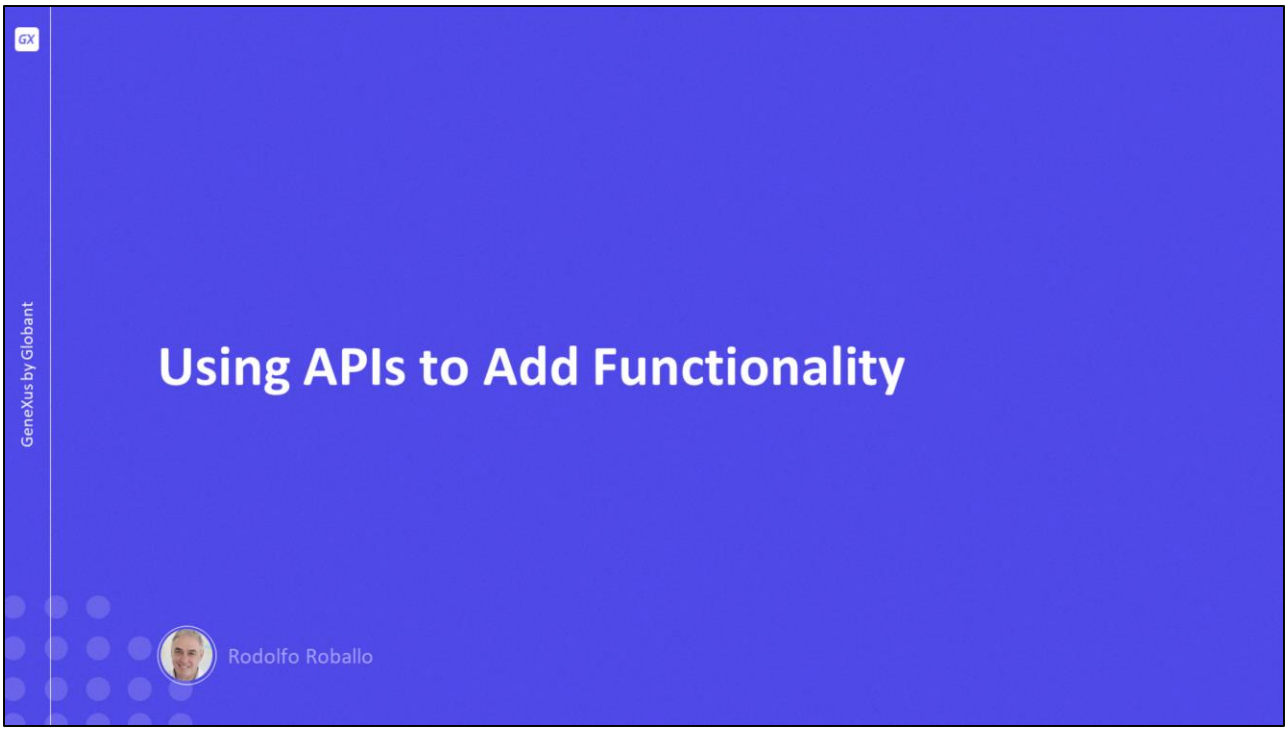
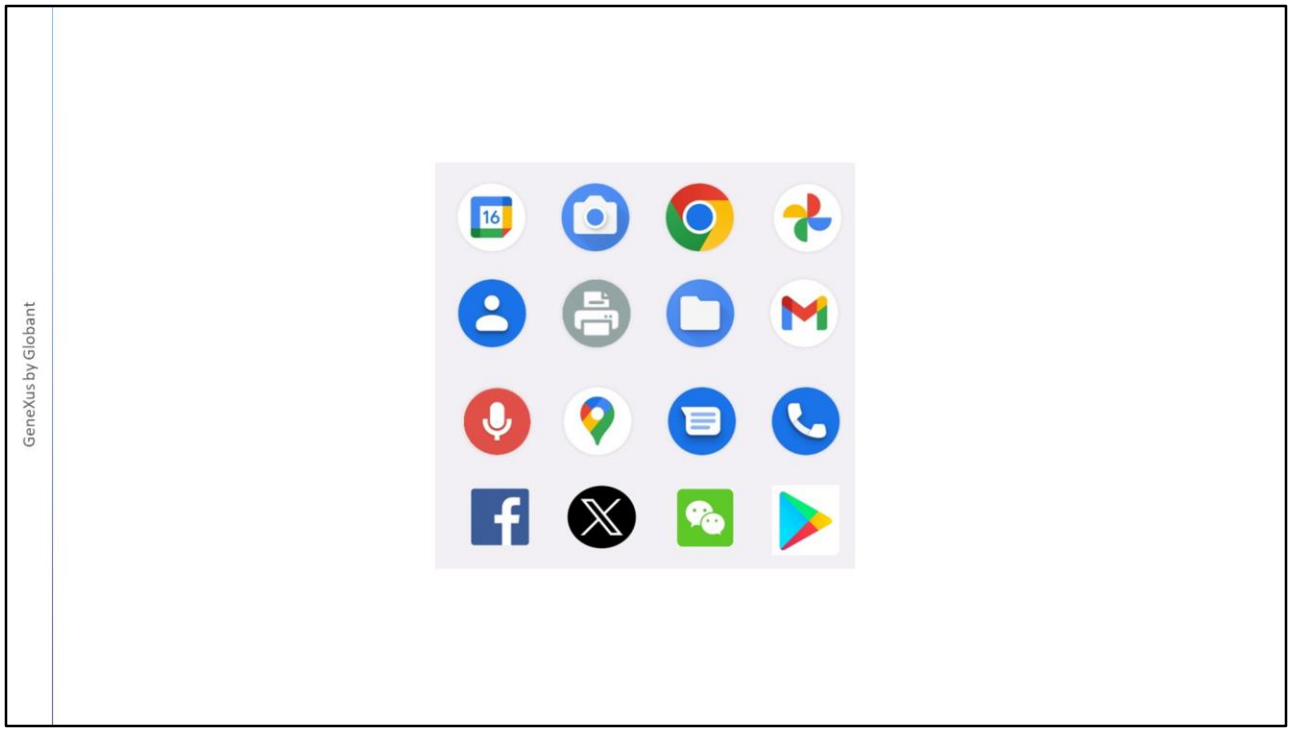


In this video, we will see how to build an application with functionalities that allow for integration with the device's resources, both physical and logical, in addition to communicating with external applications.

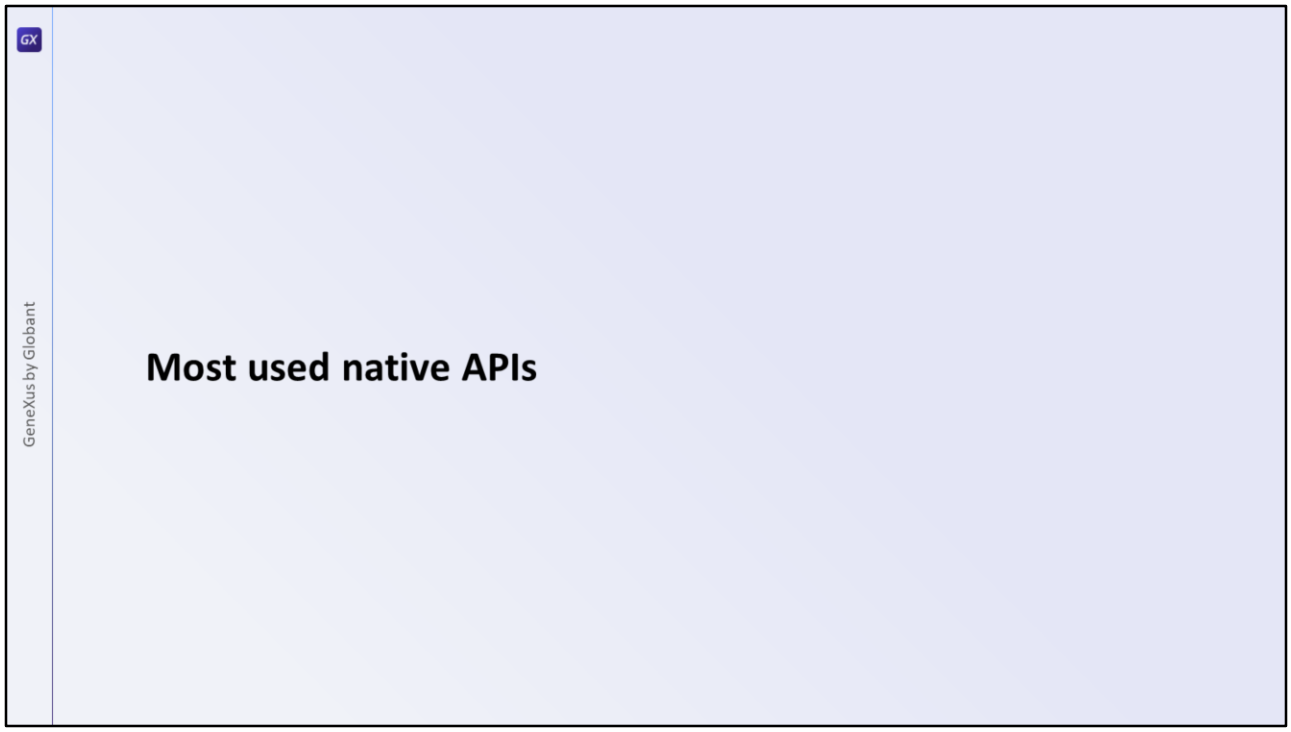


Using external objects predefined in GeneXus, we can programmatically access many of the functions of a physical device, such as using the camera, accessing the audio recorder, making a call, or printing certain content.

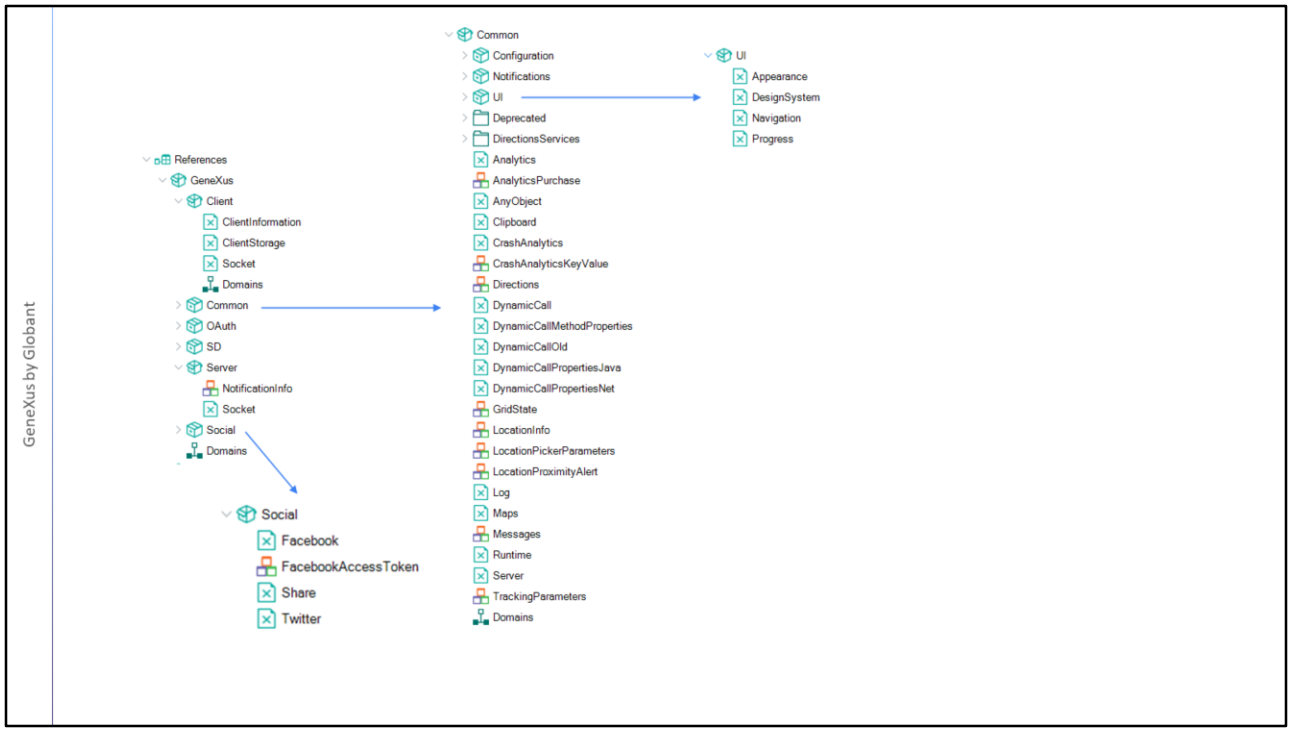
We can also access applications that are already installed on the device, such as the calendar and contacts, as well as access files, send an SMS, send an email, open the browser, or access the store.

We can also interact with social media such as Facebook or Twitter.

Next we will see how to access some of these functionalities.



Some of the APIs available in GeneXus can be used only on native mobile devices, while others are specially designed for Web applications, and others can be used on both platforms.



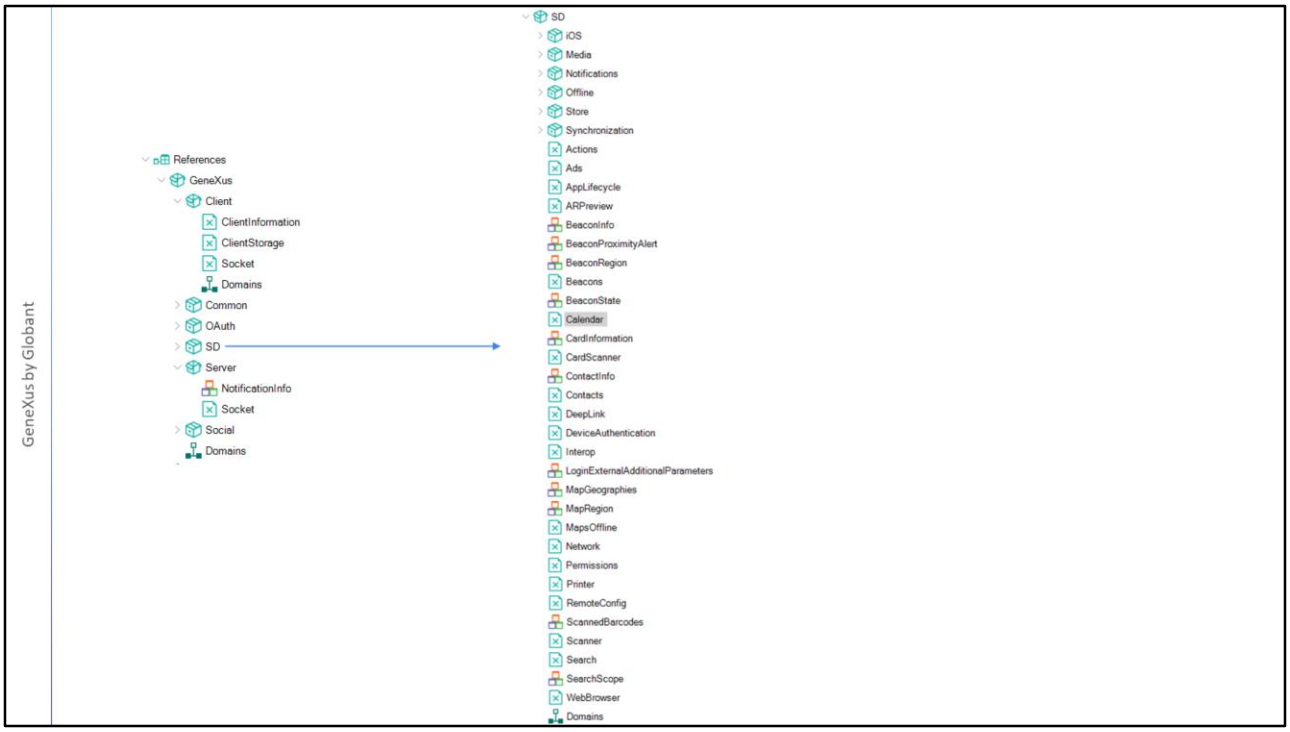
These APIs can be found in the KB Explorer below the References node, in the GeneXus module, which also includes all the SDTs, domains and Procedures required by these APIs. This module and its sub-modules are created when the KB is created, and all its objects are already compiled, which is why they are read-only. This means that they don't have to be generated every time the application is built, and this greatly improves compilation times.

On the other hand, anyone can develop a module with external objects, domains, SDTs, etc. to implement both native and web functionality and distribute it. Moreover, we will be able to import those modules and use them in the same way we use the predefined APIs. This makes collaborative work easier for the developer community.

At first sight, we can see some special APIs for SD, APIs for the Web, and others that can be used for both platforms.

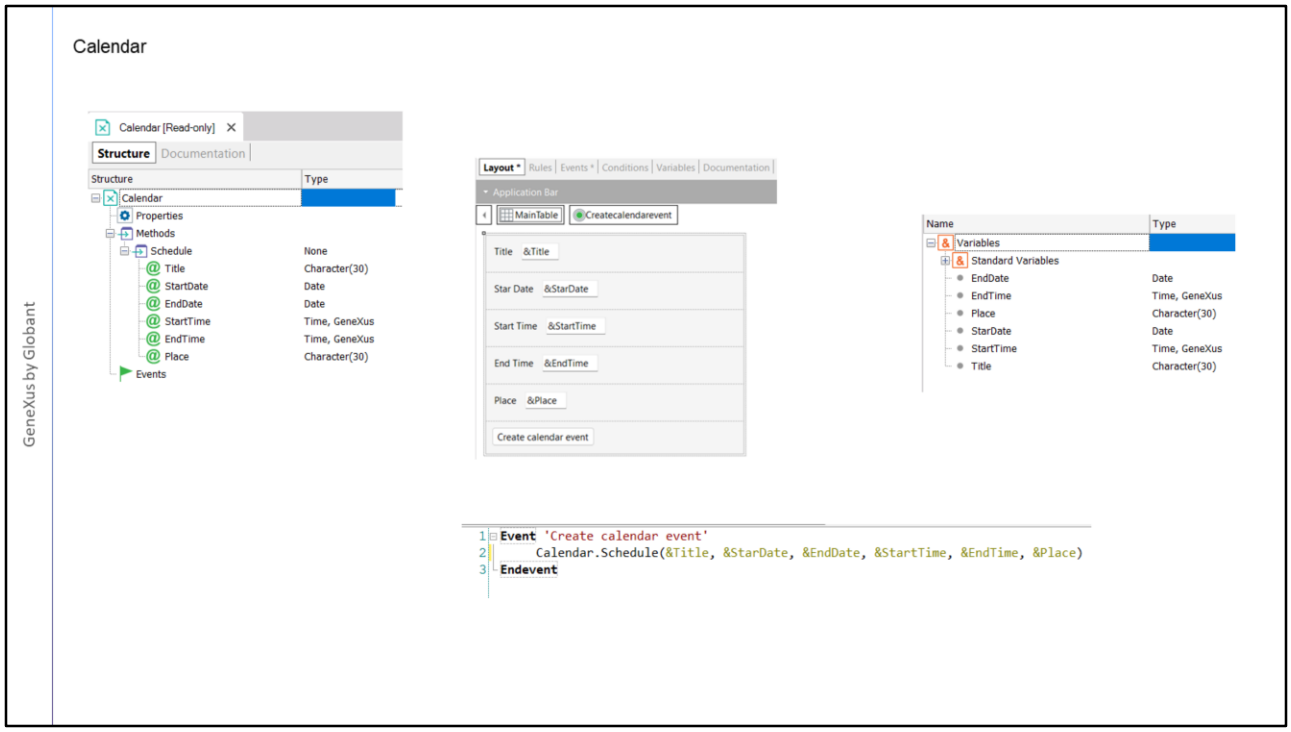
For example, in the Common submodule we see the Maps API, the Log API, or the Analytics API that can be used in both web and native applications; in the UI submodule, we find the Navigation external object to show or hide regions of the screen, or the DesignSystem external object that allows our application to configure design aspects using code.

In the Social submodule, there are APIs to interoperate with Facebook or Twitter, as well as to share information with one of the programs installed on the device.



In the SD group, we find all the APIs that we can use on a native mobile device.

Let's see how to add an appointment to the calendar.



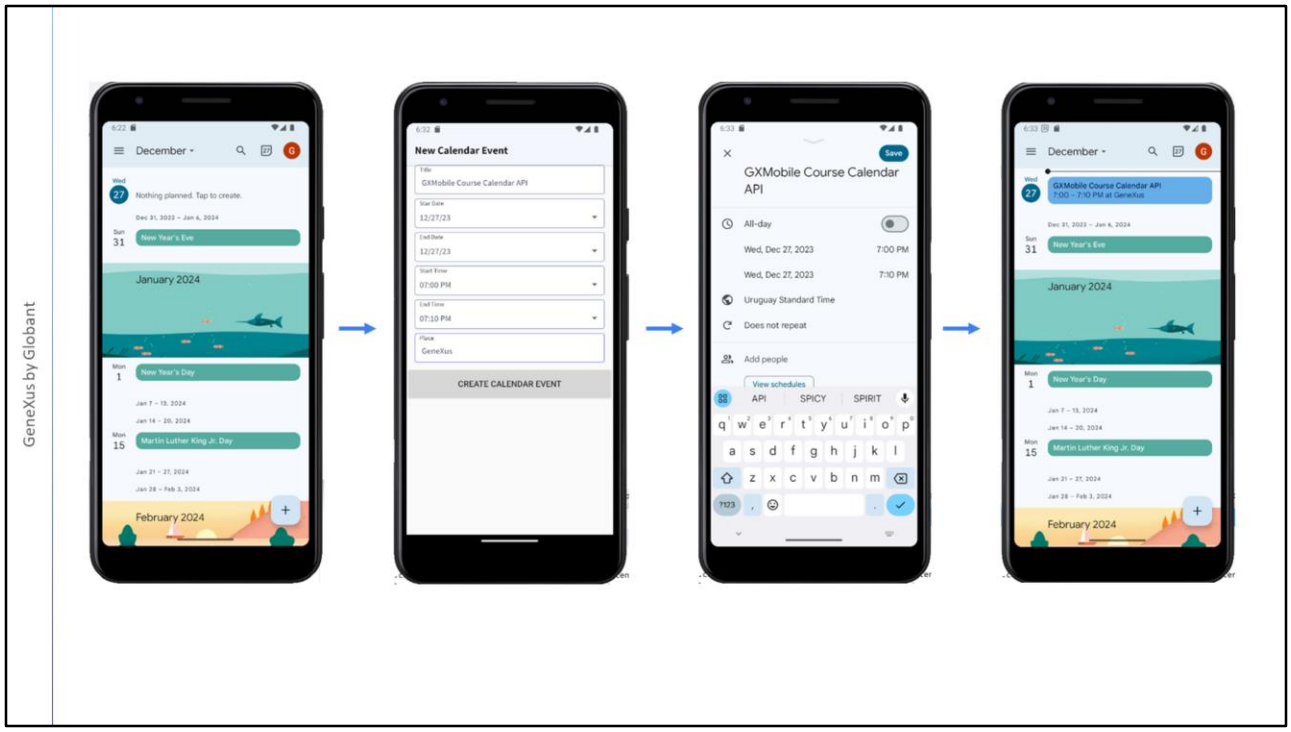
If we open the Calendar API, we see that it has no properties and has only one method, the Schedule method.

In its parameters, we recognize the information needed to add an appointment to the calendar, such as the title, start date and time, end date and time, and location.

Something important is that we have to pass all the parameters in the order in which they appear in the method; if there is a parameter whose data we don't have, we leave the place for that parameter empty but we write it in the invocation itself.

The panel named NewCalendarEvent shows on screen the variables needed for the data required by the Calendar API method and a button that we will use to call the Schedule method, passing it the required parameters.

We will set this panel as main, and since we have already opened the emulator, we run the panel.



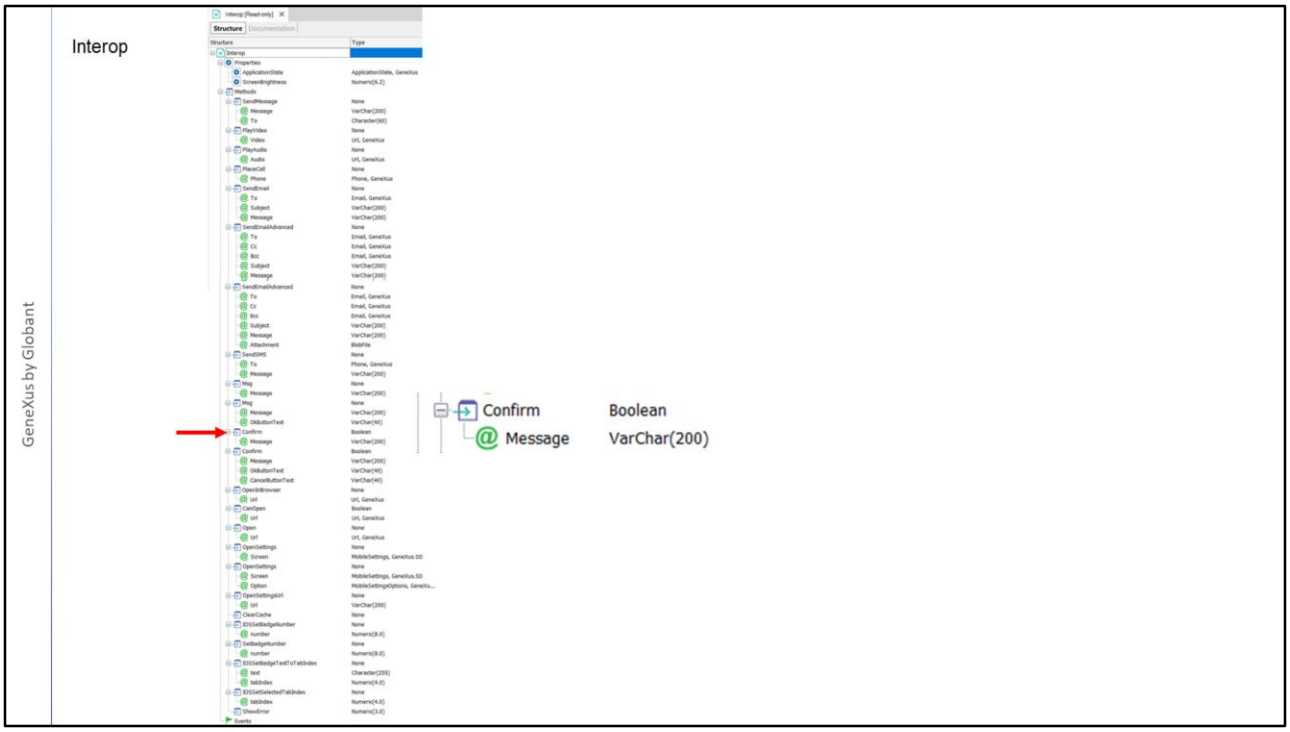
Our panel is running. Before entering data, we go to the Calendar application, log in, and see that no event has been added for today.

We enter an event called GXMobile Course API Calendar for today, within an hour, lasting 10 minutes; we enter GeneXus in location and click on the CREATE CALENDAR EVENT button.

A window was opened on the device with the data we entered, and everything is correct, so we click on Save. Now we return to the calendar and see that the event we created through code from our application using the Calendar External Object is displayed.

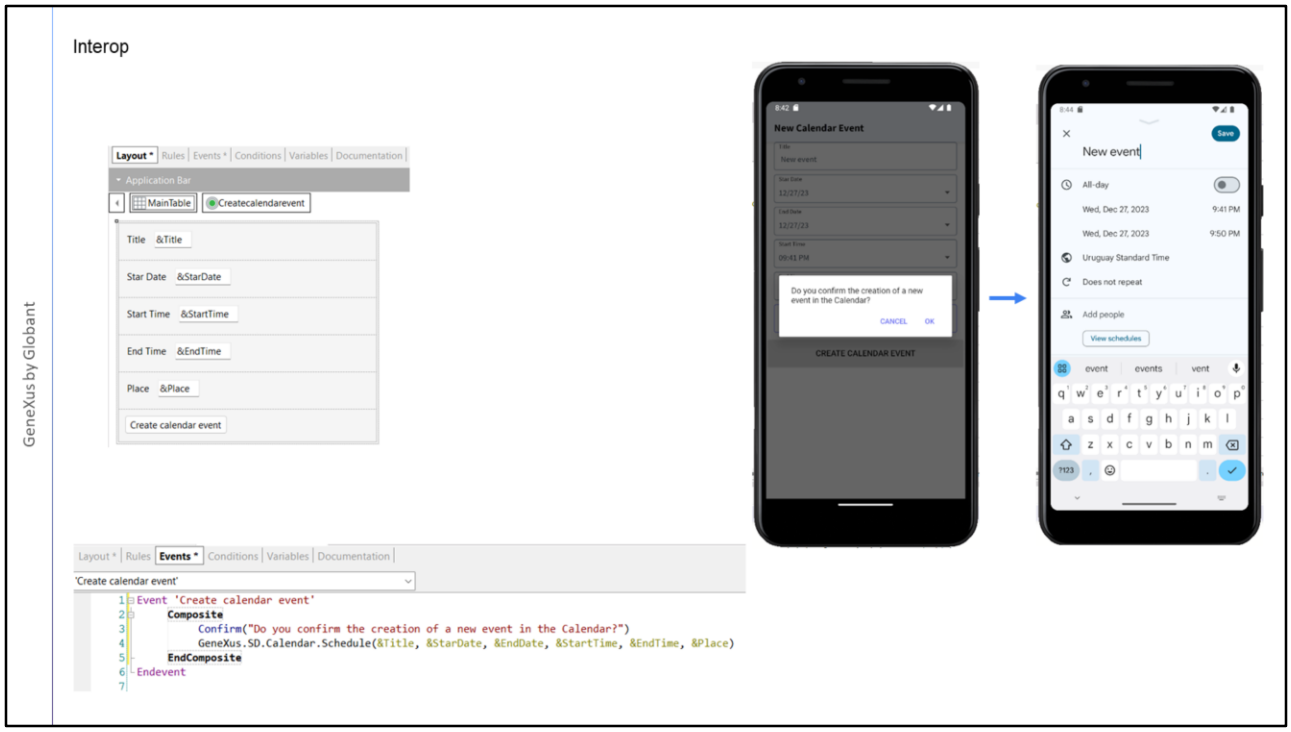
GeneXus has some external objects that provide common actions, such as going straight to the main screen, returning to the previous screen, saving content, and others that provide the most common functions of a mobile device such as sending messages, confirmations, emails, or SMS messages.

Let's look at a couple of these multifunctional APIs: the Interop API and the Actions API.



Interop is a frequently used API that offers interoperability with messaging apps, and allows sending messages by email and SMS, as well as showing messages to the user to notify them about something or to ask for confirmation of an action, run videos or audios and many other functions as you can see in the object's methods.

Let's use the Confirm method to add a confirmation instance before adding the calendar appointment we saw in the previous example.



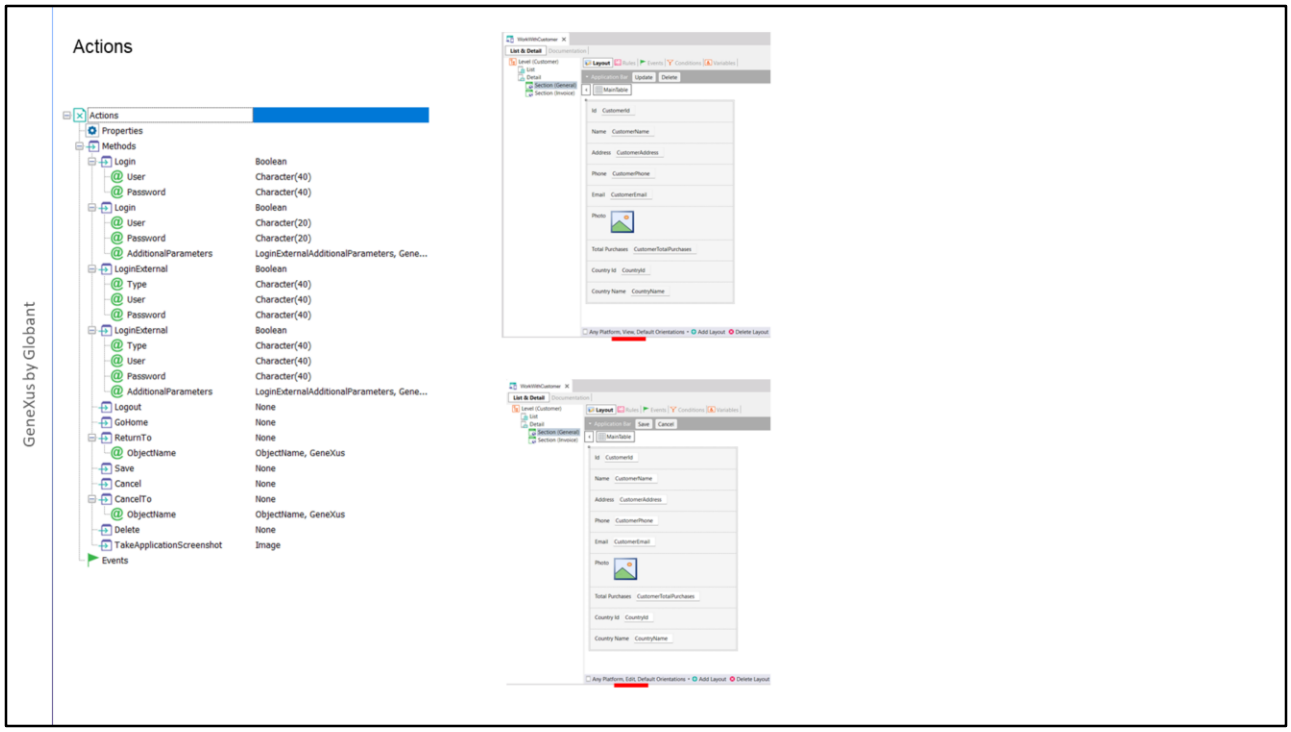
To do so, we will invoke the Interop.Confirm method by passing it as a parameter the confirmation message we want to be displayed. Given how often the Confirm method is going to be used, we don't need to write Interop.Confirm; we can just write Confirm and GeneXus will understand it.

Let's go again to the NewCalendarEvent panel and in the button event we can drag interop, period, and choose the Confirm method or, as we said, write Confirm and in brackets the text we want to display to the user. Note that although the Confirm method returns a Boolean value, it is not necessary to evaluate the returned value because GeneXus has the intelligence to run the next line of code depending on whether we accept or cancel the action.
Remember that the Composite block is optional, but we added it because it provides automatic error handling.

We run it.

When we click on the button, a pop-up window opens with the confirmation text we typed, and we can accept or cancel it.

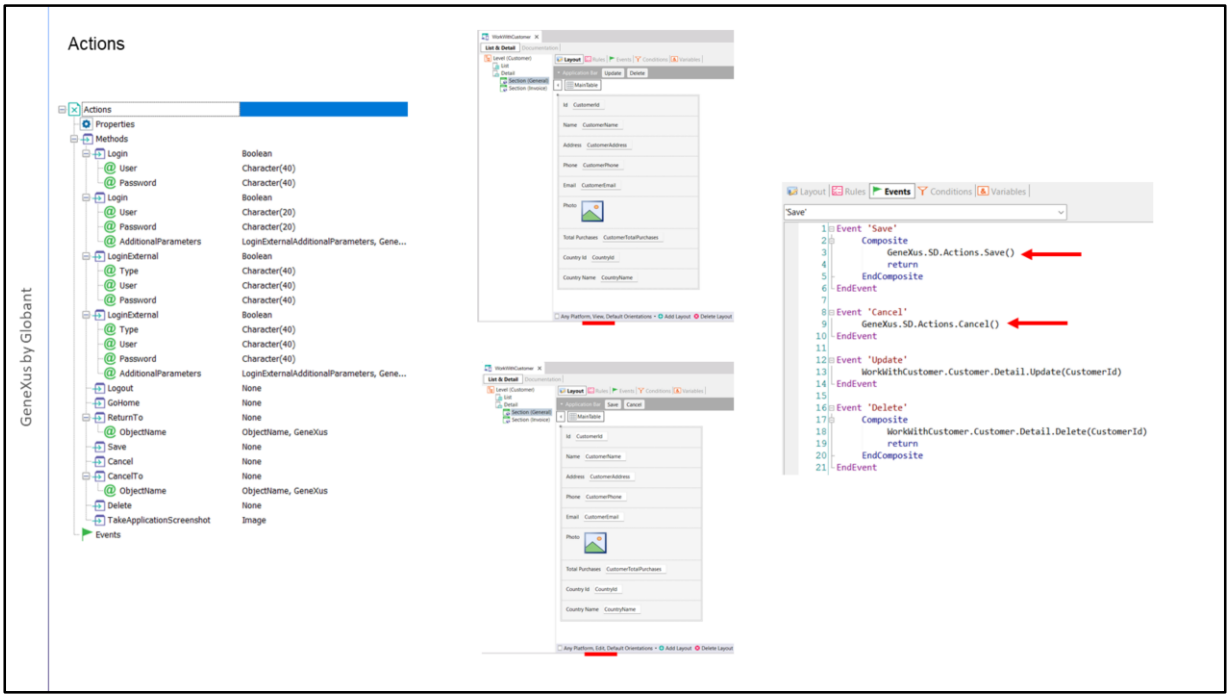
Clicking on CANCEL takes us back to the panel data entry screen and, if we click on OK, the screen for creating the new event opens.



There is an API that had already been mentioned before but we hadn't looked into how it works: the Actions API allows, among other things, to abstract the functionality of Login, Save, Cancel. We had already seen the latter methods in the events automatically implemented by the Work with pattern at the Detail level.

Here we see the WorkWith object that is created by applying the WorkWith pattern to the Customer transaction.

For example, let's look at the Detail, Section (General) node. We have the Update and Delete buttons for the View mode layout, and for the Edit mode layout, the Save and Cancel buttons.



In its programming, note that the 'Save' event, which corresponds to the case when we're editing a client's information and we want to save that information in the database, uses the Save method of the Actions API.
The Save method of this API encapsulates the invocation to the corresponding business component that effectively performs the insertion in the database. Remember that this Business Component is a REST service, an API that exposes our application on the server.

We may wonder why not only Action.Save() appears in the invocation, but the full path is used: GeneXus. Sd. Action. Save()
Remember that both GeneXus and SD are modules and the objects in them are visible inside the module, but there can be other modules inside the same KB with objects that have the same name, but which are independent objects. In order to disambiguate this situation, we use the full path, naming the main container module and then the submodules up to the object we want to use.

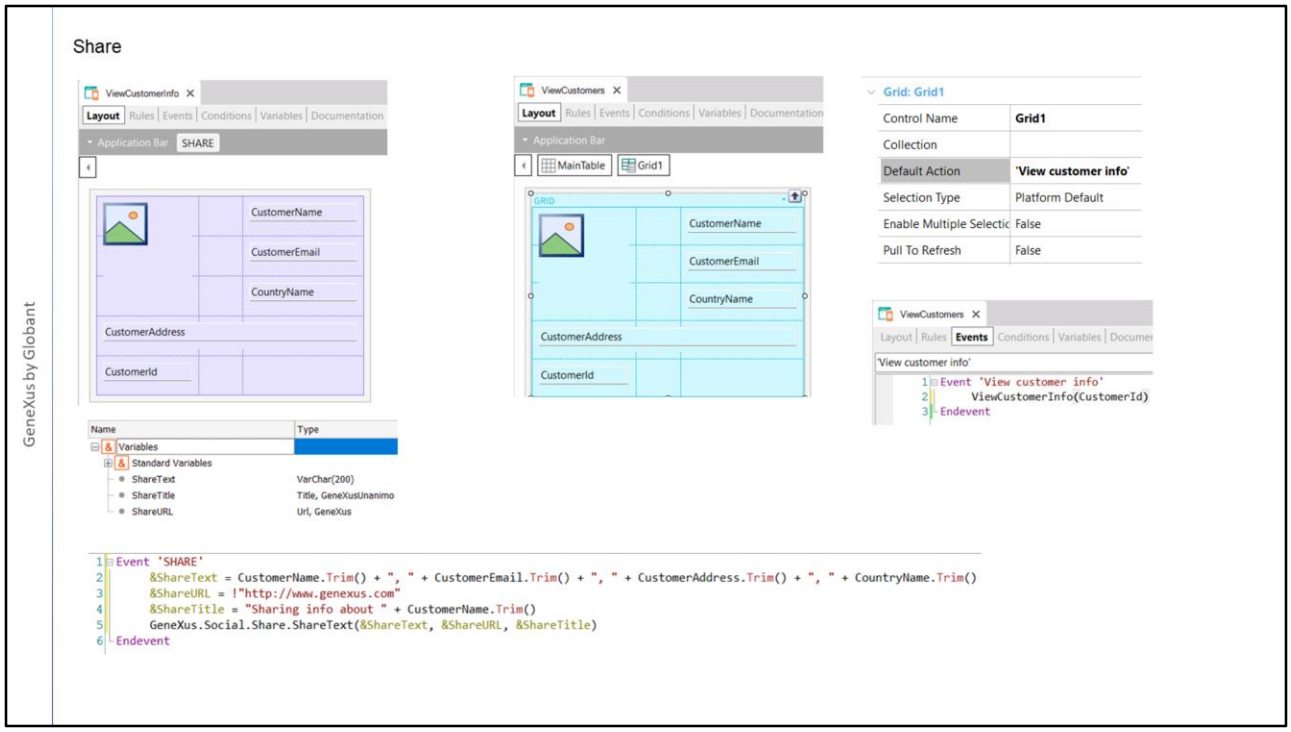
This depends on whether we have other objects with the same name, so using the full path may or may not be necessary. However, it is safer to always include the full path, especially if APIs developed specifically for the project are included and if many developers work on it.

Returning to the WorkWith example, note that the Cancel event uses the method of the same name from the Action API. All this is programmed by default in the WorkWith pattern, so if we apply this pattern to another transaction, we will see an analogous programming of these events using this API.

In the Social module, we had seen APIs to interoperate with Facebook, Twitter and WeChat, as well as to share information with one of the programs installed on the device.
Let's see examples of use of these APIs.

Each API has properties and methods according to its functionality. For example, the Share API has methods that allow us to share a text, an image, or a file.

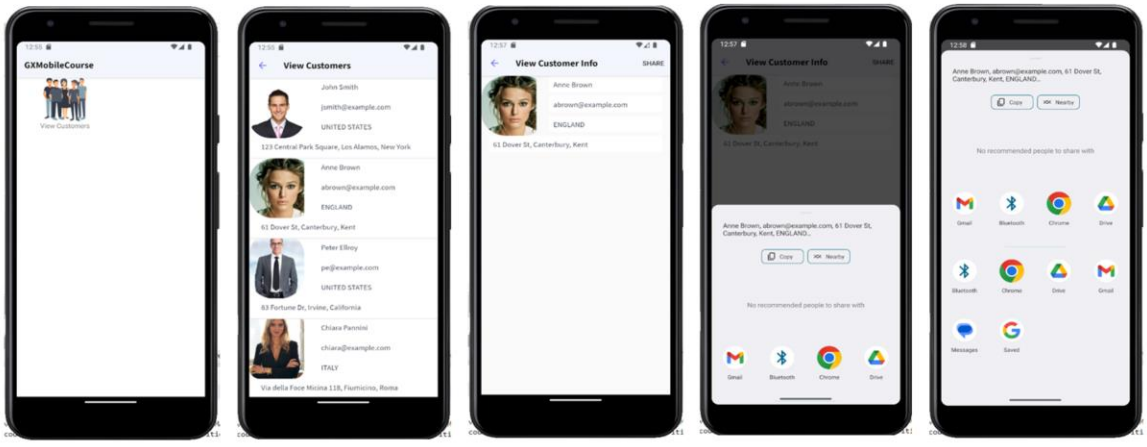
As for the Facebook and Twitter APIs, the methods allow us to perform the actions required to interact on these platforms, such as Facebook's PostToWall method or Twitter's Follow method.



Let's take a look at an example of using the Share API to share a client's data.
Here we have already created a ViewCustomerInfo panel that has the Customer attributes and receives the CustomerId in a parameter in the Parm rule.
We have already added a SHARE button to the Application Bar.
We have also defined three variables, one for each parameter –that is, one for the text with the customer data, one for the URL, and one with the title of what we are going to share.
In the button event we assign values to these variables and write the invocation to the Share API, passing it the required parameters.

In the ViewCustomers panel, we add an action with the name “View customer info” in the Default Action property of the grid, and in the corresponding event we invoke the ViewCustomerInfo panel, passing the identifier of the selected customer in a parameter.

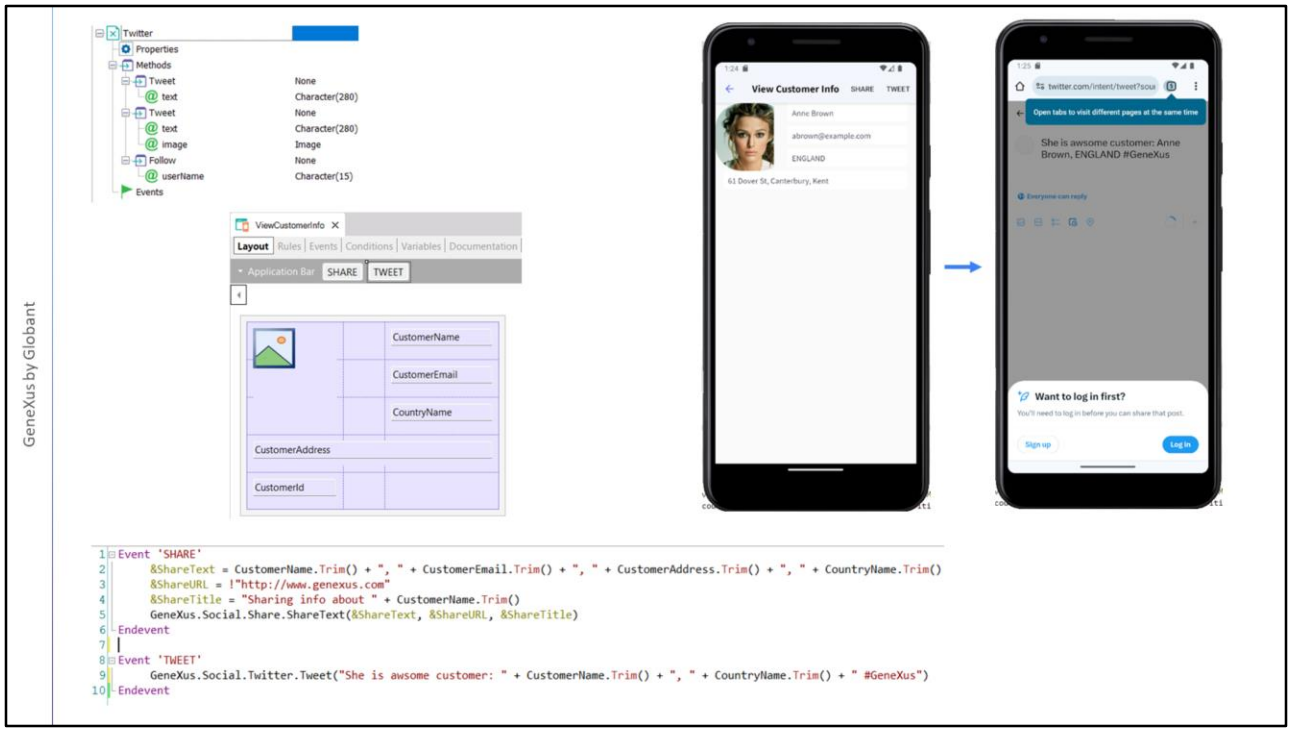
We run it...



We tap on the View Customers icon and then tap on the customer Anne Brown.

The panel with Anne's details opens and the SHARE button appears at the top right.

If we click on it... The customer's details are displayed for sharing them, as well as the applications offered by the device where we can use this information or share it.



We may also want to offer the action of “tweeting” a message related to the selected client.

To implement this we use the Twitter API and the Tweet method. The API offers two Tweet methods: in one we only send the message to be “tweeted” and in the other the text and the image. On the other hand, we have the Follow method.

We go back to the ViewCustomerInfo panel, add a TWEET button to the Application Bar, and in the button event we invoke the Tweet method of the Twitter API, passing in a parameter the text we want to tweet with the hashtag at the end.

We run it... we choose Anne Brown again and see that the browser opens on the Twitter page with a login prompt, but we can still see the message we tweeted. If we had the Twitter app installed, we would be offered to send a direct message, or write a general tweet.

We might want to do the same, but with Facebook, and we would use the corresponding API.

In this video, we saw how many APIs are available for different requirements and showed some examples of use.

As GeneXus evolves, more external objects with new functionality will become available, so we suggest you refer to the wiki documentation on a regular basis.