GeneXus Core Course

Considerations for native applications

Version: GeneXus 18

Considerations for native applications

We strongly recommend following the GeneXus Core Course as it is set up; that is, with a Web front-end (.NET) approach.

However, for those who want to map to native development as they watch the videos, here are the relevant considerations for each video, with the caveat that this may be confusing and it may be better to complete the course as is.

This is complementary material; it has no effect on the Core Course exam or any other part of the course.

Contents

First steps

"Creating the knowledge base"
If you also want to create a front end in Android or Apple, in the creation window select the corresponding check box, in addition to the default Web (.NET).

If you don’t do this now when creating the KB, you can do it later.

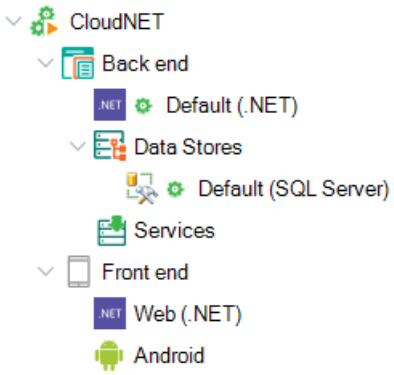
Transactions

“Designing the first transaction”

The transaction object has a front-end part (that is executed on the client, such as its screen) and a back-end part (that will be executed on the server, such as accessing the database, for example). The transaction object is only executed in full (front end and back end) when using the Web front end (.NET, Java).

The native front end (Android or Apple) does not execute the transaction object, but it will be able to use the back-end part of this object. It will be understood later in the course, when studying Business Components, that they will be able to act as services exposed on the server. The Work With pattern screens will also be supported, although in general they are not used in this type of application because the back office is usually web-only (either . NET/Java or Angular).

Since GeneXus automatically creates the application database from the transactions, even if we are developing a native mobile application, the back-end part will be implemented in one of the standard languages (.NET, Java). That’s why we will see the back end separately from the front end, in this way:



In other words, when developing for Android or Apple, .NET/Java must be used for the back end.

Therefore, to load data in the Customer transaction we will need a .NET/Java back office like the one in the course video –and that's why we see the Web front end (.NET)– or ... use the Work With pattern that will be studied several videos later, although it is not commonly used in native applications. Nevertheless, you will often have to add/delete/modify the database, which is usually done using the transaction object as a Business Component (this will be studied later on). For this reason, it is important to properly understand the logic of the transaction object.

"Running the application for the first time"

It is only valid for the web front end in Java/.NET, as explained at the end of the video.

"Attributes and domains", "Related transactions", "Transactions with more than one level", "Attribute naming conventions", "Defining rules".

It is important to understand all this because it has to do with database normalization, referential integrity, the rules that will be applied to the data to be entered, and the data itself and its tables.

"Using Patterns"

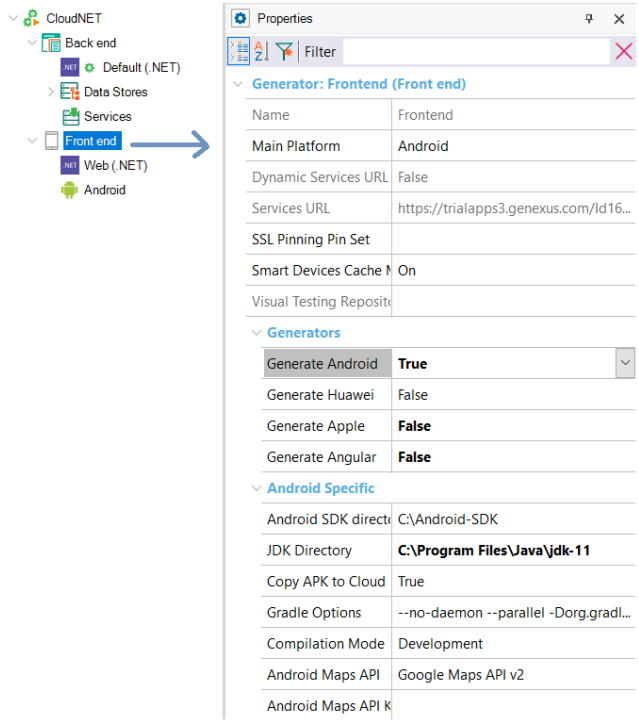
Mobile applications do not usually use the screens created by the Work With pattern because they do not usually implement the back office. However, it can be used anyway.

The pattern for native applications is not the one shown in this video ("Work With for Web"), but the general "Work With" (which in previous versions of GeneXus was called "Work With for Smart Devices"), which is also valid for Angular applications.

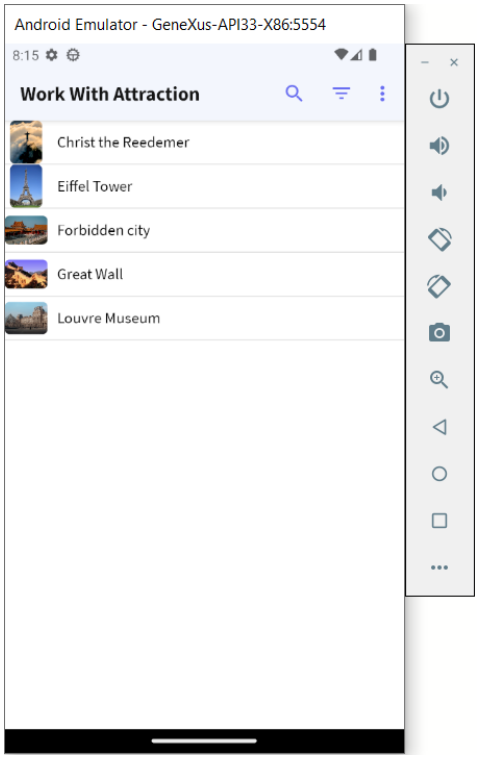
This specific video for Angular is attached for you to watch after this one, because basically what is shown there is identical to what will happen for native applications.

If you apply the "Work With" pattern to Attraction, so you can run it on Android or Apple:

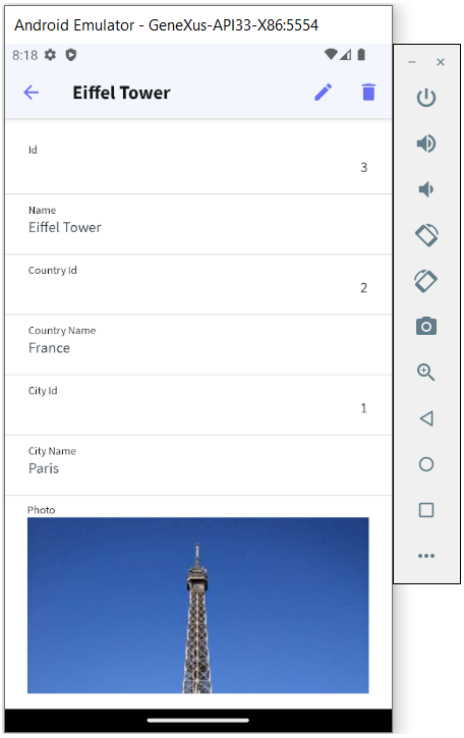
1. Make sure the Android or Apple front end is enabled. Here we will show the Android front end in order to use its emulator:



2. You will have to access the properties of the **WorkWithAttraction** object created by the pattern and set the “Main program” property to True. Next, from that object, right-click and select Run.
3. You will see that the Android emulator will open and inside it the Work With List panel:



And if you tap (“click”) on one of the attractions, the Detail of the attraction will open, in View mode:



Remarks on the Work With:
While it doesn't exist as an executable object for Web (.NET/Java), it does for mobile and Angular applications.

- In Web (.NET/Java) everything is configured from the tree shown in the video.
- In native and Web Angular applications, everything will be programmed as if it were an independent object.

As a result, the way of implementing the Work With screens is different in each case.

You will notice that once the "Work With" pattern is applied to the transaction object (in this case, Attraction), the **Business Component** property is automatically turned on. Later on, you will understand why; for now, just take note of it.

"Base table and extended table", "Defining subtypes"

They are valid exactly as they are.

"Defining attributes as formulas"

Although we could think that this will not be valid for native applications since no transaction object is generated there (instead, it will be executed in silent mode as a Business Component), and therefore nothing of what is shown running in this video will be seen in native applications, actually the definition of formula attributes applies at the transaction structure level, which means that it applies at the data model level. Therefore, the formulas will be valid in all platforms. The difference will be in the time when they will be calculated. However, the logic is exactly the same as presented here, on all development platforms. In fact, when later on we see how to run through a table and list its information, we will see how there the formula will be triggered automatically for each record without us having to do anything. The definition of a formula attribute is a logical definition that has implications for the database and the entire KB.

"Rule triggering events in transactions"

While at first glance it may seem that this will not apply to Android and Apple, actually the rules will also be executed when the transaction is executed without the screen, i.e. when its information is handled through the Business Component.

"Indexes", "Normalization of tables: A case study", "Relationships between actors of reality", "1 to 1 relationships between actors of reality", "Exporting and importing GeneXus objects", "Analysis of the transaction design model".

They are valid exactly as they are.

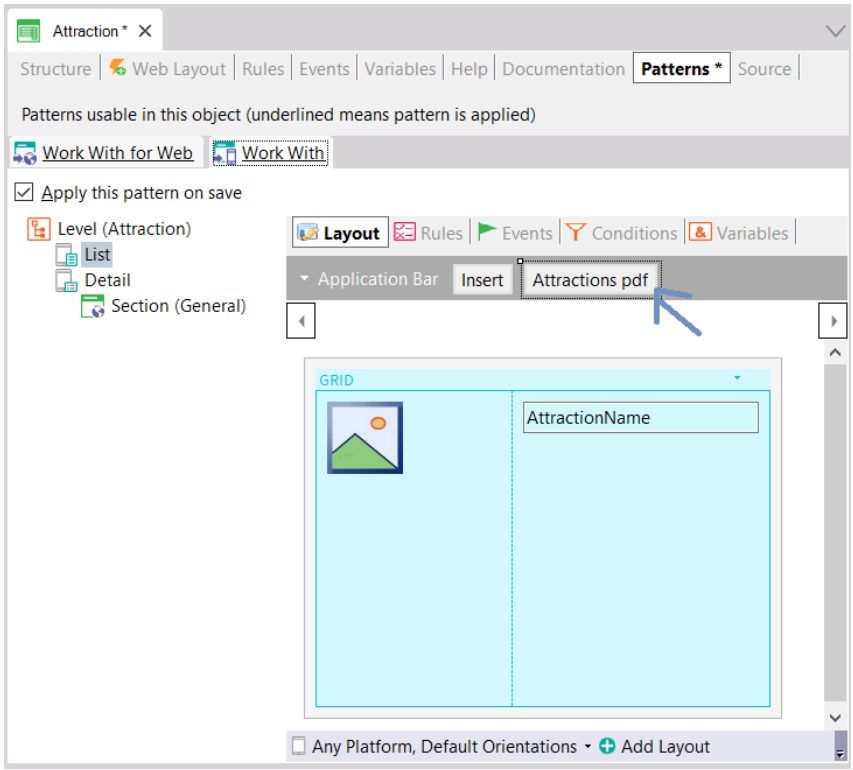
Lists and access to data through code

"Lists and For each command to query the database"

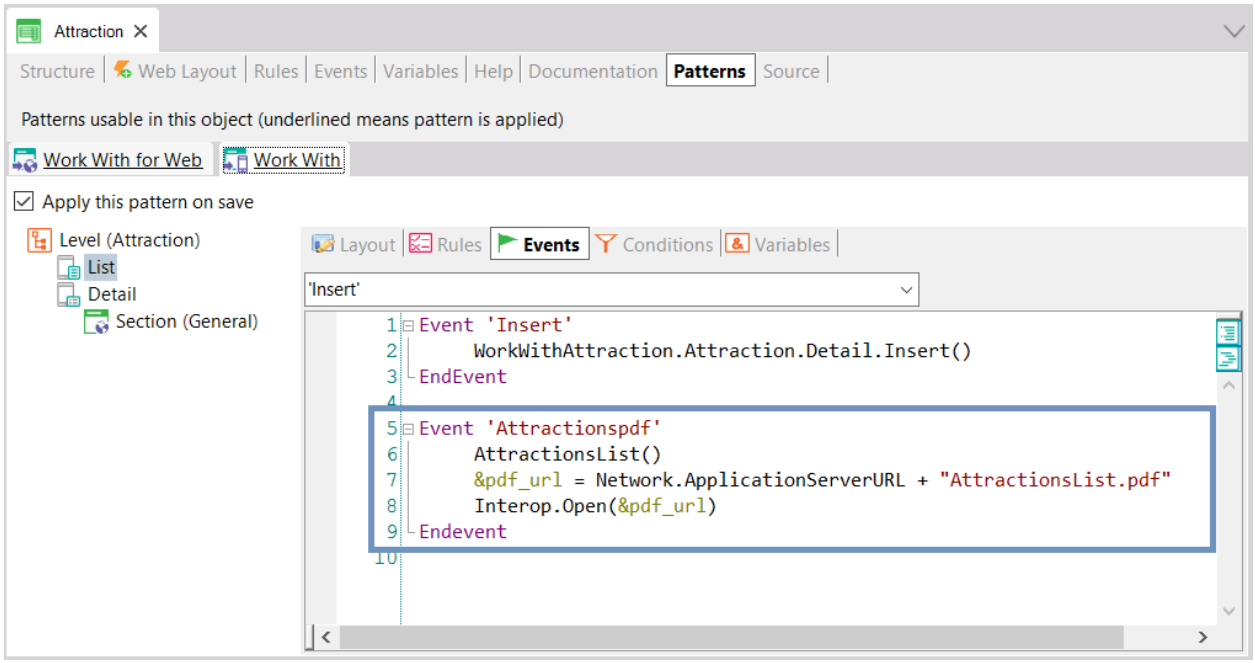
If you follow the video exactly as it is, you will see that the list will open in the .NET front end URL.

In order to run the list and open the PDF from the native application, the following must be done:

- Leave the "Call protocol" property with the default value, Internal.
- From a native panel (it can be WorkWithAttraction), add a button to invoke the procedure.



And in the Events tab add what is shown below:

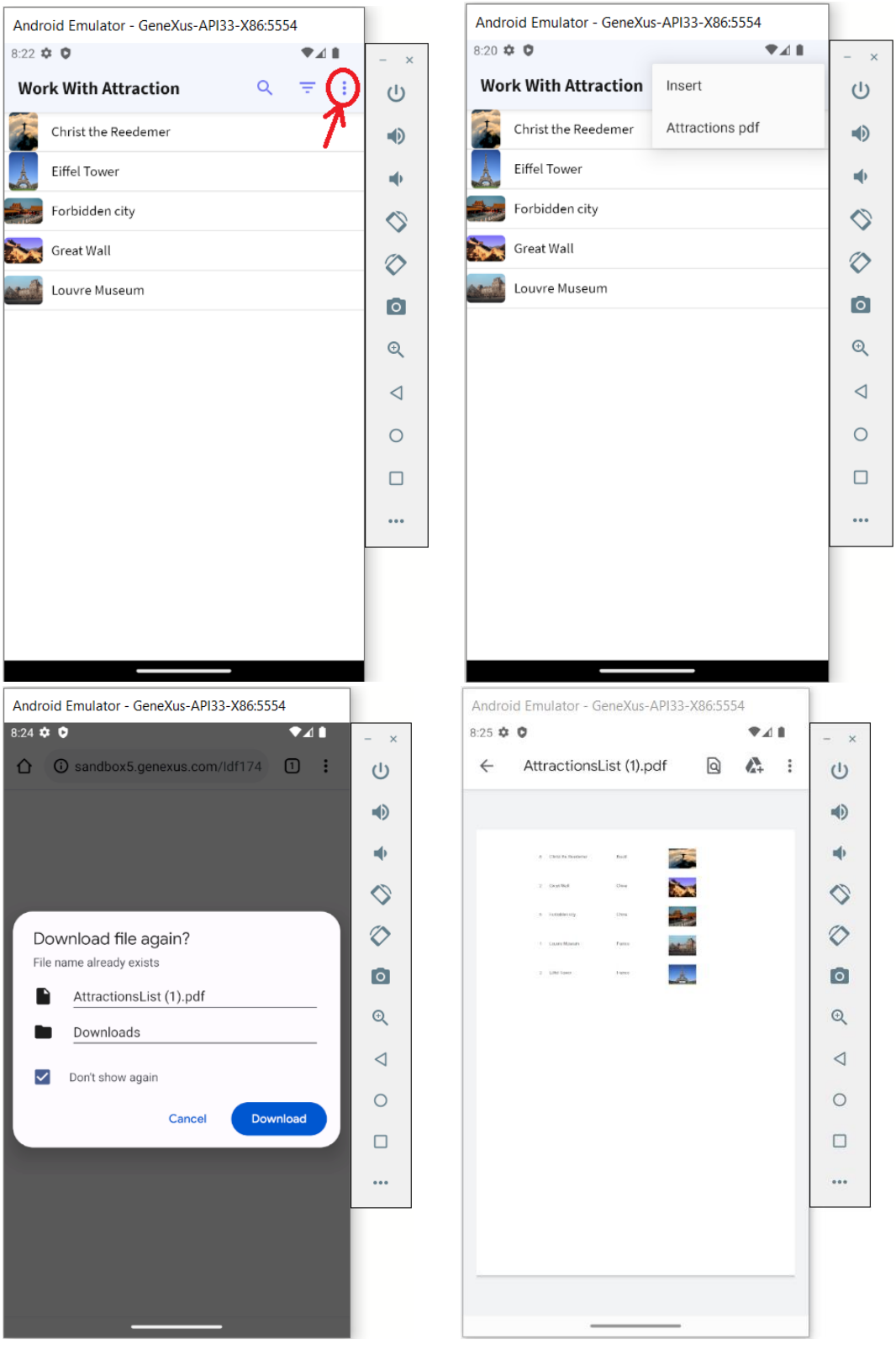


Where:

- The &pdf_url variable must be of the **Url domain** (url, GeneXus:Domain).
- "AttractionsList.pdf" is the name given to the PDF generated by the AttractionsList procedure, since the rule was specified there:

  Output_file("AttractionsList.pdf", "pdf");

- Run this WorkWithAttraction panel object (which must have the "Main program" property set to True) with right-click / Run. From there, when the panel is loaded in the Android emulator, open the menu and press the button to print the PDF. The browser will open to download the PDF file. Once downloaded you can open it (in the emulator it will open in Drive).



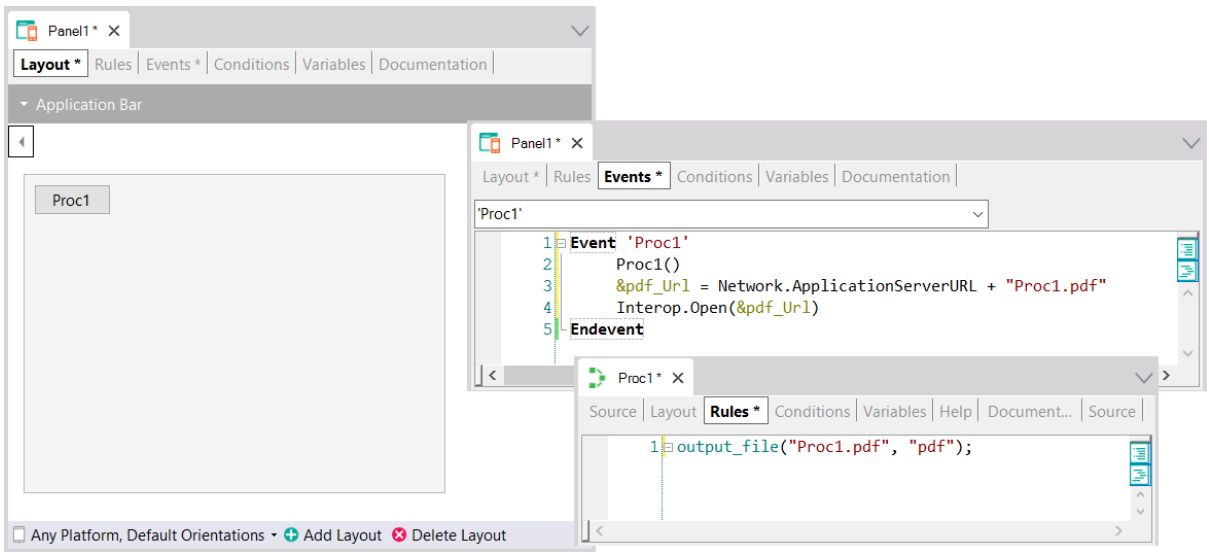
- Explanation for those who are more advanced:

- The invocation to the procedure, AttractionsList(), asks for the procedure to be executed (and this will be done on the server). GeneXus will have set the “Expose as Web Service” property of the procedure to expose it as a Rest service, so that it can be invoked from the client. This will be understood when studying the architecture of native applications.
- As the procedure has the **Output_file** rule, the result will be saved on the server as a file named AttractionsList.pdf (the one indicated in the output_file rule).
- Since we want this file to be opened by the Android front end, we must ask the front end to open the resource (in this case, the PDF). To do so, we must first indicate the URL on this server, which is what we put together in the second statement of the event (the ApplicationServerURL property of the Network external object will return the URL where the Rest services are located).
- To open a resource, the Interop.Open(...) method is available. Interop is an external object that comes in the GeneXus module (look for it below the References node).

“How to process related information”, “How to process grouped information”, “Inline Formulas”

The same considerations: to run these procedures, you will have to do the same as in the previous case. Considering that what is important here is not the native execution but the logic of the procedure itself, we suggest running as shown in the videos (Web . NET front end) and not in the front end of the native application.

If you want to do it in Android/Apple anyway, you can create a **Panel object** and in the Layout place as many buttons as Procedures you need to invoke. The variable that will contain the URL (&pdf_Url) must be defined with the **Based on** property with the “Url” domain.
You must change the **Main Program** panel property to **True**, and then you can run it simply with **right-click/Run** on the panel.

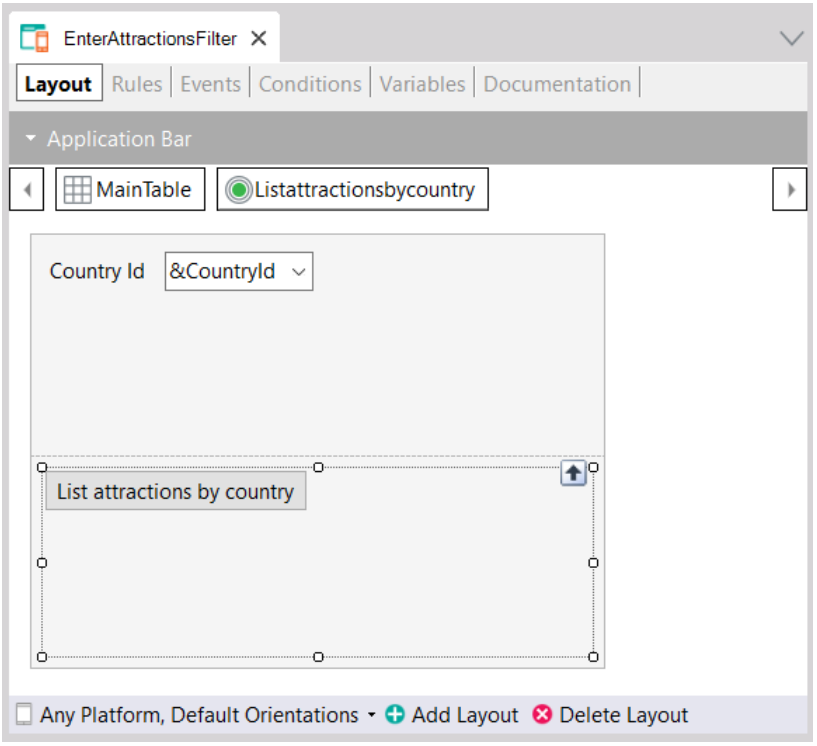


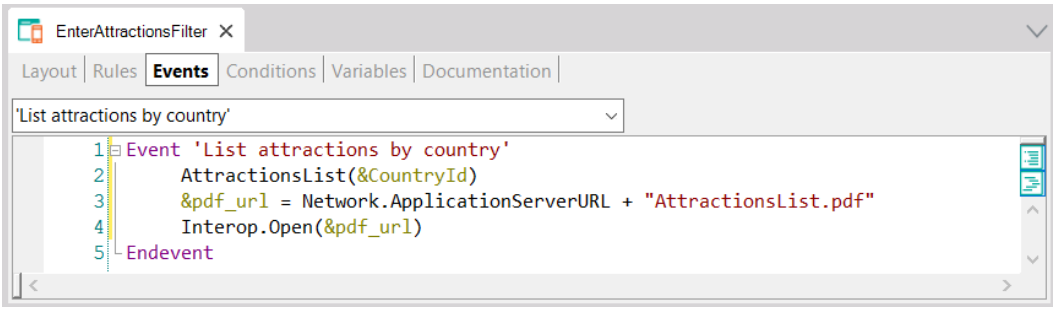
Communication between objects

"Invocations between objects", "Invocations between objects (continued)"

The Web Panel analog for native applications will be the Panel. For the sake of simplicity, we recommend following the videos as shown here, for Web Panels and not for Panels.

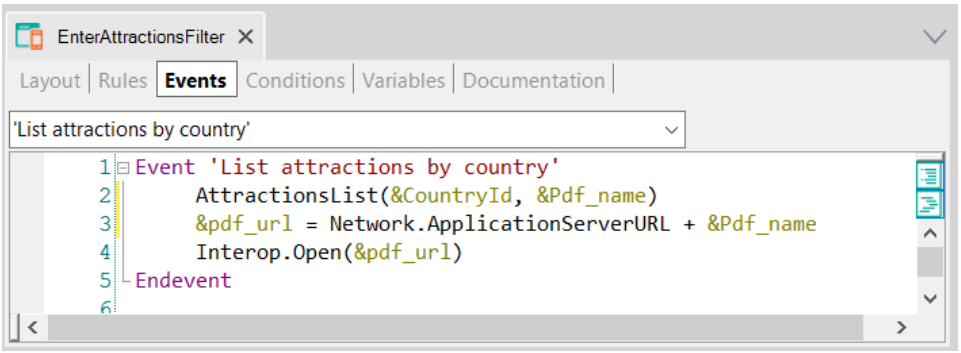
An option to run in Android/Apple the first panel of the first video, EnterAttractionsFilters:



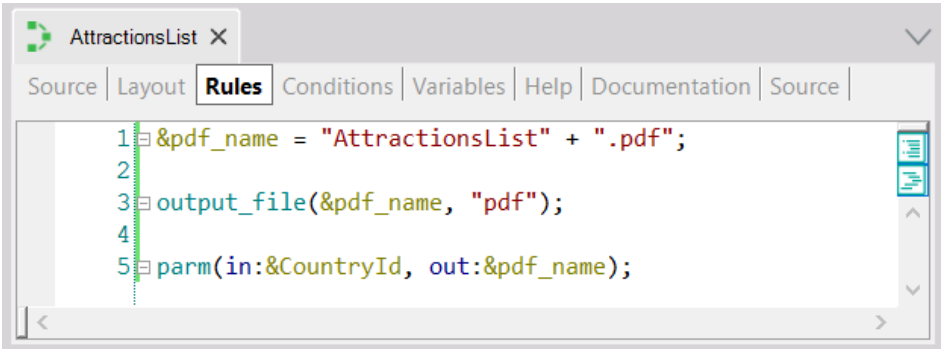


Set the panel as "Main program" and make sure that the AttractionsList proc has the value "Expose as web service" = True, "Rest Protocol" = True and "Call protocol" = Internal.

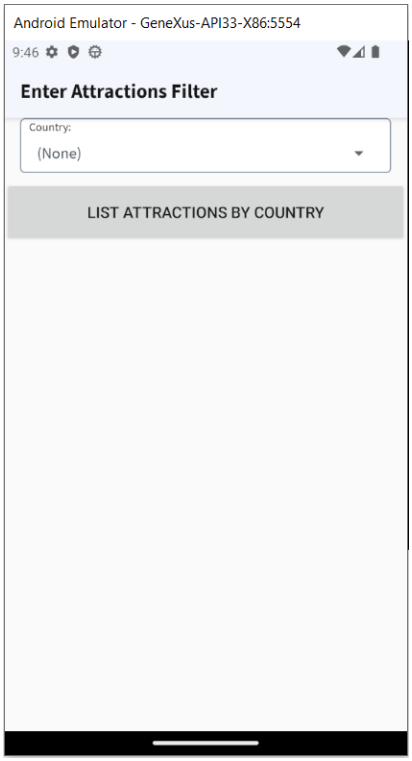
A better way to implement this so that we don't have to remember the name given to the PDF within the procedure is to also pass it by parameter:

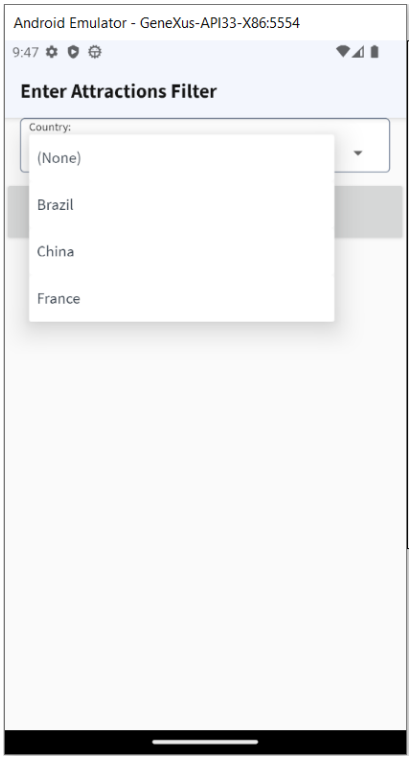


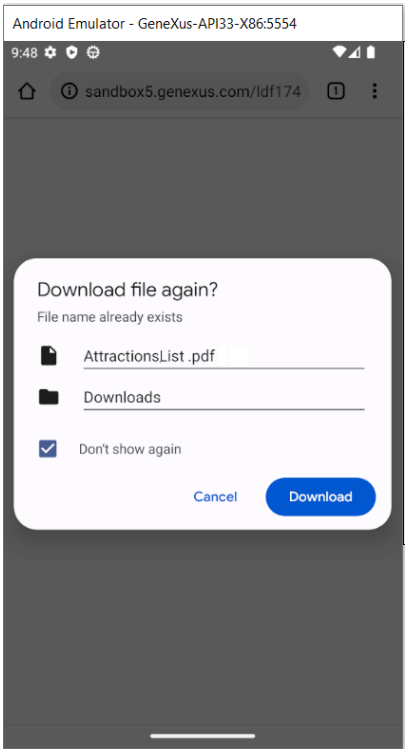
Thus, in the AttractionsList procedure:

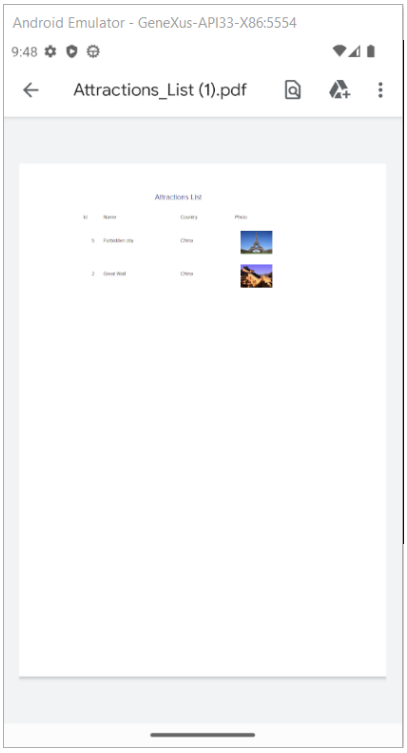


At runtime, you will see:





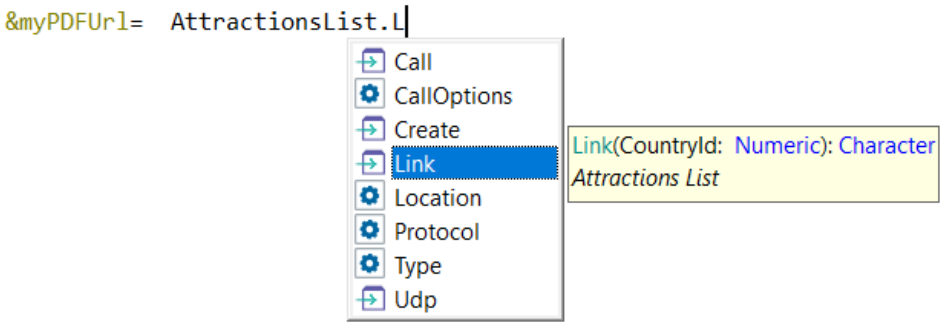




Additional observations:

When you call an object directly by its name and the list of parameters in brackets, it is as if you were writing the Call method (or Udp if the object returns a parameter). But there are other ways

to invoke them, for example, with the Link method. Now we will not dwell on their similarities and differences, we will just name it.



Structured Data Types and Data Providers

"Structured Data Types", "Variables that store data collections in memory", "Loading Structured Data Types (SDTs) using Data Providers"

All this is very important for all platforms and applies exactly the same. The consideration is the same as we saw in the previous lists (how to view a PDF list in the native front end). We suggest, as before, to follow the videos as they are, in Web front end (.NET).

Database update

These videos are essential, even more so in native applications, where we don't use the transaction as a UI object.

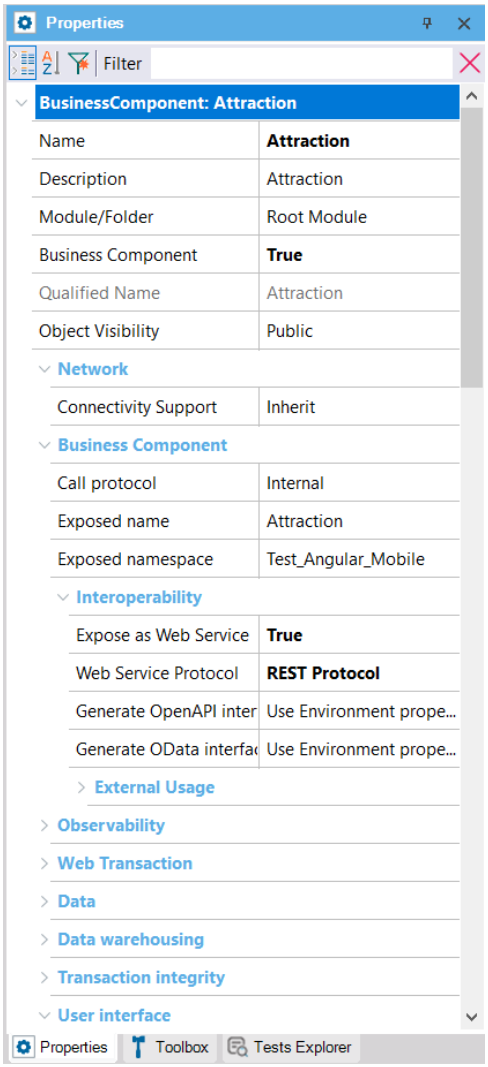
"Database update using Business Components. Justification"

When we mention the rules that the Work With pattern added to the Attraction transaction, we are referring to the pattern for Web, not the generic Work With pattern, which is the one valid for Angular or native applications. Let's remember that in Angular and native applications, the transaction object will not be executed. In fact, what will be used is what is being introduced in this video; that is, the Business Component created from the transaction.

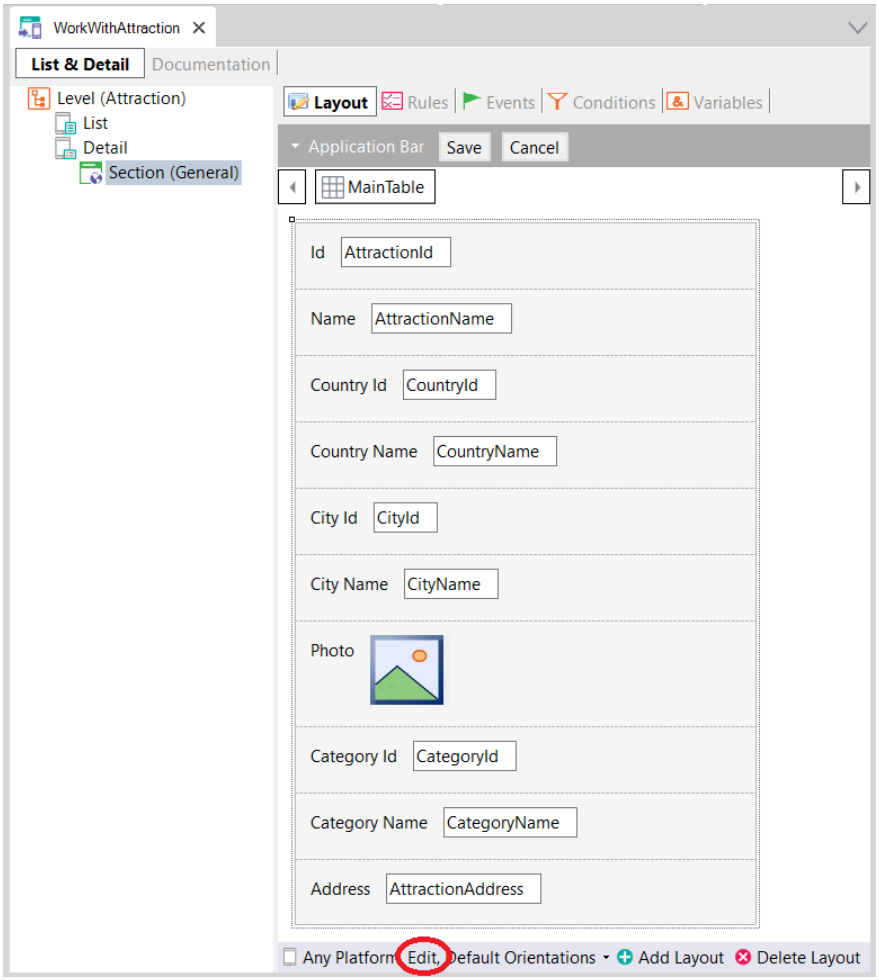
"Update using Business Components"

First, make sure you understand the video as it is, and then read the following.

If when we studied the Work With pattern you created the pattern for Android/Apple, then you will have noticed that GeneXus automatically turned on the Business Component property of the transaction and the interoperability ones shown in the image:



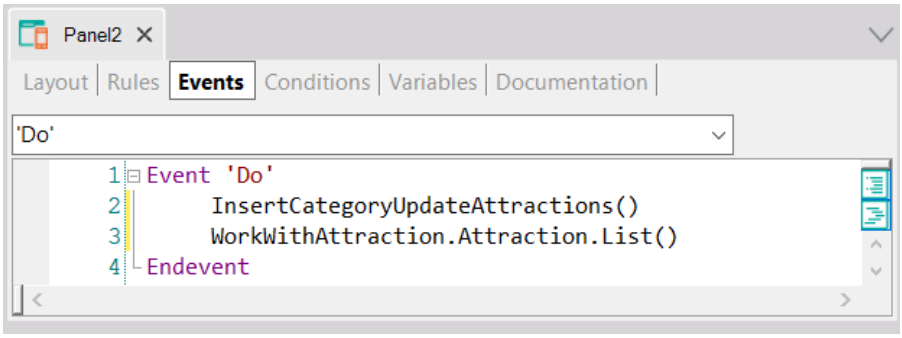
This happened because to be able to insert attractions, as well as to modify or delete through the Work With screens, in native applications you don't call the Attraction transaction but the screen corresponding to the General section, Edit layout.



Those “attributes” that we see on the screen are not actually attributes. They are the elements of the Business Component. When you click on the **Save** button, what will be done internally is to call the Rest service corresponding to the Business Component, published on the server, to perform the corresponding operation on the database. In short, from a client event, a call will be made to a Rest service of the server, which is the one that will try to update the database through the Business Component, and if it is successful, it will perform the Commit. The Rest service will automatically try to perform this Commit.

The case is different if you want to use the BC from a procedure, because the procedure is executed in the back end and there everything is identical to what was seen in the video.

To execute the InsertCategoryUpdateAttraction procedure, create a Panel with a Do button in the layout and in the associated event write:



Again: the procedure with “Call Protocol” = Internal. Note that when invoking the procedure from the event, GeneXus automatically turns on the properties “Expose as Web Service” and “REST protocol” of the procedure. We take advantage of this and invoke the Work With List to see how the new attraction is listed.

When the Business Component is used inside a procedure, it is not necessary to execute it as a Rest service (the Business Component, not the proc), because the procedure will be executed on the server, and it is there that it will use the Business Component. For this reason, the commits can be concatenated there as the developer wants, making them explicit. Everything as seen in the video.

The final part of the video, where it says that BCs can be used not only in Procs but also in Web panel events, is the part that will not be so identical in native applications, where it will have limitations.

“Update using Business Components. Example”

To do this in native applications, instead of the “CategoriesAndAttractions” Web Panel, create the analog Panel (and set it as main to be able to run it with Run).

For the first part, when invoking the proc from the ‘Do’ event of the panel button, everything will be identical (although note that the proc will automatically be exposed as a Rest service).

BUT: when in the middle of the video the procedure Source code is copied to the ‘Do’ event of the Web panel, things will not work in the same way. That’s because when you do this in the Web Panel, even if it is a client event, GeneXus internally moves this code to the server, and executes it there.

For **Panels** this will not be the same. In the Panel, that code will be executed on the client, not on the server. And, logically, on the client there is no access to the database. Assigning values to the BC elements and even the Insert or Update operation will work, because for this the BC is invoked as a Rest service. However, the result query with If will not work, not even If &category.Success(), and even less the Commit or Rollback.

To perform the Commit or a Rollback there will be no other choice but to invoke a procedure that performs it (which will always be executed on the server side). So, in the end it is convenient to do everything in that procedure.

It will not be possible to use them in client events (executed in the front end), For each commands, or formulas that access the database, such as find(CategoryId, CategoryName = ''Monument'').

Therefore, these implementations in a Panel will have to be done by invoking procs in the events (because they are executed on the server, they do have all the database access commands available).

You will understand all this better when you watch, in the Architecture section, the video called "GeneXus applications architecture" which is three videos ahead.

"Data population with Business Components and Data Providers"

The Insert method for BC collection will not work at the client event level, so you will have to include the entire event code in a procedure.

"Automatic data population"

The Data Provider created by GeneXus to populate the Category table with data (and the same goes for Attraction) will be executed in the back end; that is, it will be generated in .NET in our case. It is therefore independent of the front end. The .NET front end in the video is being used only to display the entered data.

"Update with procedure-specific commands. Introduction"

It is valid as is, taking into account that these commands that access the database can logically only be executed on the server side, that is, in the back end.

Architecture

"GeneXus applications architecture"

It is important to understand the differences in the architecture of an application whose front end uses Web panels (.NET or Java) as opposed to one that uses Panels (Angular or native).

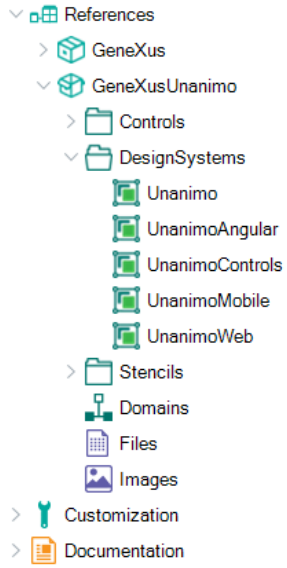
Web screens with back-office focus

All the videos included here are for a Web front end (.NET or Java). Although the Web Panel object is not used for Angular or native front end, part of the logic will be valid, with some differences, for Panels, and here it is explained from scratch, so we suggest watching these videos as carefully as the previous ones.
Then, in the native application development course, you will study the specific features of Panels.

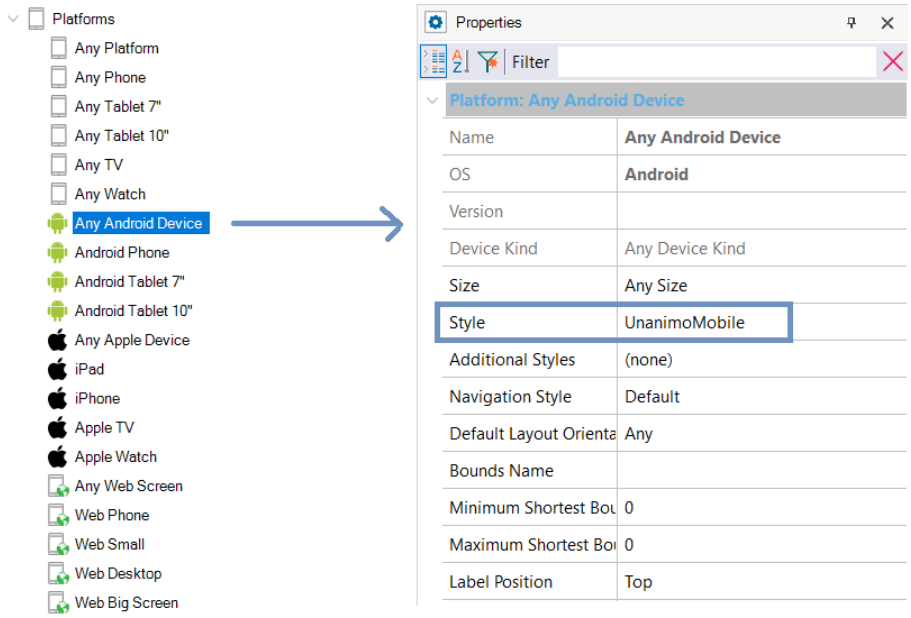
Screen design and modeling

UX Design. Introduction
What we show in the video is for a Web front end (.NET/Java), but it is conceptually almost the same as for any other front end. Some important differences:

- For the Web front end (.NET/Java), a Design System Object is defined that will apply by default to all screens through a version property named Default Style. When a KB is created, a DSO with the same name is created by default, and this DSO will import by default everything that comes from the predefined one named UnanimoWeb:



- For the native front end, the way to indicate which DSO will command the layout of the screens is from the platform properties. Here you can see that the default value is the UnanimoAngular DSO, which comes predefined in the GeneXusUnanimo module.



We recommend that you follow this video as it is set up. You can also take this opportunity to explore the UnanimoMobile DSO and try the WorkWithAttractions that you had applied for the Attraction transaction for Android.

Later on (not now), you may be interested in watching this webinar where we explain the first steps to design from scratch some screens for a customer-facing Angular application, without relying on any pre-existing DSO but creating one from scratch. It also mentions how it would be for native applications.

All the other videos of the Core course are valid as they are.