

## Modules

Welcome!

My name is Silvia Keymetlian, and I work for the GeneXus Support team.

Today, I will be telling you about the Module object, a new object in GeneXus X Ev3. I will be showing you how to use it and the advantages that this object offers for understanding, maintaining and integrating a KB.



Working With Modules

Modules vs. Folders

KB Conversion

Support

We will start the presentation on modules by dividing it into four topics.

First, we will explain what the module object is, and how it is used from GeneXus, both in existing and new knowledge bases.

We will then move on to view the behavior of modules in relation to folders.

Afterwards we will see what happens when we convert a KB into a new version, and the aspects we should bear in mind regarding compatibility.

And we will end with the generators that support this functionality and which generators do not.

Let's then start by considering how to work with modules from GeneXus.

What?



We will start with an example of an ERP, like the one in the figure, where a resource is requested, and if the resource is available on stock it is delivered. Otherwise, a purchase order must be made to buy that product or the resource required.

We must then verify the existence of budget to that end, in which case the purchase is made and later recorded in the accounting.

Following purchase of the resource, it is then delivered as previously.

In this case, each of these icons here includes several objects. These objects are usually grouped to better understand the application, so that, for example, each developer focuses on a group of objects or on the functionality that is developed.

Modules will be of help here in the sense that we may define a module in GeneXus and group those objects in a GeneXus module.

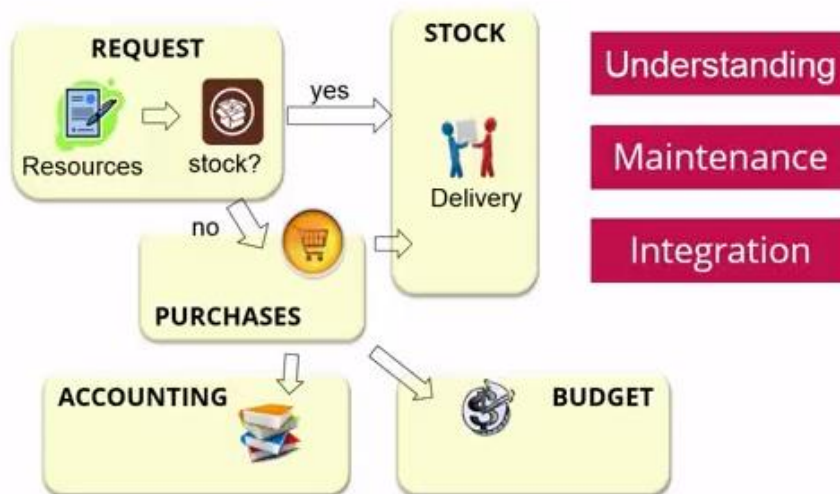
Using modules will also imply a benefit for understanding the KB, for its maintenance, and for integrating objects from different modules.

Even when a developer joins a work team, it is much easier to show him-her a functionality through objects in a module than modules not sorted or organized within modules.

Previously, we did this in GeneXus by using folders, with a specific nomenclature to indicate which objects belong to a specific functionality, but now there is a much easier way to do it.

That is why we say that using modules enables us to better understand the KB, its maintenance and its integration.

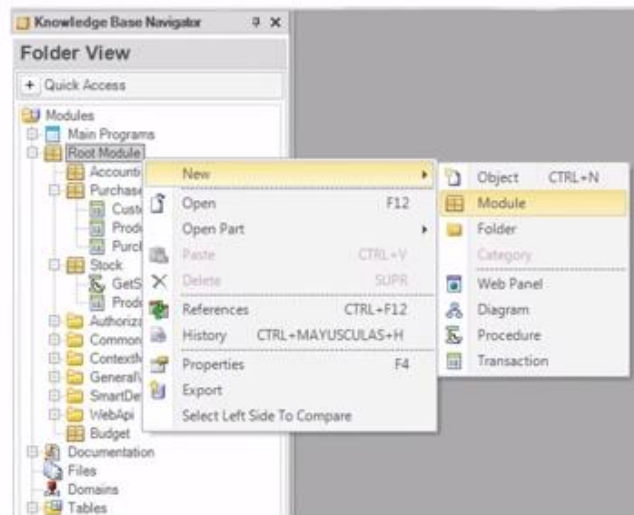
What?



So, how do we define a module in GeneXus?

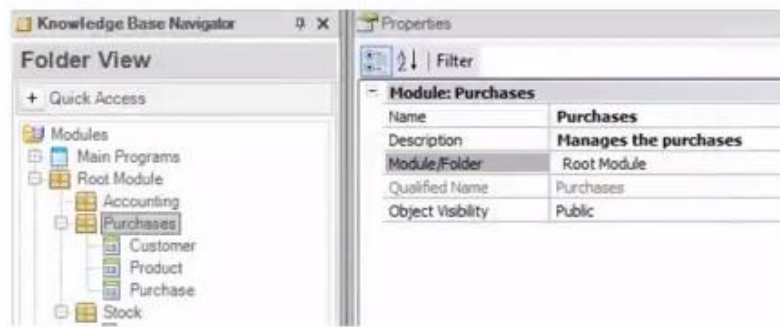
By simply clicking the right button on the Root Module, or on any other module we already have, and –as with any other new object- “new / module”.

How?



That is how we define a new module.

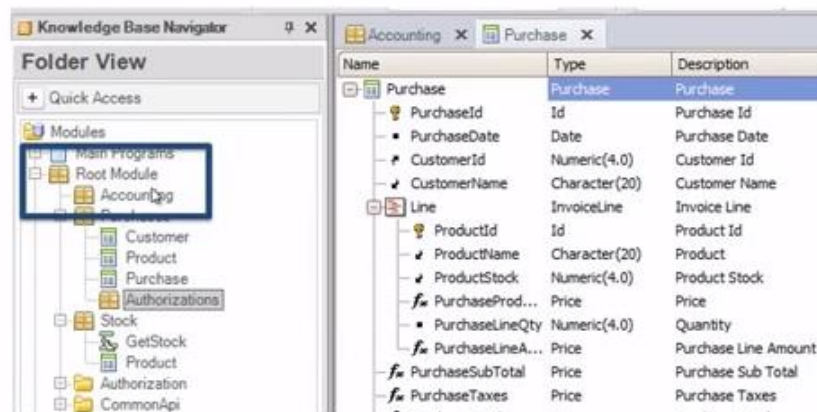
How?



Upon defining it, we are asked about properties such as name, description and other properties associated with modules.

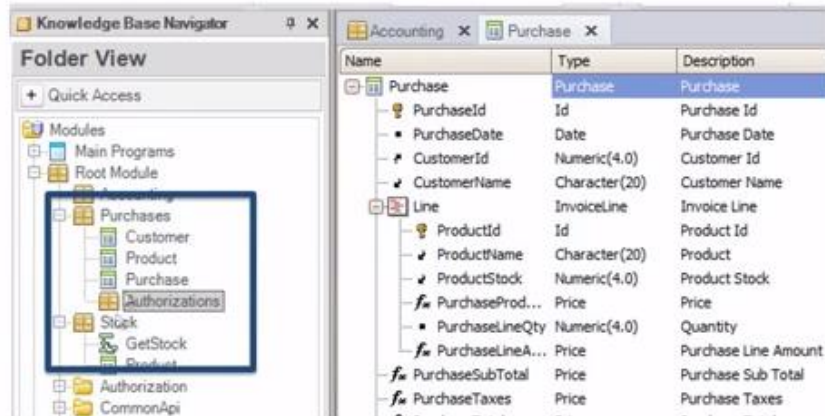
Once the module is defined, the resulting structure is similar to a structure of folders. In the Evolution 3 version, all objects belong to a module. If we do not define our own modules, the root module is always created by default for all KBs in Evolution 3.

How?



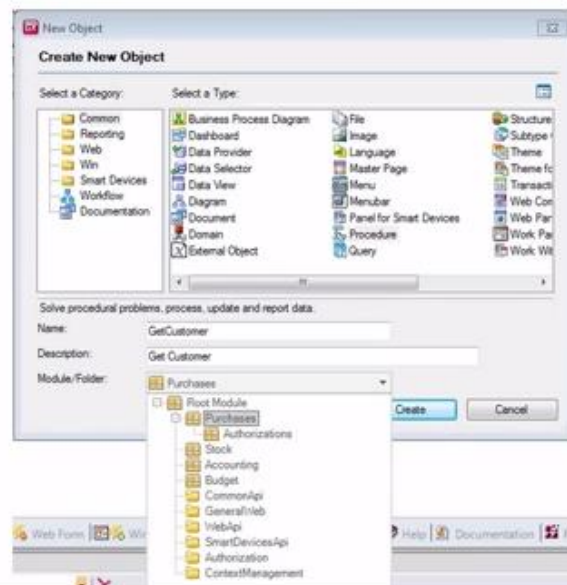
Modules may also have sub-modules. In this case, the purchase module has the Authorizations sub-module.

## How?



Also, every time that we define an object, we may indicate the module it belongs to. From the Module folder option, we indicate to which module that object will belong.

## New Object



And when we define new modules, we may include objects in them with drag and drop.

Which objects may be defined within a module?

Domains, images, language, themes, attributes, tables, files, and others, will not belong to any module because they are global objects of the whole knowledge base.

Win objects, just like work panels, menu, and menu bar, may only belong to the Root Module because Win generators do not support that functionality.

The other objects that appear in the column below...

### Which Objects can be Defined in a Module?

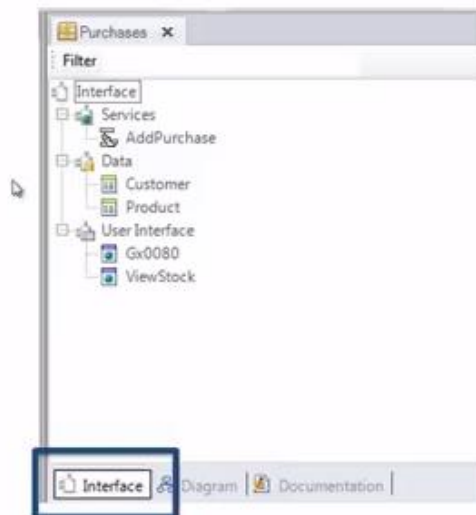
Object	Module
Domain, Image, Language, Theme, Attribute, Table, File	No
Work Panel, Menu, Menu Bar	Root Module
Folder, Transaction, Procedure, Web Panel, Panel for SD, Data Selector, SDT, Diagram, Document, External Object, Subtype Group	Yes

...may belong to any module we define in the KB.

What does a module consist of?

Having defined a module, we may open it by clicking the right button “open” on the module. There we will find three tabs:

### Interface Tab



One is called Interface, another one is Diagram, and the third one is Documentation.

The Interface tab is divided into three sections: Services, Data, and User Interface.

In the Services section we define what is known as the module’s Api, that is, the objects included in this section may be Procedure, Data Provider, and Extended Object.



The Data section includes transactions, business components, and SDTs. The User Interface section includes the master page, web components, and web panels.

Every time that an object is included in a module, it is automatically organized into one of these three sections.

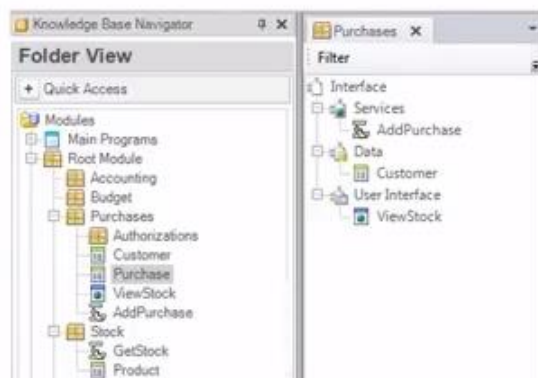
Let's then see how to determine which objects appear on the interface tab and which do not.

In the figure, we see the Purchase module we saw before.

Working With Modules

GeneXus

## Object Visibility Property



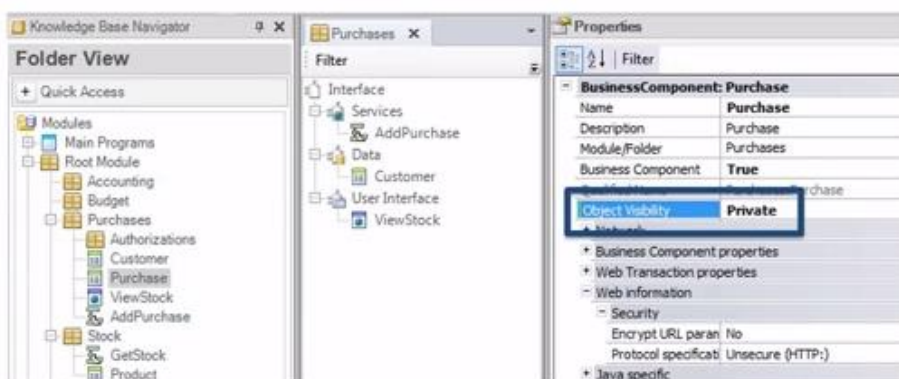
The difference here is that when the module is opened in the interface tab, the purchase transaction is now viewed, while the customer transaction is.

And why is that? It's because the Purchase transaction has the Object Visibility property with value Private. This is a new property of objects as from the introduction of modules in the KB.

Working With Modules

GeneXus

## Object Visibility Property



Objects that belong to a module have an Object Visibility property with possible values Public and Private.

Public means that the object may be used from anywhere in the knowledge base. These objects are the ones we see in the interface tab.

And Private means that it may be used only within the module, including sub-modules as well. Therefore, a sub-module may use the private objects of the parent module.

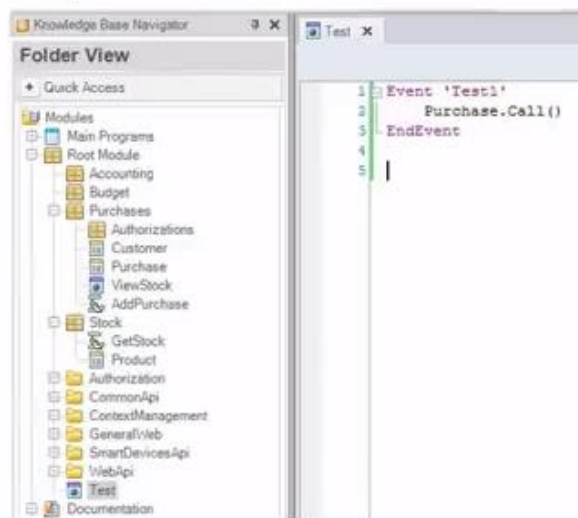
By default, all objects have the Public property, and the list of those public objects constitute the module's interface. Private objects are not viewed on the interface tab, so this object may not be used from any other object in the KB.

For example, if we want to make a call of the Purchase TRN from an object in the Root module,

Working With Modules

GeneXus

### Object Visibility Property



this case will result in a specification error.

Working With Modules

GeneXus

### Object Visibility Property



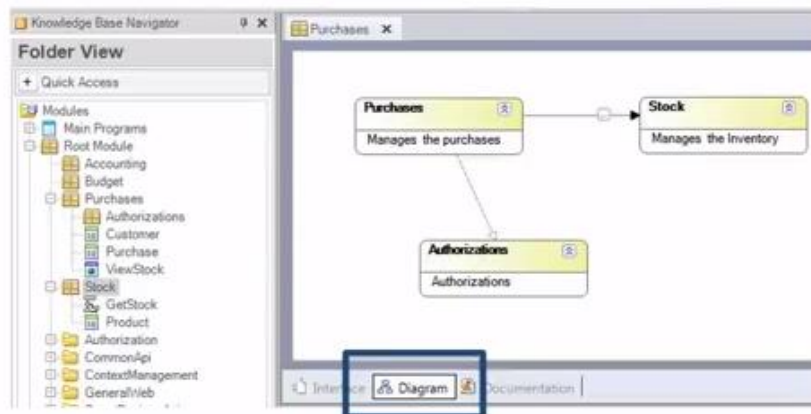
And that's because the object appears as private and may not be called from another object.



So, there is where we must define interfaces to access that object, but we will not be able to make a direct call to that object. That is how we manage which objects are viewed in the Interfaces tab and which are not.

The other tab that appears in the module object is the Diagram tab, which enables us to view how a module relates to the rest, which may be referenced or it may be a sub-module.

## Diagram Tab

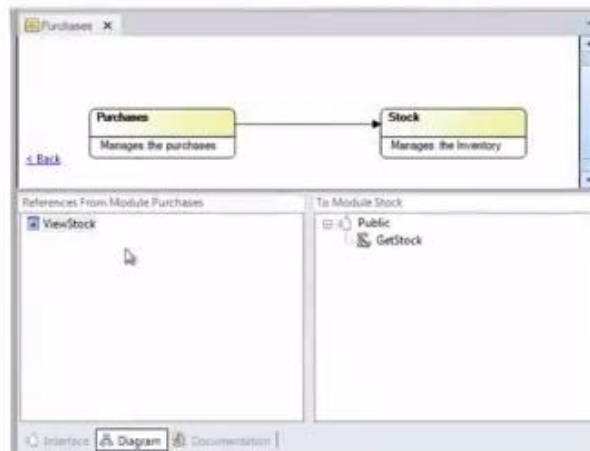


The sub-module relation appears when a module is part of another module; like in this case Authorization -part of the Purchase module- which is a module of Purchase. It is there that we show the relation with this dotted line.

On the other hand, the reference relation between modules is shown with a full line, indicating in this case that, from the Purchase module, the procedure used is located in the Stock module.

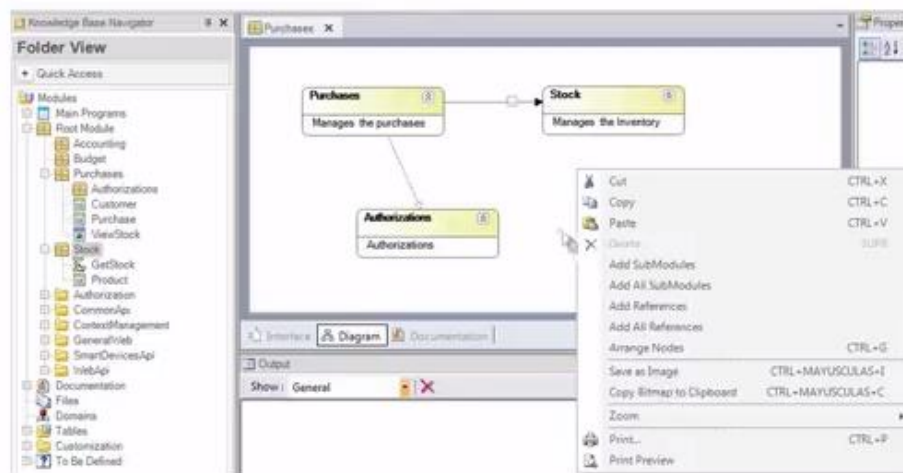
Also, we may double click to view, in the detail, which objects belong to a module and which belong to the other module.

## Diagram Tab



And with the right button clicked on the Diagram tab, we get a contextual menu with other options that enable us, for example, to add to other sub-modules or add all sub-modules to see how they relate to one another.

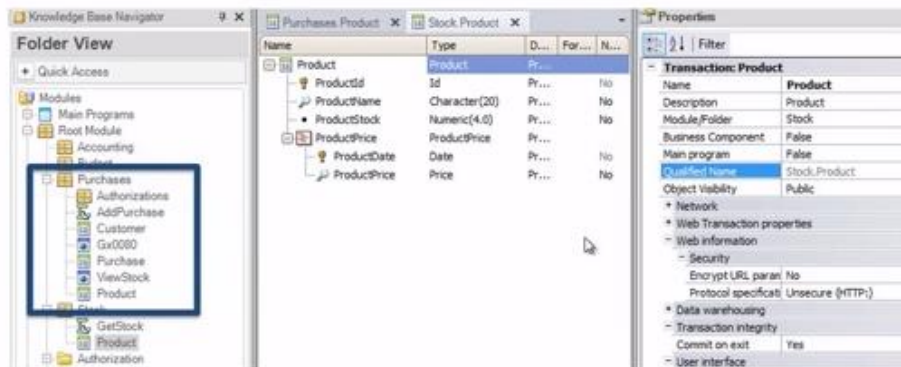
## Diagram Tab



Another aspect to bear in mind when working with modules is that we may have objects under the same name with different modules.

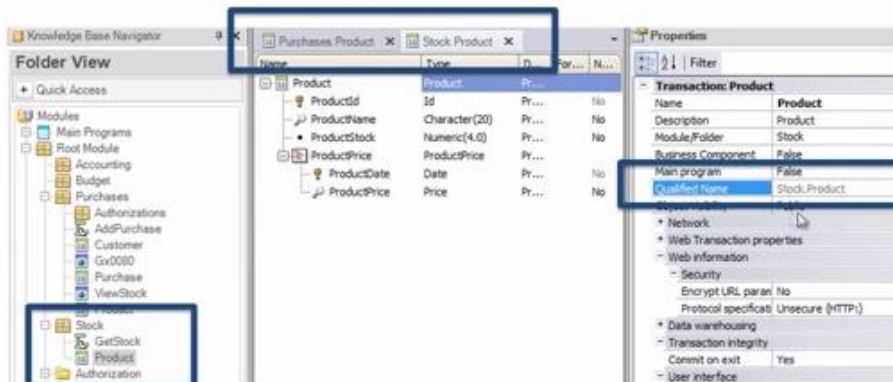
In this case, we have the Product object inside the Purchase module, and the Product object inside the Stock module.

## Qualified Name Property



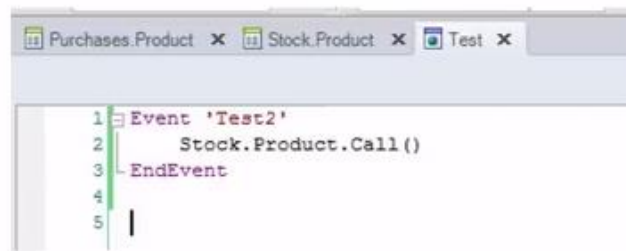
What we can do to exactly determine the object we are referring to is indicate it with the QualifiedName property, which is the name of the module (dot) name of the object.

## Qualified Name Property



This is how we can have two objects with the same name in different modules. And when we want to reference the objects, we will also have to write them down as Name of module (dot) name of object...

## Grammar



except for the case of objects in the Root module, because they do not belong to any module.

This is necessary only in the case of an ambiguity, when we have two objects with the same name, and we will not have to reference the object with the module name in front when the object is one of a kind.

We should also take into account that when we use modules in the KB, the URL of the objects in our application is changed, because when we execute an object, from now we will get the module name prior to the object name, for any generator.

## URL Syntax

### Java

.../servlet/packageName.module.object

### .NET

.../module.object.aspx

### Ruby

.../module.object.html

### SOAP Services

.../module.object?wsdl

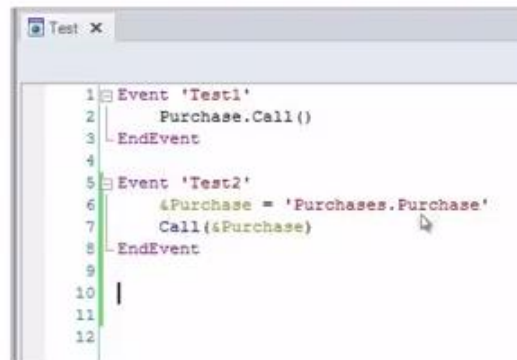
### Rest Services

.../rest/module/object/1

And specifically in the case of the JAVA generator, we have to necessarily use the packageName property to indicate the name of the package, and it is also added to the URL generated to reference that object.

As we saw before, the purchase transaction has the object visibility property with value Private, so we cannot make a direct call to that transaction, though we may make a dynamic call to that transaction.

## Dynamic Calls



```
1 Event 'Test1'
2   Purchase.Call()
3 EndEvent
4
5 Event 'Test2'
6   &Purchase = 'Purchases.Purchase'
7   Call(&Purchase)
8 EndEvent
9
10
11
12
```

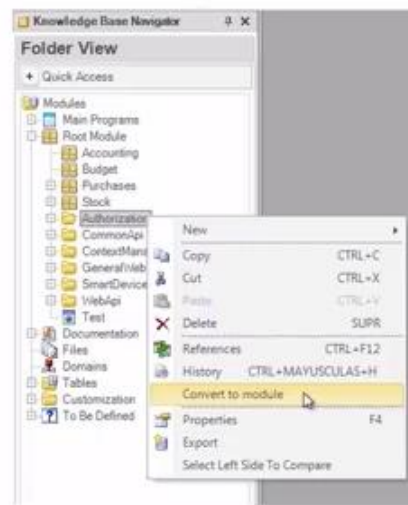
For example, in this case, in the test2 event we are invoking the Purchase transaction as a dynamic call. The only thing we must remember is that when the Expand dynamic call property is set with value Yes and this transaction has the object visibility property as Private, it will never be accessible for the caller, so it will not be included among the objects liable of being executed.

Let's now see the difference between using a module and using a folder in a GeneXus KB.

## Modules vs. Folders

For instance, something we may do is to convert a folder into a module by clicking the right button on a folder. This would be the easy way of turning the folders we were used to handling, into modules.

## Convert a Folder into a Module



So, as we have seen, both modules and folders help us in organizing objects in a KB. Altogether, they constitute a hierarchical tree where the Root Module is the root. In the dialog of folder view, we can see the tree formed. Nevertheless, we must say that conceptual differences exist between modules and folders.

	Module	Folder
Used to organize objects in a KB	✓	✓

One of those differences is that modules are part of the Qualified Name property and folders are not. That enables us to have two objects with the same name in different modules.

	Module	Folder
Used to organize objects in a KB	✓	✓
Qualified Name property	✓	✗

Additionally, modules may be parents to folders, while folders may not be parents to modules.

	Module	Folder
Used to organize objects in a KB	✓	✓
Qualified Name property	✓	✗
Can have child Folder/Module	✓	✗

So, how do we decide when to use a folder and when to use a module?

The general rule is that we may use modules to encapsulate, and folders to organize the objects within the module.

	Module	Folder
Used to organize objects in a KB	✓	✓
Qualified Name property	✓	✗
Can have child Folder/Module	✓	✗
Use for	Encapsulation	Organization within Modules

And what we must bear in mind in regards to modules when we convert a KB of versions prior to Ev3 is that:

### KB Conversion

- **Changes to the KB**
  - Root Module is created
  - All objects are assumed to belong to the Root Module
  - Dots (.) are replaced for Underscore (\_\_) in the Name property
- **Code Generation:** Not affected

The Root Module is always created by default, and it is assumed that all objects belong to that Root Module, except when we define new modules and we move those objects to the new



modules. When we have objects with (dots) in the name, these will be replaced with an Underscore (\_) in the Name property.

In regards to the code generated, if we do not define new modules in the KB, the code generated does not change. It will only make a difference when we start to introduce modules in the KB. And regarding compatibility, after we have opened the KB with Ev3, we cannot open it with previous versions because there is no compatibility in that sense.

And in the case of using GXserver, we will also need GXserver in Ev3. In Ev3, we must use the same version of GeneXus and GXserver.

GeneXus

## KB Conversion

- **Changes to the KB**
  - Root Module is created
  - All objects are assumed to belong to the Root Module
  - Dots (.) are replaced for Underscore (\_) in the Name property
- Code Generation: Not affected
- **Compatibility**
  - No backward compatibility
  - GX and GXserver Ev3

As we saw, then, the generators that support this functionality are the Java, Net, and Ruby generators, as well as the generator for Smart Devices. WIN generators do not support this functionality.

GeneXus

### Supporting Modules

- Java Web
- .NET Web
- Ruby
- Smart Devices

### Not Supporting Modules

- Java Win
- .NET Win
- Cobol
- RPG
- Visual FoxPro
- .NET Mobile

Now I would like to invite you to use GeneXus XEv3 and discover the benefits of this new functionality. If you have any doubts, you may contact me through my email address:

[silvia@genexus.com](mailto:silvia@genexus.com)

# Thank you!

More Info & documentation  
[wiki.genexus.com](http://wiki.genexus.com)

Silvia Keymetlian  
[silvia@genexus.com](mailto:silvia@genexus.com)