GeneXus by Globant

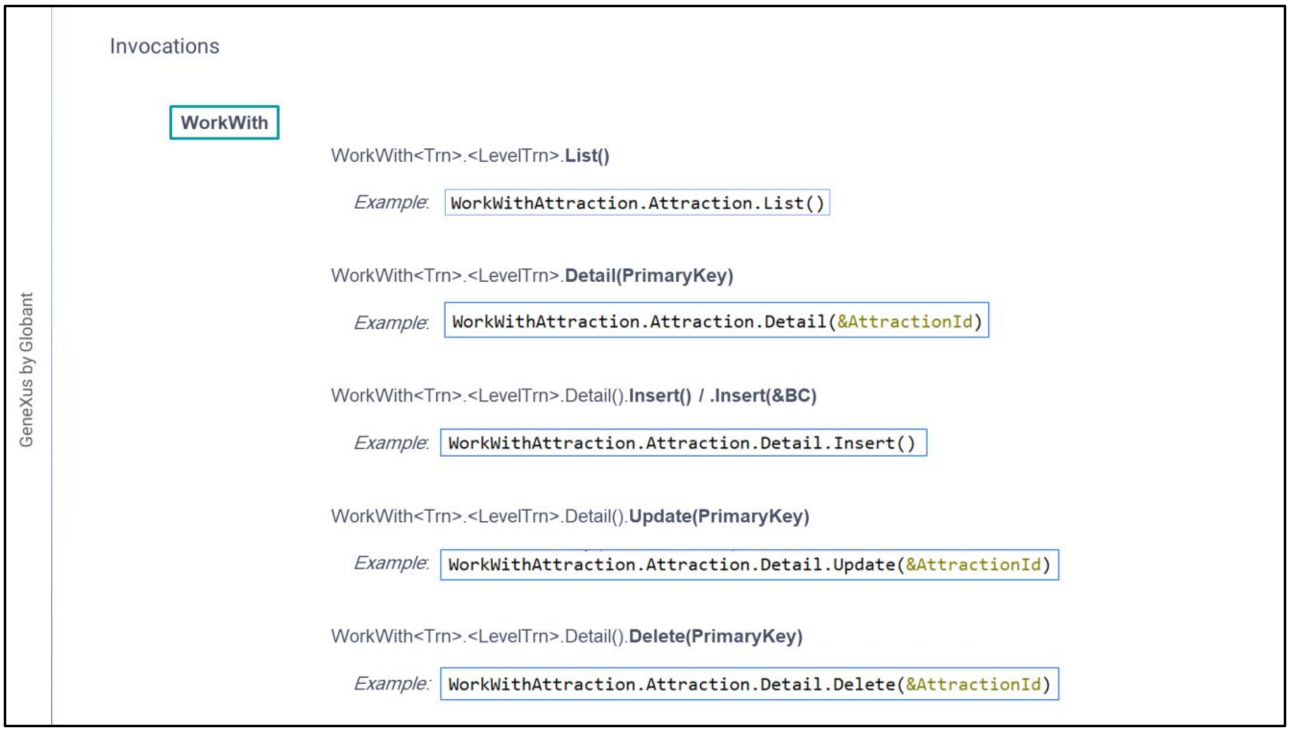
GeneXus™ by Globant

training.genexus.com

GeneXus by Globant

Invocations Between Mobile Objects

Diego Marranghello


Invocations

WorkWith

WorkWith<Trn>.<LevelTrn>.**List()**

Example: `WorkWithAttraction.Attraction.List()`

WorkWith<Trn>.<LevelTrn>.**Detail(PrimaryKey)**

Example: `WorkWithAttraction.Attraction.Detail(&AttractionId)`

WorkWith<Trn>.<LevelTrn>.Detail().**Insert()** / **.Insert(&BC)**

Example: `WorkWithAttraction.Attraction.Detail.Insert()`

WorkWith<Trn>.<LevelTrn>.Detail().**Update(PrimaryKey)**

Example: `WorkWithAttraction.Attraction.Detail.Update(&AttractionId)`

WorkWith<Trn>.<LevelTrn>.Detail().**Delete(PrimaryKey)**

Example: `WorkWithAttraction.Attraction.Detail.Delete(&AttractionId)`

In this video, we will see how to invoke different objects and the invocation options available. For the latter we will use CallOptions, which will allow us to modify at runtime certain properties related to the user's experience.

But first let's review the invocations' syntax.

For example, the case of invocations to WorkWith objects:

To invoke the List of a WorkWith, we indicate the name of the WorkWith, period, the name of the level, and then the List method without parameters. In this way, we will access the list of records.

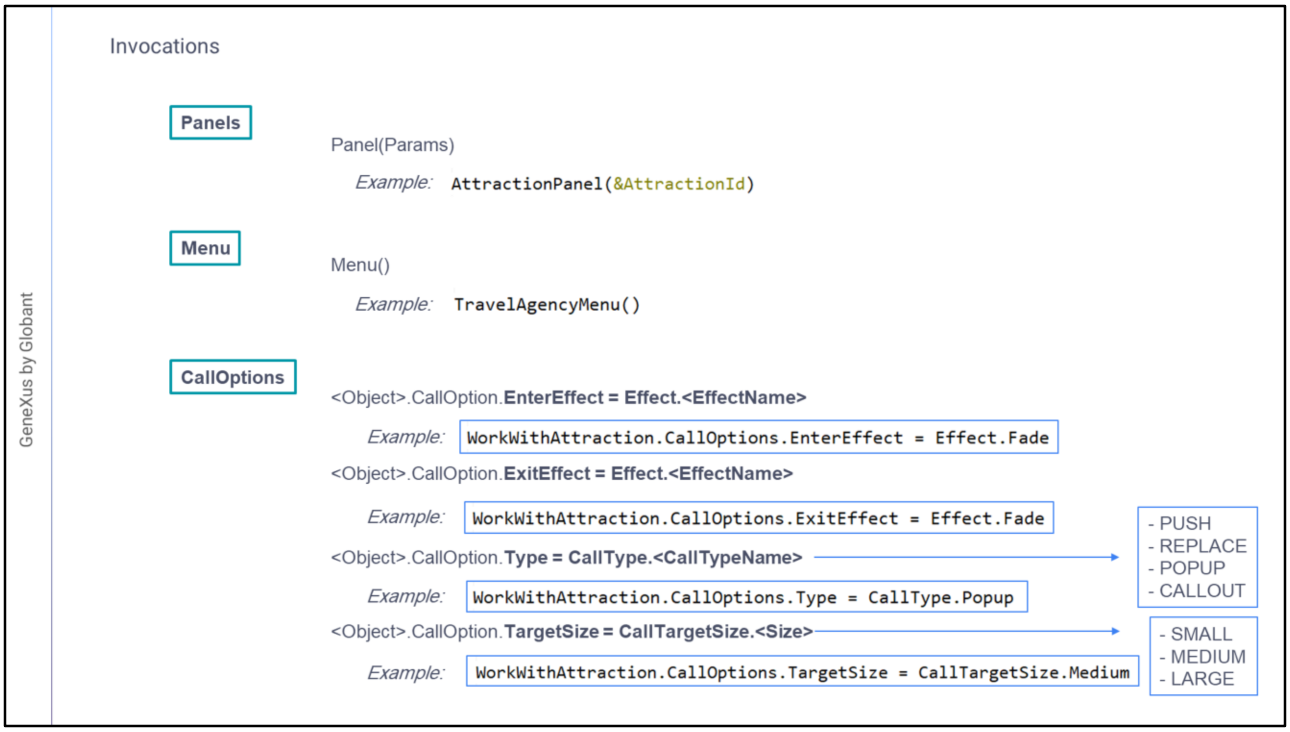
To access the details of a particular record, in view mode, we use the syntax shown on the screen, where we need to pass the primary key to identify whose detail we want to show.

For the Detail in Edit mode, we need to indicate the mode:

Insert, Update, or Delete. That is, if it will be a new record, or an update or deletion of an existing one.

In the case of Insert we have 2 options, including not to pass any parameter so that the user can enter all the data from scratch. Or if we want to already initialize some values in that Insert, we must pass them in a Business Component. For this case, we must previously initialize in the Business Component the values that we want to pass and then those values will appear initialized on the screen.

And for the cases of update and delete, we must pass by parameter the primary key of the record that we want to delete or update.



GeneXus by Globant

Invocations

Panels

Panel(Params)

Example: `AttractionPanel(&AttractionId)`

Menu

Menu()

Example: `TravelAgencyMenu()`

CallOptions

<Object>.CallOption.**EnterEffect = Effect.<EffectName>**

Example: `WorkWithAttraction.CallOptions.EnterEffect = Effect.Fade`

<Object>.CallOption.**ExitEffect = Effect.<EffectName>**

Example: `WorkWithAttraction.CallOptions.ExitEffect = Effect.Fade`

<Object>.CallOption.**Type = CallType.<CallTypeName>** → - PUSH - REPLACE - POPUP - CALLOUT

Example: `WorkWithAttraction.CallOptions.Type = CallType.Popup`

<Object>.CallOption.**TargetSize = CallTargetSize.<Size>** → - SMALL - MEDIUM - LARGE

Example: `WorkWithAttraction.CallOptions.TargetSize = CallTargetSize.Medium`

On the other hand, we have the invocation to the Panels, which is exactly the same as the invocation to any other GeneXus object, indicating the name of the panel, and passing the necessary parameters according to the input parameters that the Panel object has.

We can also call a Menu object.

Then we have the CallOptions, which as we said will allow us to modify at runtime certain properties related to the user's experience; these configurations must be done right before invoking the object.

The properties that can be configured are as follows:

• The transition effects, which can be either Input or Output. The possible values are those of the Effect domain.

This is an enumerated domain, which offers all these options.

• The type of call, using the CallType enumerated domain, with Push, Replace, Popup and Callout types (the latter only available on iOS).

This configuration allows us to modify the behavior of the call, relative to the call type, which will have to do with the stack of invocations and with the operation of the called object.

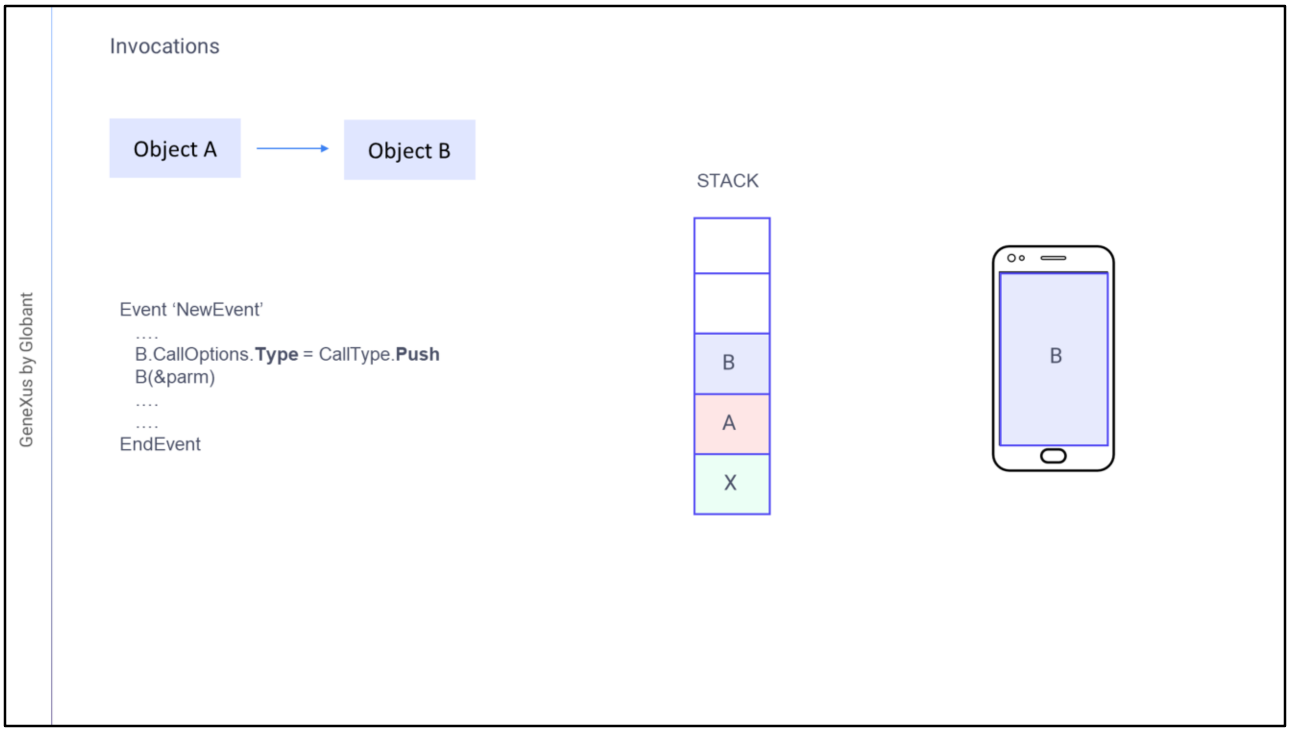
The Push and Replace types define what will happen to the invocation stack when the call is made, which will have to do with which object is returned to at the end of the execution of the call or when going back.

The Popup and Callout types will be used for invoking screens of popup or callout style.

For Popup, the screen may be modal or not depending on whether there are parameters returned. Modal windows block all functions and focus on a particular action. The user can only do that action or close the window; if it is not modal, tapping outside the area will return to the caller.

In the case of Callout, it will not be modal.

When we choose one of these two options, the other CallOption appears: CallTargetSize, to indicate the size of the Popup or Callout screen, with the options Small, Medium and Large.



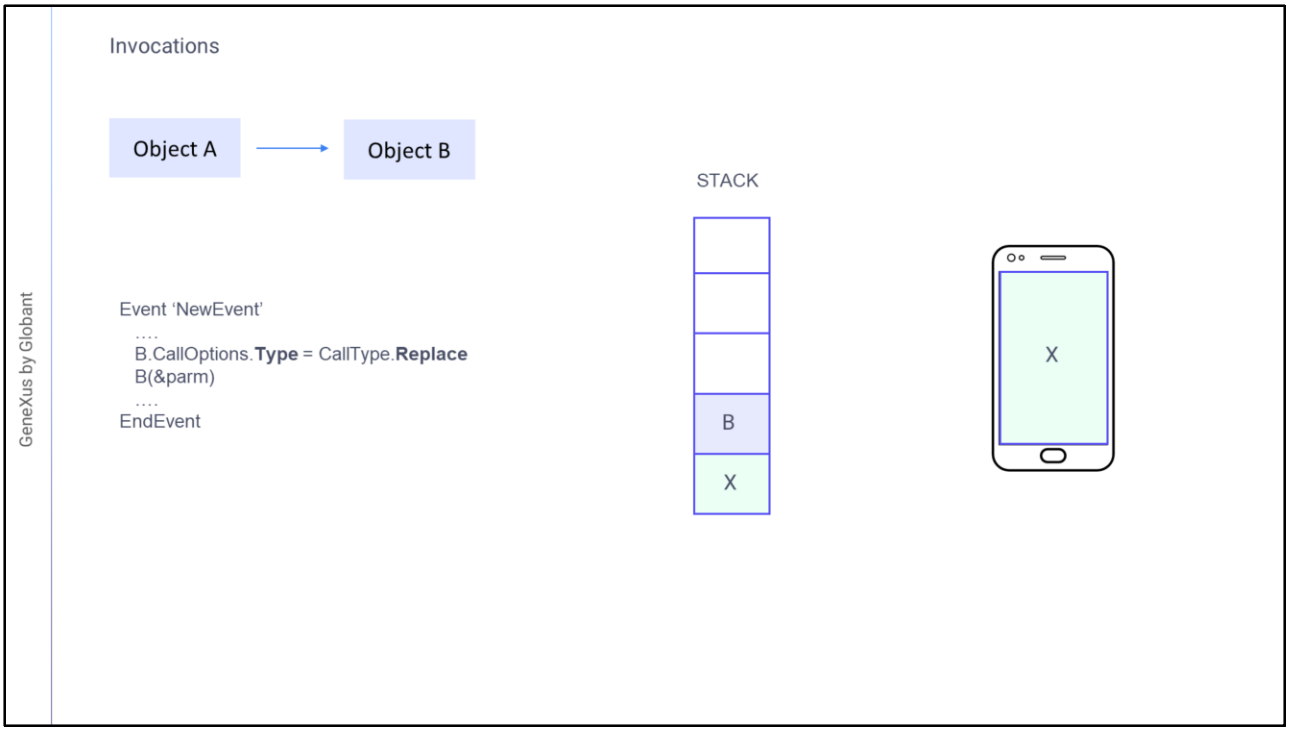
Invocations

```
Event 'NewEvent'
   ....
   B.CallOptions.Type = CallType.Push
   B(&parm)
   ....
   ....
EndEvent
```

As for the Push and Replace types, to understand the operation of both, let's see a small example:

Suppose that an object X called an object A.
If now from A in an event we call an object B with the type of call Push, the called object is placed on top of the stack.

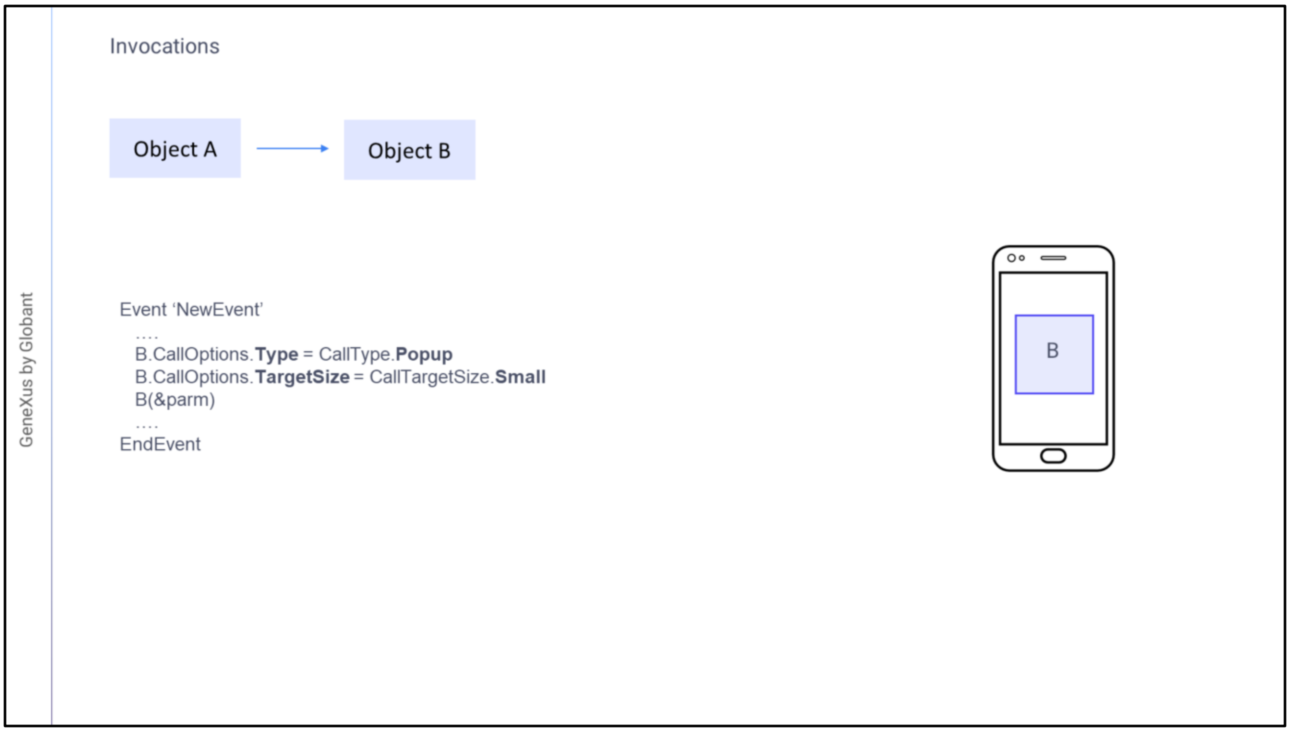
Its screen opens above the caller's screen, taking exactly the same place. And to continue running the caller waits until the execution of the called object B is finished, which is thus removed from the stack.



Invocations

```
Event 'NewEvent'
  ....
  B.CallOptions.Type = CallType.Replace
  B(&parm)
  ....
EndEvent
```

If instead from object A we call B with the Replace call type, the called object will also be opened taking the same screen area as the caller, but it will replace the caller object in the stack. So, when its execution is finished, it will not continue to run A's event, but it will return to the object that was previously in the stack; in this case, the object: X



GeneXus by Globant

Invocations

Object A
Object B
B

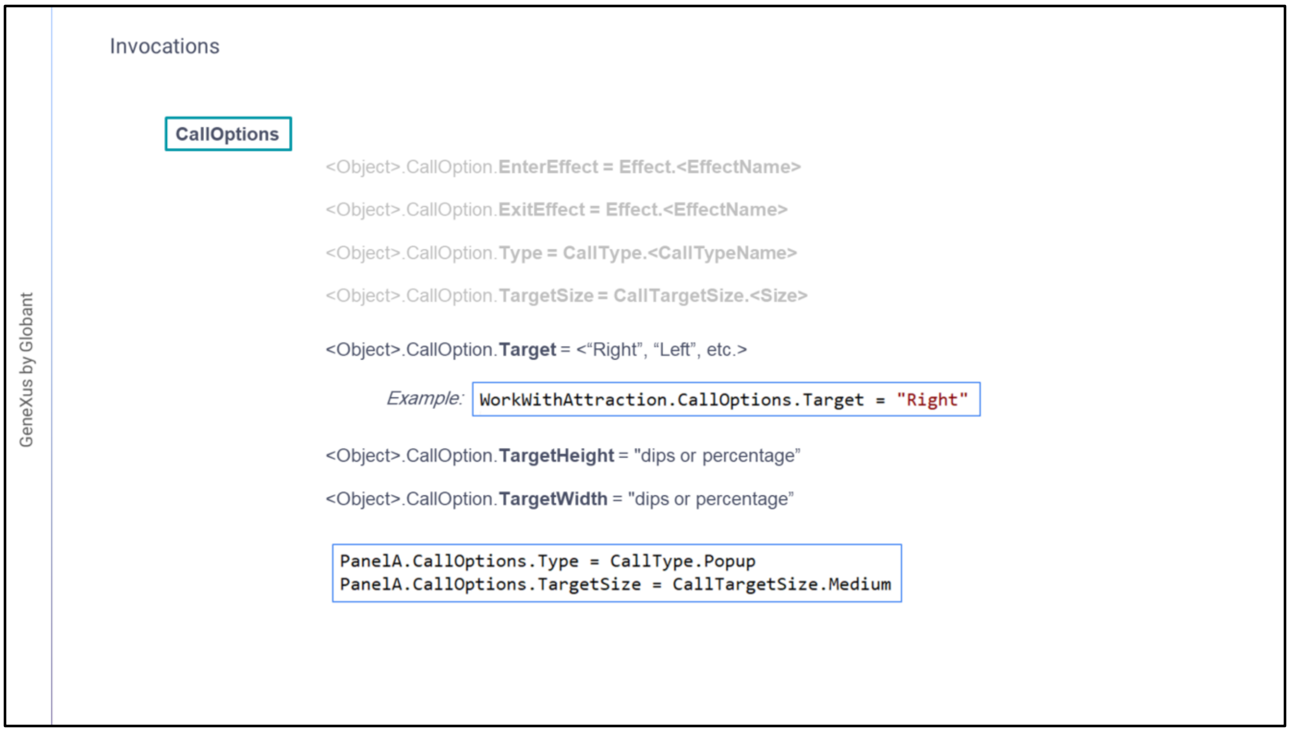
```
Event 'NewEvent'
   ....
   B.CallOptions.Type = CallType.Popup
   B.CallOptions.TargetSize = CallTargetSize.Small
   B(&parm)
   ....
EndEvent
```

Now regarding Popup and Callout, let's look at the Popup type.

If the TargetSize is not changed, it will take the same screen area as the caller.
Otherwise, it will take up the area we have specified.

If any of the invocation parameters is output, the dialog will be modal; that is, the caller will wait for the return of the execution of B to continue.

If none of the parameters is output, the dialog will not be modal.


GeneXus by Globant

Invocations

CallOptions

<Object>.CallOption.**EnterEffect = Effect.<EffectName>**

<Object>.CallOption.**ExitEffect = Effect.<EffectName>**

<Object>.CallOption.**Type = CallType.<CallTypeName>**

<Object>.CallOption.**TargetSize = CallTargetSize.<Size>**

<Object>.CallOption.**Target** = <"Right", "Left", etc.>

Example:
```
WorkWithAttraction.CallOptions.Target = "Right"
```

<Object>.CallOption.**TargetHeight** = "dips or percentage"

<Object>.CallOption.**TargetWidth** = "dips or percentage"

```
PanelA.CallOptions.Type = CallType.Popup
PanelA.CallOptions.TargetSize = CallTargetSize.Medium
```

Returning to calloptions, we also have:

• The target, where the called object will be displayed, to indicate in which of the possible targets we want to load the called screen.

This makes sense when there are navigation styles with multiple targets for the same screen –that is, split screen, which is more used in tablets than in phones. It cannot be used with Callout or Popup.

• In a customized way we can define the size of the called window, specifying the Width and Height, in dips or percentages, using the TargetWidth and TargetHeight properties.

For example, if we wanted to call a panel as a Popup with a Medium size, we could configure the Type and TargetSize properties as shown on the screen and then call the object as usual.

GX

GeneXus by Globant

GeneXus™
by Globant

training.genexus.com