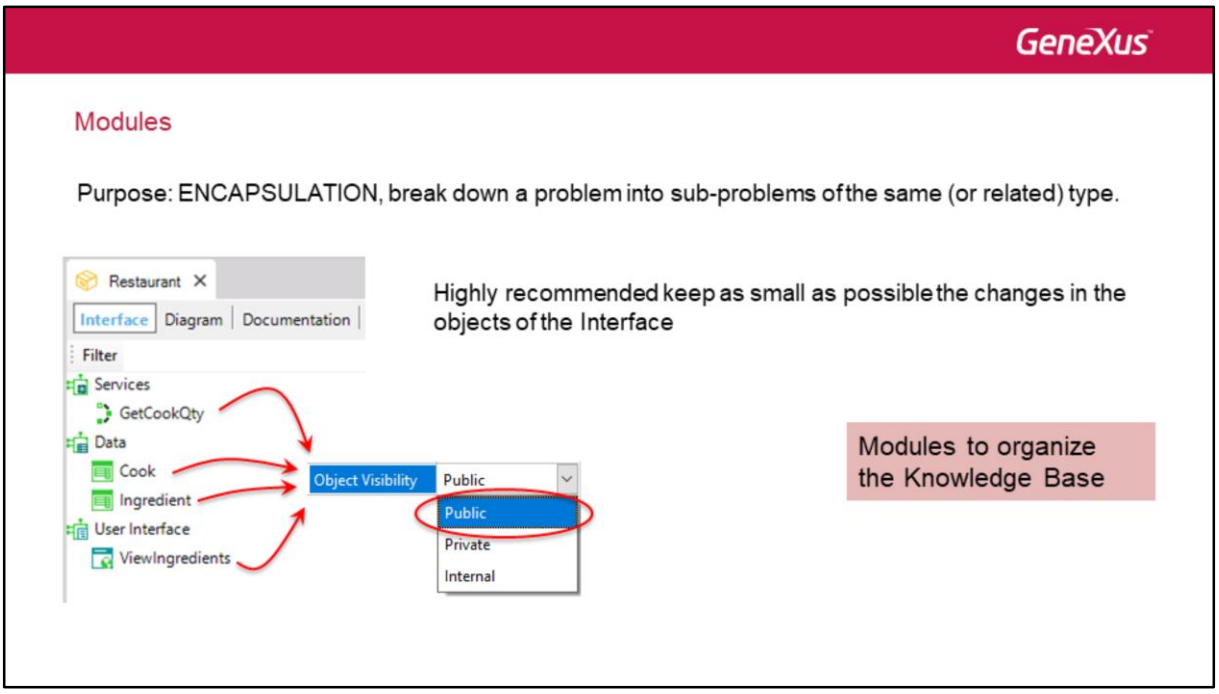
Modules

Update to GeneXus
from Evolution 3 to version 15

GeneXus 15

Review

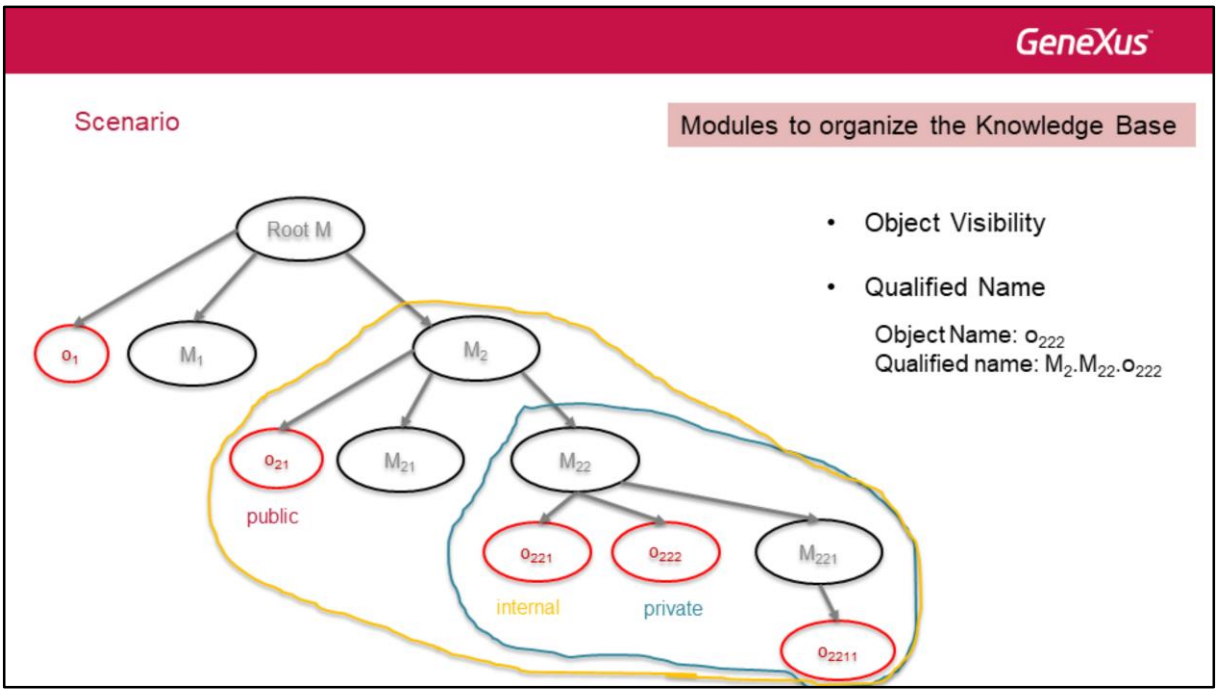


Los módulos surgieron para hacer más fácil el proceso de diseñar aplicaciones, posibilitando **organizar la KB** en diferentes secciones –objetos módulo- que se concentran en proveer una serie reducida de servicios especializados. Todas esas secciones cooperan para proveer un servicio más complejo. Esto permite que los desarrolladores se concentren en desarrollar funcionalidades sin la necesidad de saber sobre el resto de la aplicación.

Por tanto, el objeto módulo permite partir el problema en sub-problemas del mismo (relativo) tipo. Estos sub-problemas son suficientemente simples para permitir una solución directa, que es luego combinada con las soluciones de los otros sub-problemas utilizando las Interfaces definidas para encontrar una solución para el problema original.

Se recomienda al diseñar la interfaz de un módulo concentrarse en sus objetos para intentar que no cambien demasiado para no afectar a otros módulos que los utilizan.

La interfaz es definida utilizando la propiedad Object Visibility de cada objeto del módulo. Sólo los objetos públicos serán parte de la interfaz.



Recordemos que un objeto **private** sólo puede ser accedido por objetos del mismo módulo o por objetos de módulos hijos de éste, pero no por objetos que estén “más arriba” en la jerarquía o desciendan de módulos que están más arriba.

En el ejemplo, los únicos objetos que pueden invocar al objeto o_{222} privado son el o_{221} y el o_{2211}.

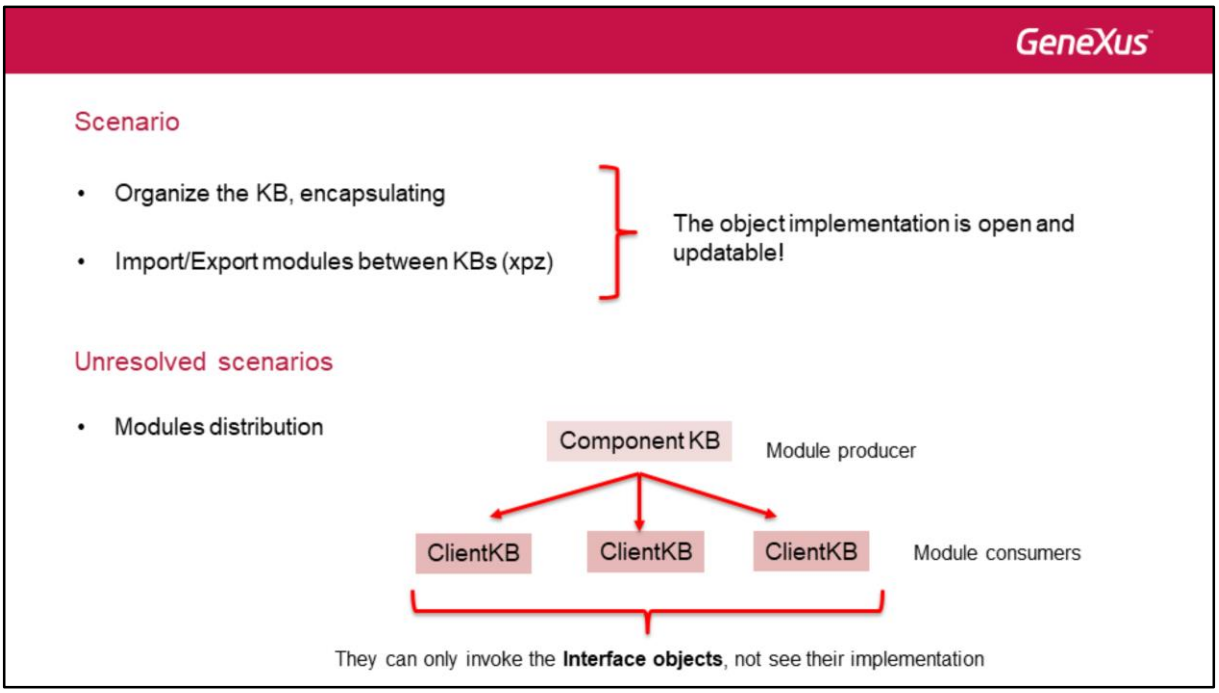
Los objetos **internos** se agregaron para que puedan ser utilizados entre objetos de una rama del Root Module. Es decir, un objeto que tenga la propiedad Object Visitility en **Internal** podrá ser accedido por todo objeto con el que comparta un módulo raíz que no sea el Root Module sino algún descendiente de éste.

En el ejemplo, o_{21} puede acceder (y no porque sea público, en esto lo mismo da su visibilidad) al objeto o_{221}. Si o_{221} fuera private, no podría.

A los objetos públicos puede acceder cualquier objeto de la KB. Recordemos que los objetos públicos son los que compondrán su interfaz. Por tanto, si bien o_{221} no será visible en la interfaz del módulo M_{22}, de todas maneras un desarrollador que esté trabajando en un objeto del módulo M_2 o de M_{21} podrá invocar al objeto o_{221}.

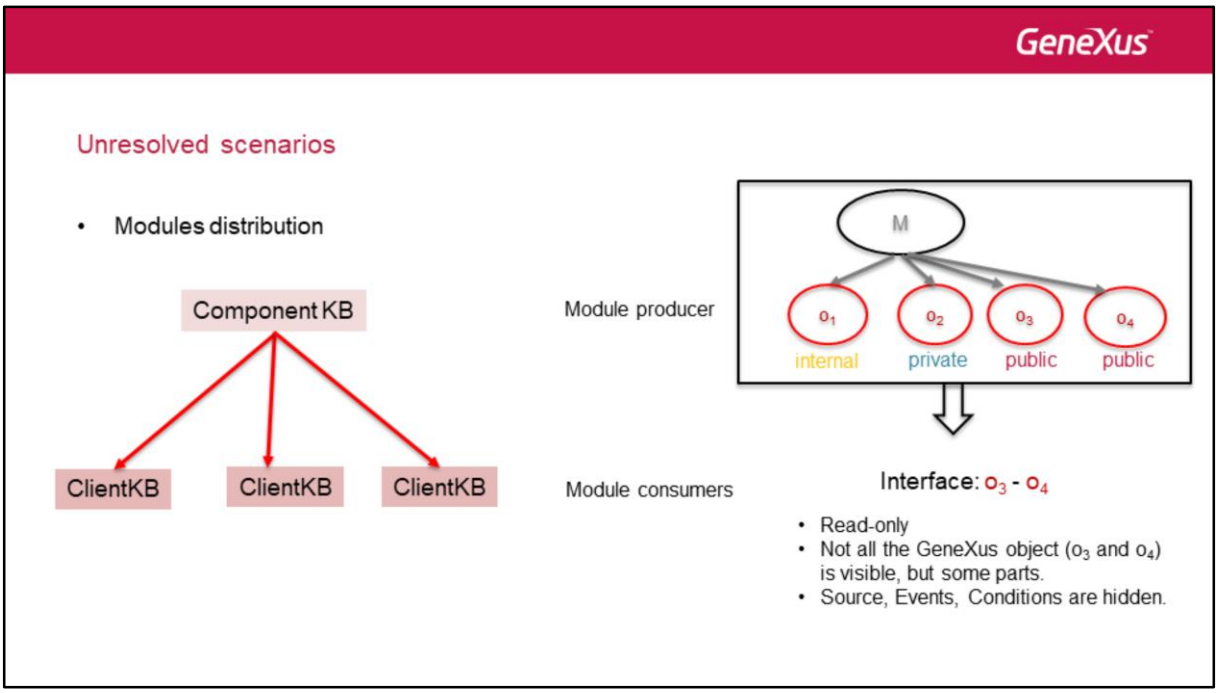
Por otro lado recordemos que en distintos módulos podemos repetir el nombre de objeto, dado que a partir de que los objetos se salvan dentro de un módulo, su nombre queda calificado automáticamente con el path. Los objetos del módulo Root son los únicos para los que el Qualified name es el propio name.

Esto provoca un problema cuando se trata de transacciones paralelas en distintos módulos, ya que si bien las transacciones pueden pertenecer a módulos diferentes, las tablas son únicas, así como los atributos. Dejamos este tema para que lo repase en nuestro community wiki.

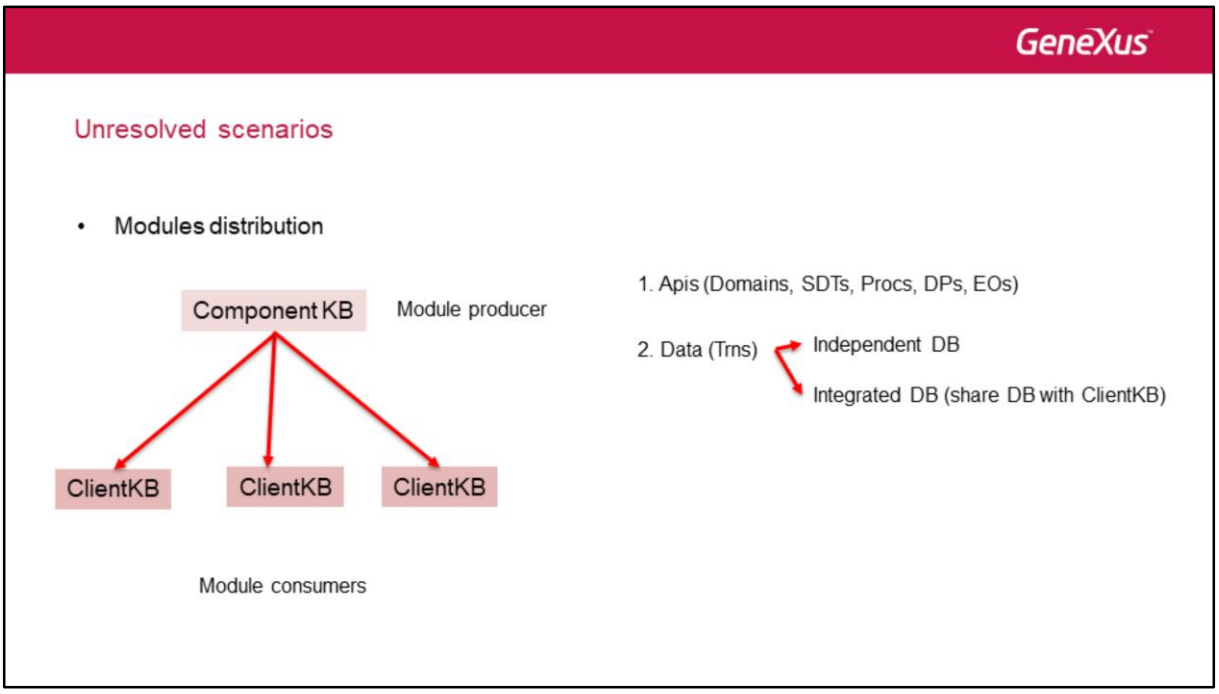


El escenario que teníamos hasta GeneXus 15 era uno donde a lo sumo podíamos exportar un módulo como xpz e importarlo en otra KB. Y el comportamiento del módulo iba a ser exactamente el mismo. De hecho, todos sus objetos pueden ser abiertos por cualquier desarrollador de la KB destino y acutualizados. La Object Visibility solamente aplica a la hora de que un objeto invoque a otro.

No teníamos manera de distribuir módulos de forma tal que funcionen en cierto sentido como apis, donde el cliente no pueda ver la implementación de los objetos del módulo sino que solamente pueda utilizarlo a través de su interfaz, invocando a los servicios y pantallas definidas (sin poder ver su código). Y sin poder enterarse siquiera de la existencia de los objetos privados o internos del módulo.



No teníamos resuelto este escenario donde queremos que una KB desarrolle el módulo y otras puedan consumirlo, es decir, utilizar lo que el módulo provee, pero sin poder tocar nada de allí, ni siquiera ver la implementación (salvo la de las partes necesarias para poder utilizar el objeto, como la regla parm, el form y las variables.



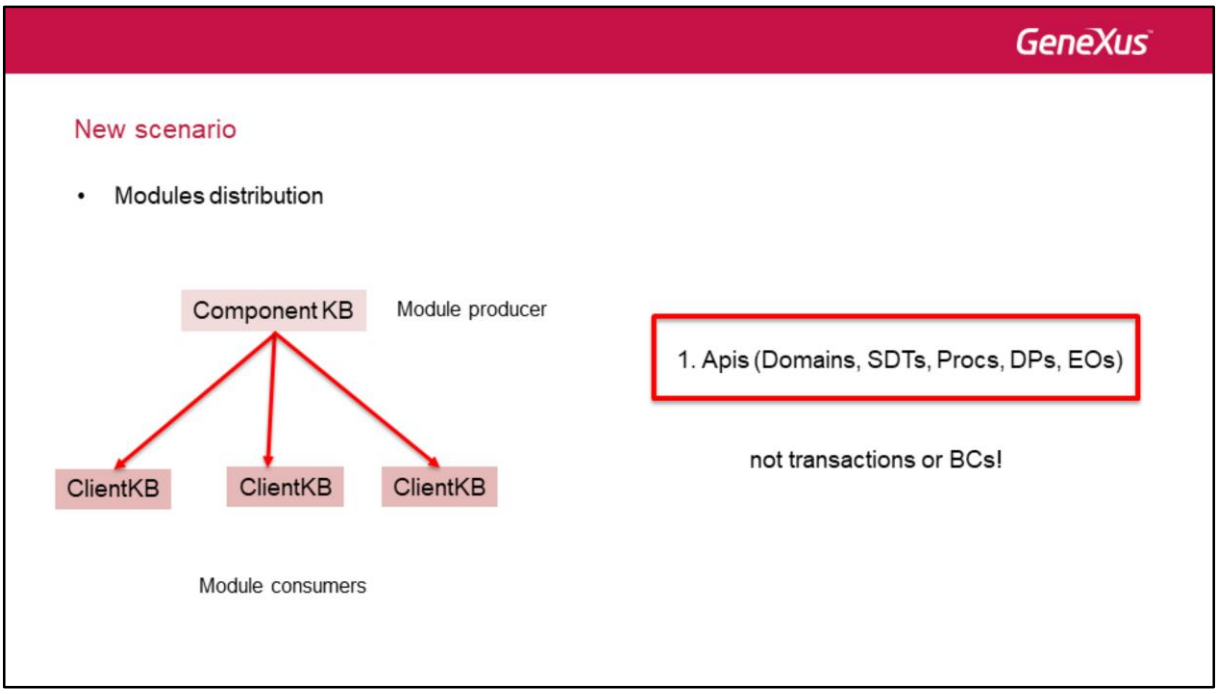
Este escenario a su vez se abre en tres:

1. Módulos que implementan apis que no requieren de acceso a base de datos (no contienen transacciones). Este es el escenario que se ha resuelto en GeneXus 15 y que abordaremos en lo que sigue.

2. Módulos que sí contienen acceso a base de datos. Allí se nos abren a su vez dos escenarios: módulos cuyas Trns deben trabajar sobre bases de datos independientes de las KBs que vayan a consumir esos módulos, y módulos cuyas Trns deben integrar la misma Base de Datos que la KB que consume el módulo. Estos escenarios siguen en investigación, por lo que serán resueltos en próximas versiones de GeneXus.

GeneXus

Modules in GeneXus 15



Decíamos que GeneXus 15 resuelve el escenario en el que el módulo implementa una o varias apis. Eso significa que provee servicios para resolver algunas funcionalidades, pero que no requieren de datos de tablas propias.

Es decir, los únicos objetos del módulo serán Dominios, SDTs, Procs, DPs y EOs. Podrían haber paneles. Pero no transacciones o business components!

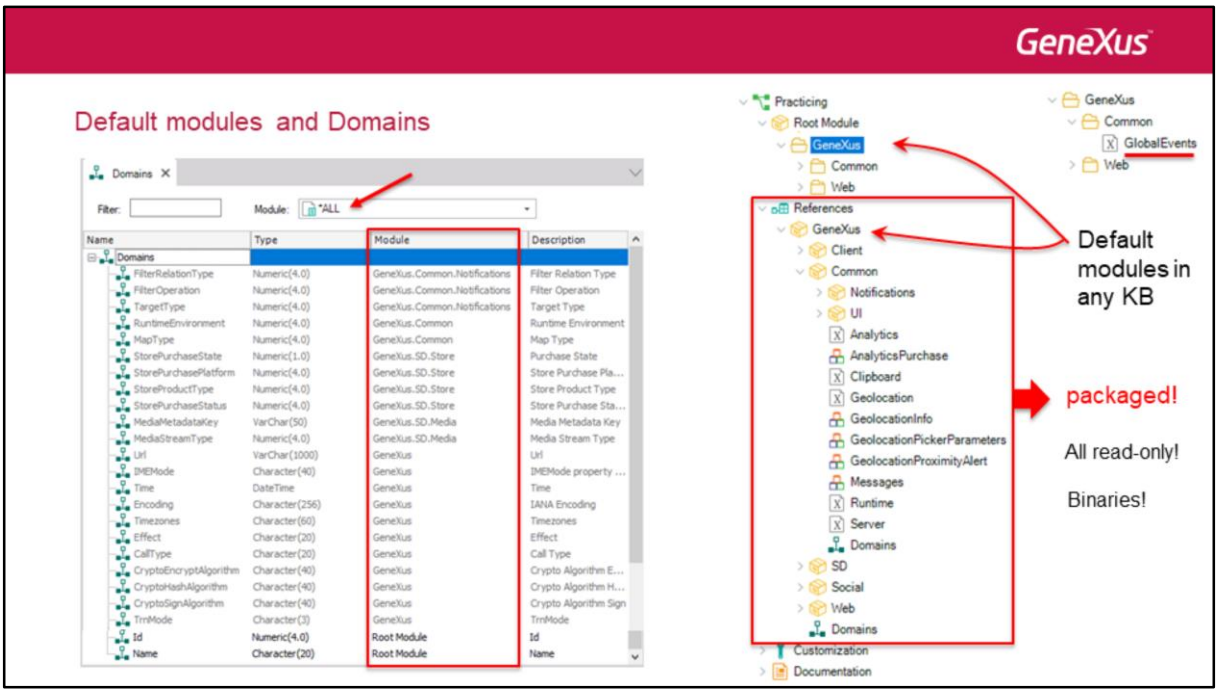
GeneXus

Which objects can be defined in a module?

Object	Module
Image, Language, Theme, Table, Attributes, Files	... Are global to de KB
Folder, Transaction, Procedure, Data Provider, Data selector, Web Panel, Panel for Smart Devices, Work with for Smart Devices, **SDT**, **Domain**, Diagram, Document, **External Object**, Subtype Group.	... Will belong to the module where they are defined and can be defined in any Module in the KB

Obsérvese que los SDTs, External Objects y dominios antes eran globales y ya no. Por esa razón tendremos un nodo Domains en el KB Explorer por cada módulo.

Todavía no mencionamos qué es ese módulo GeneXus bajo el nodo References. Ya lo veremos.



En toda KB creada con GeneXus 15 habrá dos módulos default:

- el módulo Root, bajo el que se encuentra el folder GeneXus.
- El módulo GeneXus bajo el nodo **References**.

Este nodo References es en el que se van a encontrar todos los módulos que en esta KB se vayan instalando para ser consumidos. Si observamos, ya existe aquí un módulo con varios submódulos. Este módulo, conocido internamente como GeneXus core, contiene todas aquellas apis (objetos externos y sus SDTs) que en la versión previa se encontraban bajo el nodo Root de la KB.

¿Cuál es la gran diferencia? Que este módulo (y sus sub-módulos) no está abierto sino empaquetado. Esto significa que no pueden editarse los objetos dentro de él como hacíamos antes. Todo lo que puede verse puede utilizarse, pero ¡es read-only! Y no solo eso: lo que aquí se muestra de los objetos es solo su interfaz. En verdad lo que aquí se importan ¡son los binarios! De esta manera se ahorra una considerable cantidad de tiempo de spec, generación y ejecución.

Este es el primer caso de un modulo que es producido por una KB (del equipo de desarrollo de GeneXus) y distribuido automáticamente (instalado) a todas las KBs que utilicen GeneXus 15.

Esto significa que si usted ha editado algo que habíamos distribuido en versions previas de GeneXus, ahora será sobreescrito. Recomendamos que cree sus propios dominios o SDTs dado que ya no será posible personalizar aquellos distribuidos por GeneXus.

Este modulo GeneXus nos permite saber que cuando un cliente está trabajando con una version específica de GeneXus tiene la version correcta del módulo y sus objetos (APIs, SDTs, Domains).

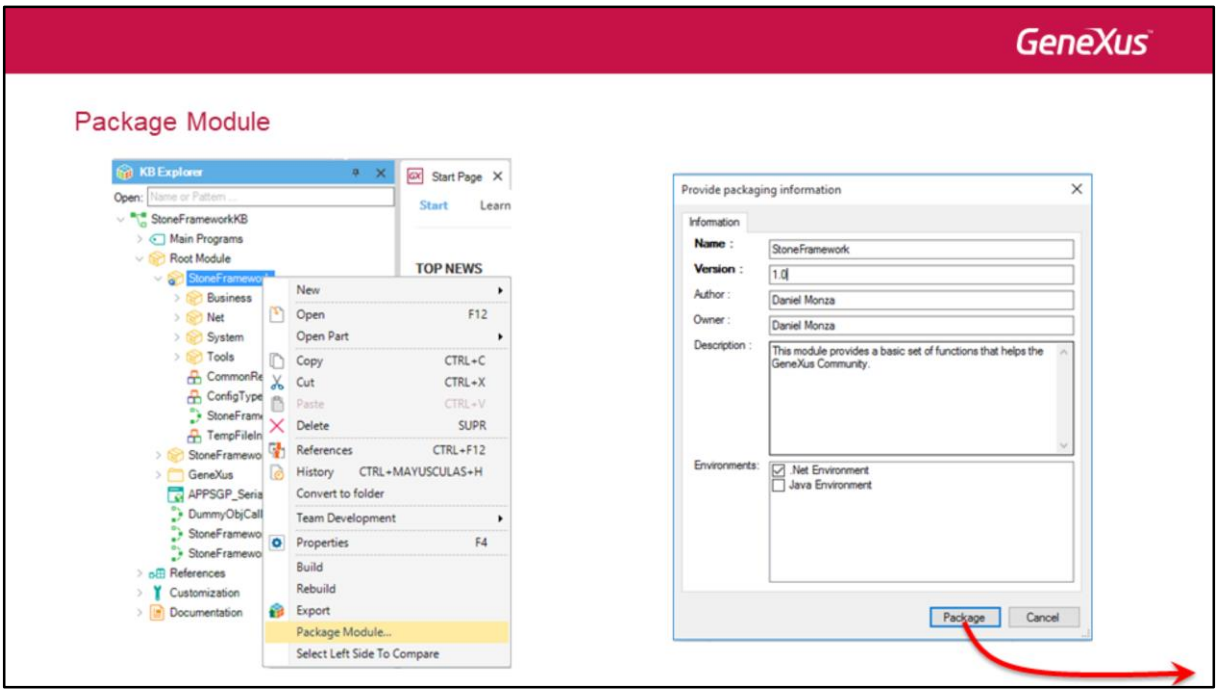
En el editor de dominios hay ahora una nueva columna que muestra a qué modulo pertenece cada dominio listado. Los dominios que pertenecen a aquellos módulos distribuidos directamente por GeneXus aparecen como read-only, y lo mismo sucederá con dominios que provengan de módulos que los desarrolladores instalen en la KB.

¿Por qué el folder GeneXus del Root Module contiene un folder Common con un external object GlobalEvents? ¿Por qué ese EO no está también en el módulo GeneXus, empaquetado, como los demás? La respuesta es simple: si bien es un objeto predefinido, dentro de su función está el que

el desarrollador lo modifique. No puede ser read-only.

Lo mismo sucederá con algunos dominios y EO del GAM.

Como decíamos antes, a partir de GeneXus 15 podemos empaquetar nuestros propios módulos para distribuirlos a otras KBs que los consuman.

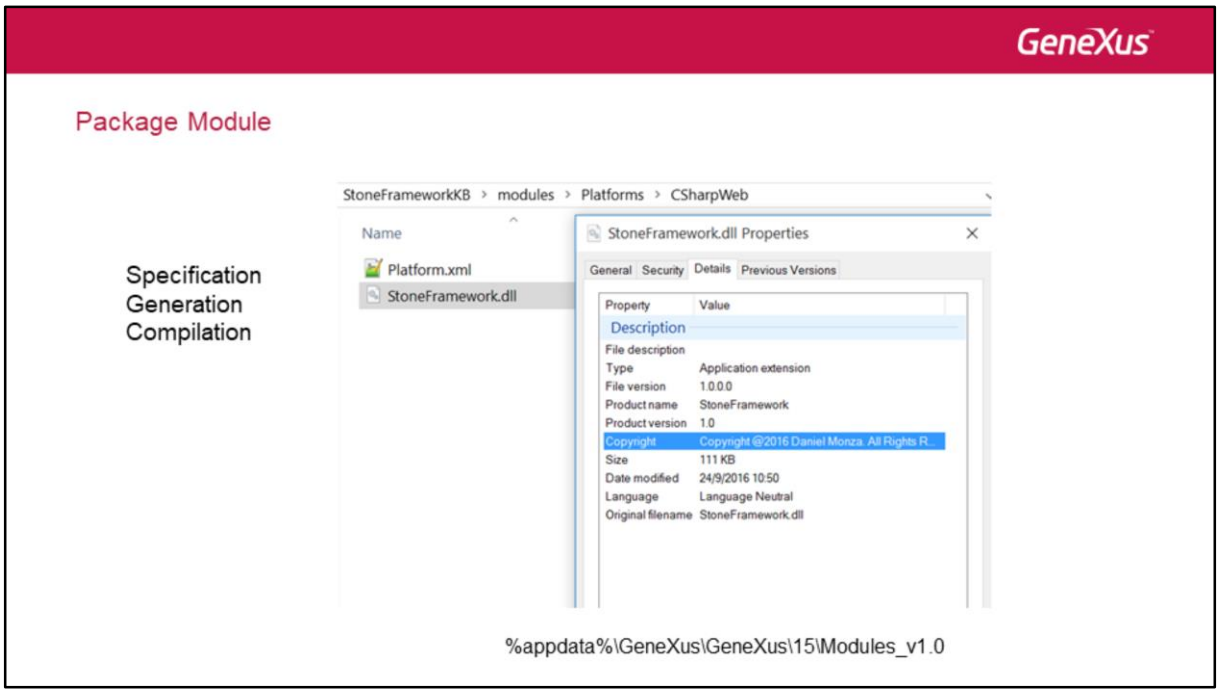


Supongamos que en nuestra KB, bajo Root Module hemos desarrollado un módulo StoneFramework con objetos y submódulos que implementan algunas funcionalidades básicas como una función que transforma una cantidad de dinero en una moneda determinada, en texto, de acuerdo al idioma que se elija.

Queremos hacer un paquete con ese módulo para distribuirlo a otras KBs, que no puedan modificar esos objetos, ni ver su implementación completa, sino que sólo puedan usarlos. Para ello hacemos botón derecho sobre el módulo, y vemos que aparece la opción Package Module... que nos abre una ventana para proveer la información que vemos. El desarrollador debe configurar, entre toda la info, en cuáles de los environments de la KB va a querer que los objetos se generen. ¿Por qué? Porque lo que se va a distribuir no son los objetos GeneXus, sino la interfaz y ¡los binarios ejecutables!

En el ejemplo estamos indicando que queremos se genere el módulo en .NET únicamente. Observe que la KB podría tener también environment Android e iOS.

Al presionar Package...

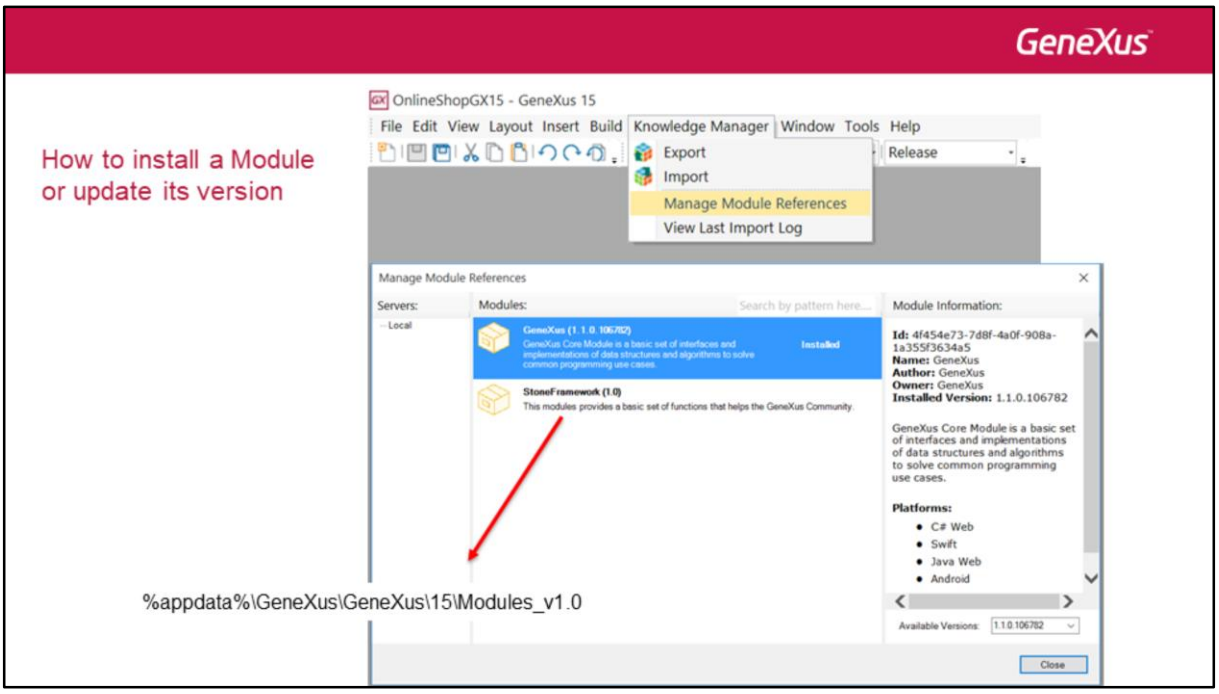


GeneXus especifica, genera y compila todos los objetos dentro del modulo (hace rebuild) y una vez que el proceso culmina, crea en este caso una dll (pues le pedimos que armara el paquete solo para .Net; si hubiéramos seleccionado además Java habría generado un .class también).

El paquete contiene los binarios (e.g. dll) y los archivos de definición del modulo. Lo genera con extensión ocp que significa Open Packaging Convention. Es un zip. En este caso en: "<Knowledge Base Directory>\modules\StoneFramework_1.0.opc“

El binario es creado proveyendo version e información de copyright.

Ya tenemos listo el paquete para ser instalado en cualquier otra KB. Por ahora, para que el módulo pueda ser instalado debe ser copiado en un lugar específico del disco (en el AppData del usuario). Si no se copia allí no será visto automáticamente por el Module Manager. Lo vemos en el siguiente paso.

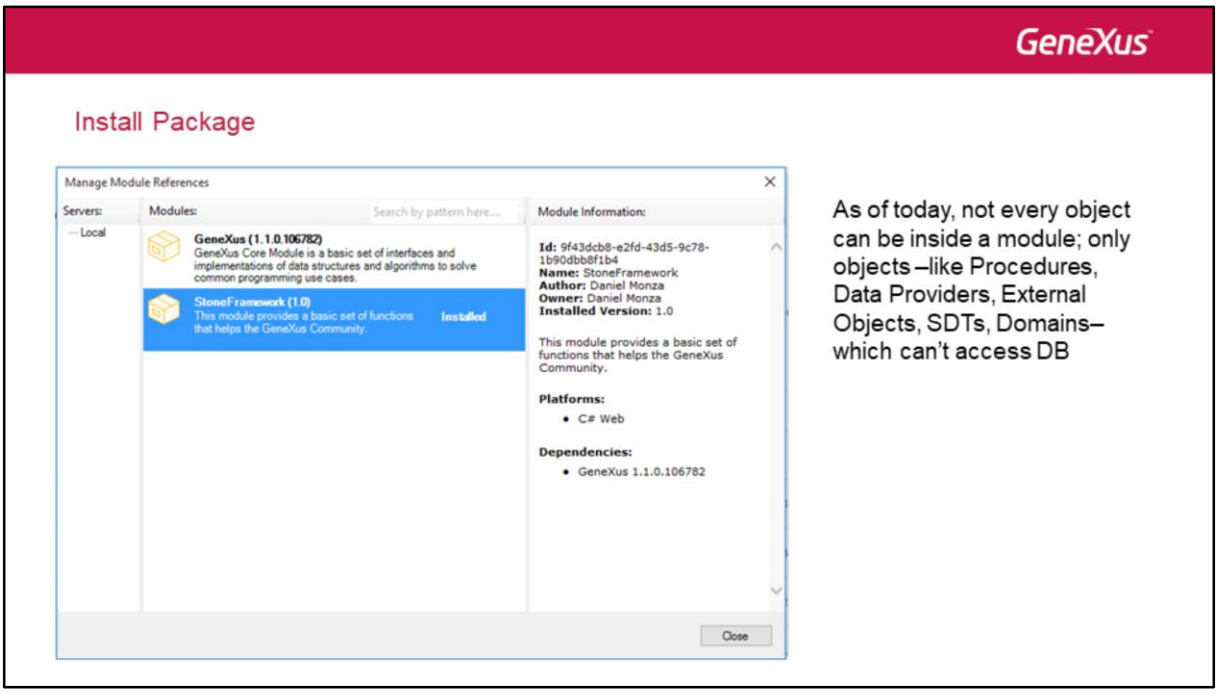


Ahora bajo la opción Knowledge Manager del menú de GeneXus podemos ver la opción Manage Module References.

Aquí veremos todos los módulos que la KB tiene instalados (siempre habrá por defecto uno: el modulo GeneXus) y los que tiene a disposición para instalar.
Por ahora son encontrables para instalar los paquetes que se encuentren en el directorio AppData del usuario. Estamos trabajando en una opción para compartir módulos, ya sea a través del marketplace o de una red de la empresa.

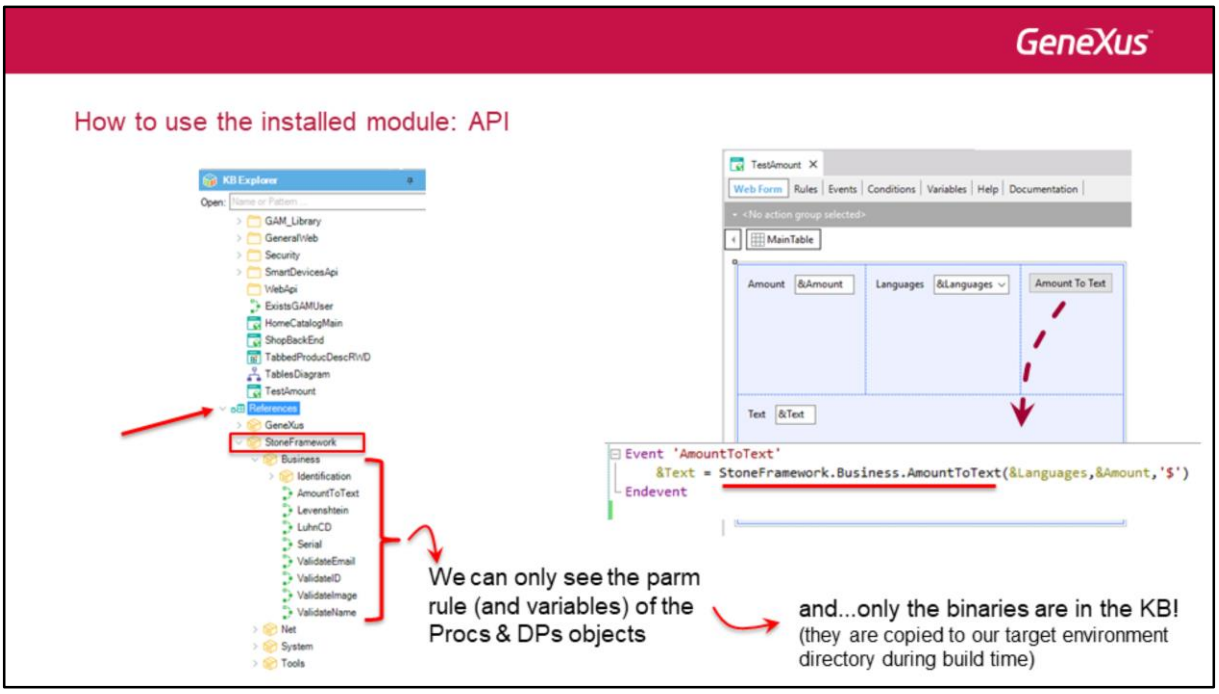
Para módulos ya instalados, como el GeneXus, nos ofrecerá actualizar la versión si quedamos desactualizados. De hecho podemos ver en el combo de la derecha las versiones disponibles del módulo.

Algunas veces puede pasar que no se vea en el Manager un modulo correspondiente a un paquete que tenemos correctamente copiado para instalar. Esto se resuelve ejecutando **GeneXus.exe/install** que automáticamente lo copia al lugar correcto y ahora sí el modulo aparecerá en la ventana del manager.



Como decíamos al principio, solo podemos resolver un escenario actualmente y es aquel en el que los objetos del módulo no acceden a la base de datos. Esto era porque teníamos que resolver el siguiente conflicto: tengo una DB en mi app, pero traigo objetos de un módulo de otra KB, por lo que no sabemos a qué DB acceden esos objetos del módulo. Tenemos que resolver cómo manejar este tipo de conexión y si el módulo puede acceder a otra DB o si tiene que ser la misma.

El ejemplo que veníamos presentando era el de un modulo con procedimientos que resuelven un requerimiento simple, como validar una imagen o email, o enviar una cantidad o texto (que son herramietnas que usamos muchas veces en nuestras KBs y que no tienen necesidad de acceder a la DB).



Al importar un módulo quedará bajo el nodo **References** del KB Explorer. La pregunta ahora es ¿cómo podemos utilizar los objetos de su interfaz (api)?

Como sabemos, al igual que los objetos del módulo GeneXus, estos objetos serán read-only. Sólo debemos saber cómo invocar a los objetos y qué es lo que cada objeto hace. Es decir, solamente tendremos una API disponible de esos objetos y lo único que veremos será su regla parm, dado que debemos conocer qué parámetros utilizar al invocar los objetos, y sus tipos de datos. En el módulo también podrán venir Eos, SDTs y Domains que serán visibles, claro.

En el ejemplo tenemos un procedimiento del modulo, AmountToText, que devuelve un texto formateado de acuerdo a los valores que le son enviados por parámetro. En la KB que instaló el módulo se define un web panel con tres variables en pantalla y un botón. El evento de este botón invoca a la API del proc del modulo StoneFramework.Business. Invocar a una API es lo mismo que invocar a cualquier objeto GeneXus. Podríamos no haber incluido el qualified name del objeto AmountToText si sabemos que AmountToText es único.

Cuando ejecutemos, GeneXus tomará al proc AmountToText como un objeto normal en la KB en lo que hace a las invocaciones (solo que AmountToText no necesitará ser generado, pues tenemos los binarios)

GeneXus™

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certificactions	training.genexus.com/certifications