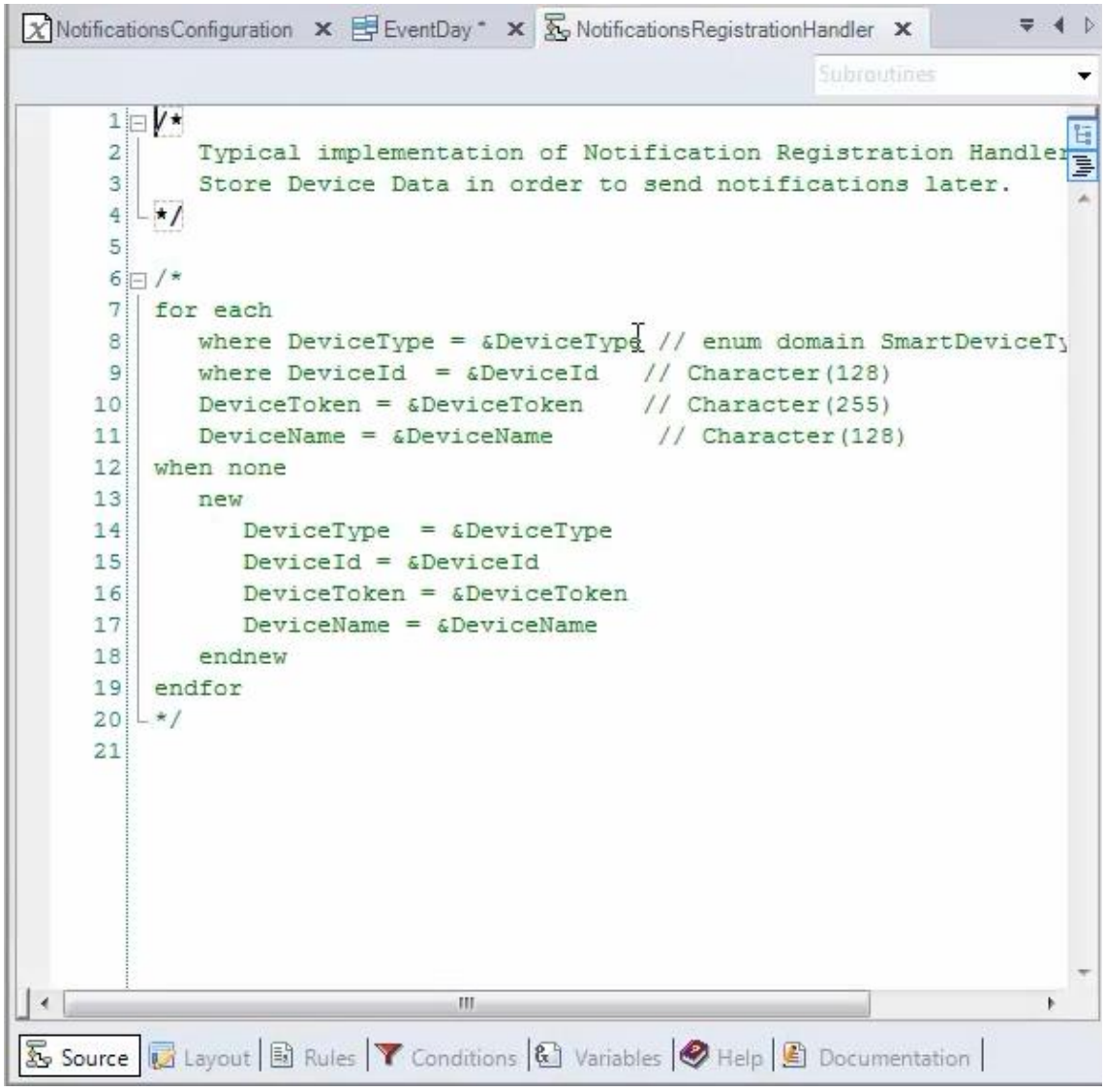
Debemos luego implementar la registración de los dispositivos



Para ellos ya se cuenta con el procedimiento predefinido de este nombre

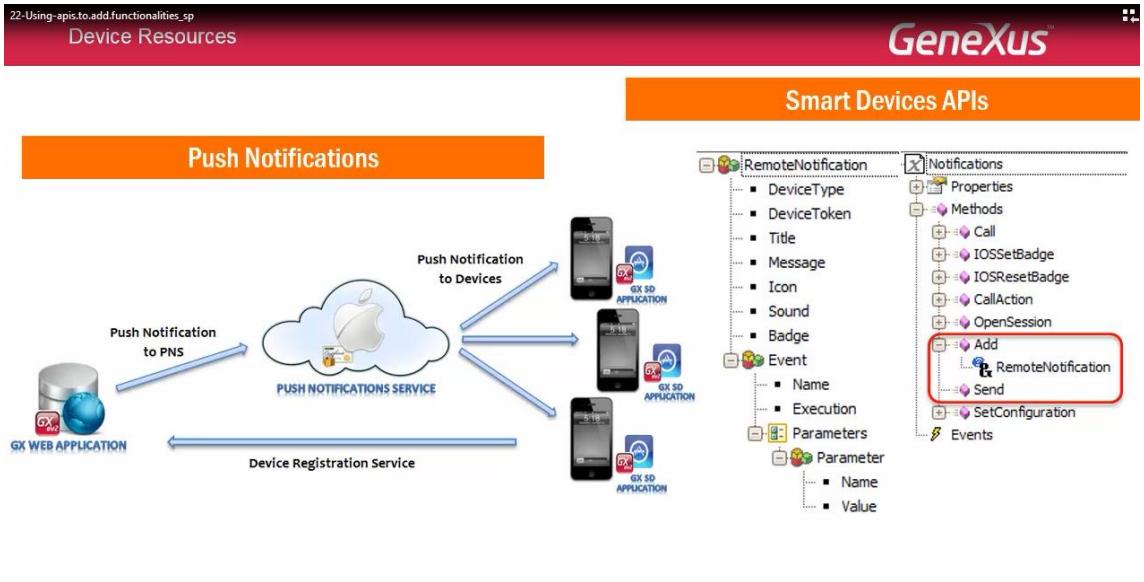


Podemos crear uno nuevo y colocarlo aquí… o utilizar el predefinido

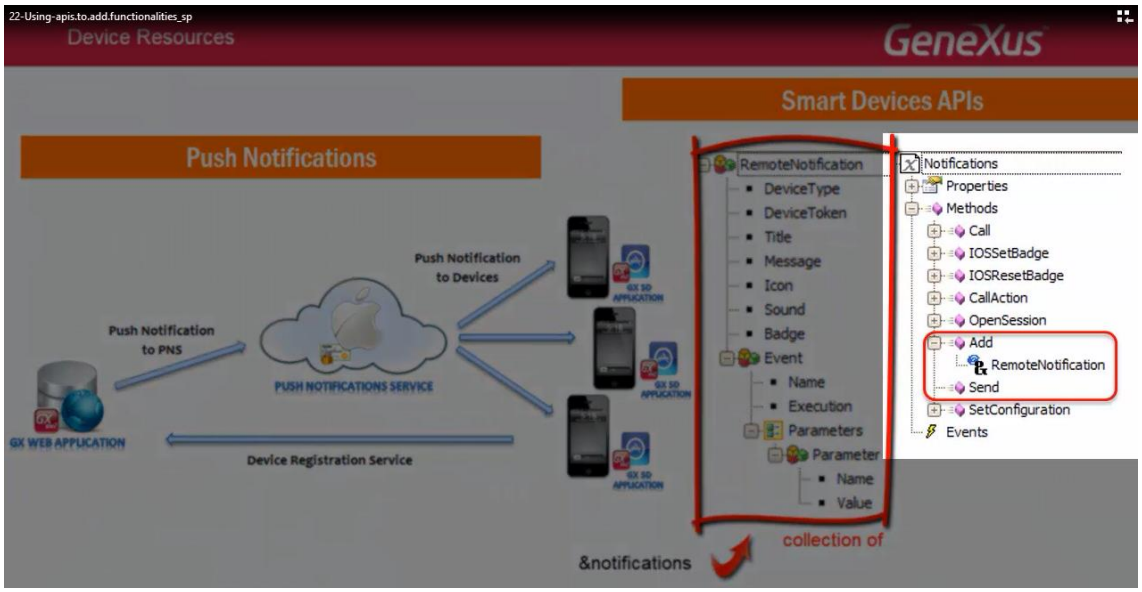


creando una transacción que tenga estos atributos, para registrar entonces los dispositivos en el sistema.

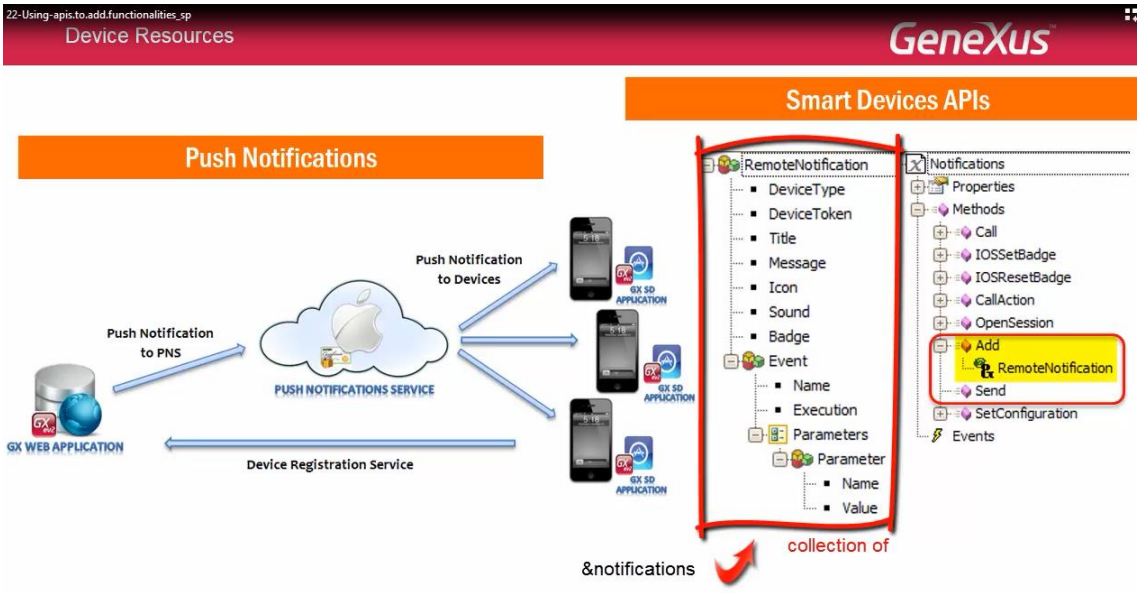
Este proc será invocado cada vez que se ejecute la aplicación en el dispositivo.



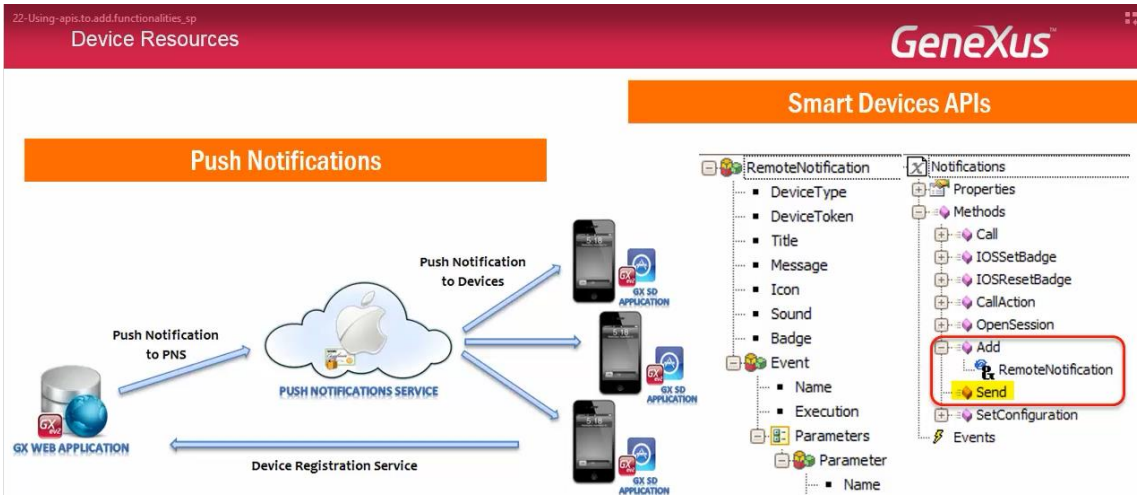
Luego desde el objeto que necesitemos que inicie las push notifications, nos creamos una variable de tipo colección del sdt que viene con la api notification



(sdt de nombre: RemoteNotification) y la vamos cargando a través del método Add de la api:

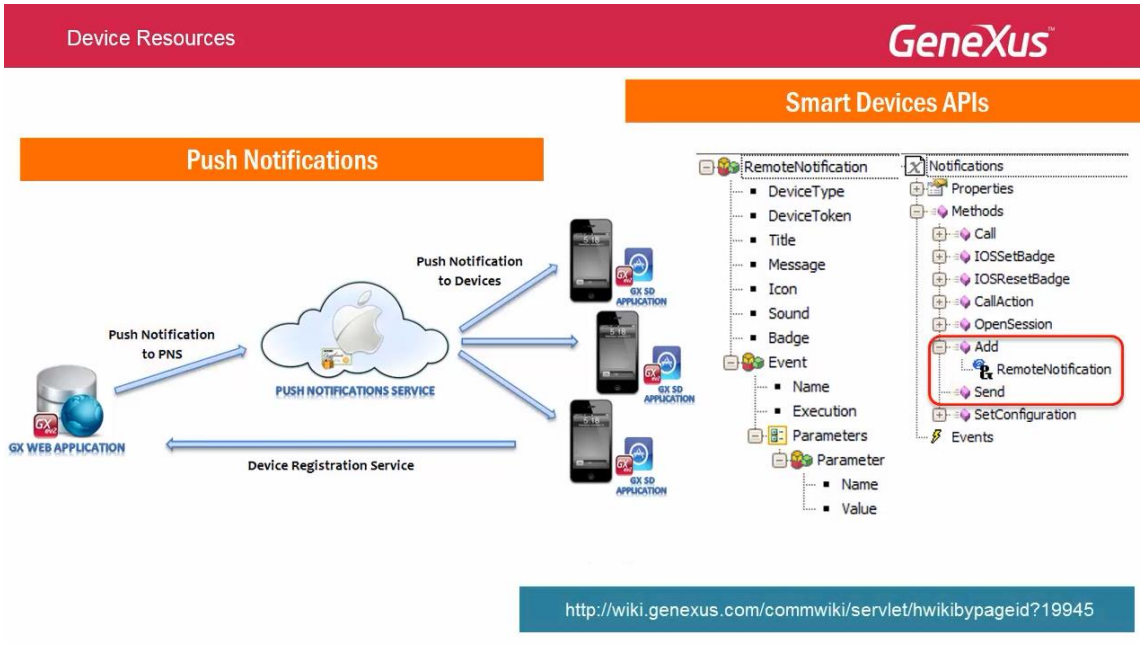


Con las notificaciones a ser enviadas a cada dispositivo registrado. Luego de cargada esta variable colección, enviamos las notificaciones haciendo uso del método Send

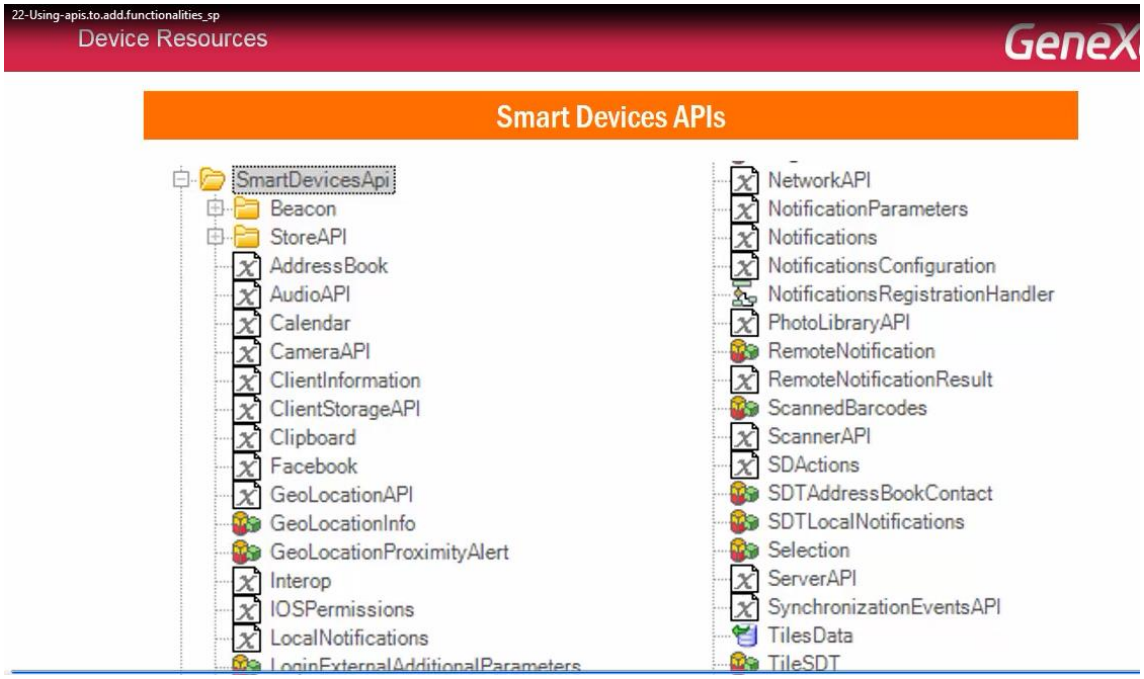


Este método en capsulará la invocación a los servicios de notificaciones de cada tecnología.

Vea más sobre push notifications en nuestro wiki:



Hemos presentado en este video



Sólo algunas de las muchas apis que ofrece geneXus, y que van aumentando día a día.

Para obtener información de las demás api's o acceder a detalles de las apis aquí vistas, lo invitamos a dirigirse a nuestro wiki.

Video filmado con GeneXus X Evolution 3