

# Curso GeneXus Core

## Consideraciones para Angular

Versión: GeneXus 18

# Consideraciones para aplicaciones Angular

Recomendamos realizar el [curso GeneXus Core](#) tal cual está armado, es decir, con el enfoque para Front-end Web (.Net).

No obstante, para aquellos que quieran ir mapeando con el desarrollo Web Angular a medida que van viendo los videos, aquí dejamos las consideraciones pertinentes para cada video, pero con la advertencia de que tal vez esto pueda confundirte y sea mejor que esperes a terminar todo el curso tal cual está.

Este material es complementario, no tiene ningún efecto sobre el examen del Curso Core ni sobre ninguna otra parte del curso.

## Contenido

Primeros pasos.....	2
Transacciones .....	3
Listados y acceso a los datos por código.....	5
Comunicación entre objetos.....	8
Tipos de datos estructurados y Data Providers.....	9
Actualización de la Base de Datos.....	10
Arquitectura .....	15
Pantallas Web con foco en Back-office .....	15
Diseño y modelado de pantallas .....	15

## Primeros pasos

### [“Creación de la base de conocimiento”](#)

Si se va a querer construir también un front-end en Angular, en la ventana de creación seleccionar el check box Web(Angular) -además del .Net que viene por default-.

De no hacerlo en este momento de creación de la KB, de todos modos se lo podrá hacer más adelante.

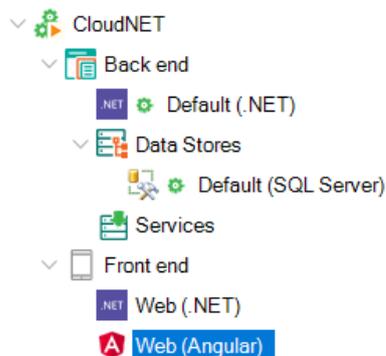
## Transacciones

### [“Diseño de la primera transacción”](#)

El objeto transacción tiene una parte que es front-end (es decir, que se ejecuta en el cliente, como su pantalla) y una parte que es back-end (lo que se ejecutará en el servidor, como el acceso a la base de datos, por ejemplo). El objeto transacción sólo se ejecuta integralmente (front-end y back-end) cuando se trata del front-end Web (.Net, Java).

El front-end “Web (Angular)” no ejecuta el objeto transacción, pero sí podrá utilizar la parte back-end de ese objeto. Se comprenderá más adelante en el curso, cuando se estudien los Business Components, que podrán funcionar como servicios expuestos en el servidor. También soportará las pantallas del patrón Work With y por allí será que se podrá implementar un Backoffice con Angular.

Como es a partir de las transacciones que GeneXus crea automáticamente la base de datos de la aplicación, aunque estemos desarrollando una aplicación para Angular la parte del back-end será implementada en alguno de los lenguajes estándar (.NET, Java). Es por ello que veremos separado lo que es back-end de lo que es front-end así:



Dicho de otro modo: cuando se desarrolla para Angular (tanto Back-office como Customer-facing) necesariamente se requiere utilizar .NET/Java para el back-end (sin tener por qué utilizarse el front-end de ese generador).

Por todo lo dicho, para cargar datos en la transacción Customer necesitaremos tener un Backoffice .NET/Java (como el del video del curso) -y es por ello que vemos el “Front end Web (.NET)- o... usar el patrón Work With que se estudiará varios videos más adelante.

### [“Ejecución de la aplicación por primera vez”](#)

Sólo vale para el Front-end web en Java/.NET, como se explica al final del video.

[“Atributos y dominios”](#), [“Transacciones relacionadas”](#), [“Transacciones con más de un nivel”](#), [“Nomenclatura de atributos”](#), [“Definición de reglas”](#).

Todo esto es importante entenderlo porque hace a la normalización de la base de datos, a la integridad referencial, a las reglas que se aplicarán sobre los datos a ser ingresados y a los datos en sí mismos y sus tablas.

#### [“Uso de patrones”](#)

El patrón para aplicaciones Angular no es el que se muestra en este video (el “Work With for Web”), sino el “Work With” general (el que en versiones anteriores de GeneXus se denominaba “Work With for Smart Devices”), que vale también para aplicaciones móviles nativas.

Adjuntamos [este video](#) específico para Angular, para que lo visualices después de este.

Si le aplicas el patrón “Work With” a Attraction, para que puedas ejecutarlo en Angular deberás acceder a las propiedades del objeto **WorkWithAttraction** creado por el patrón y colocar valor True a la propiedad “Main program”. A continuación, posicionado sobre ese objeto, presiona botón derecho del mouse y selecciona Run.

El equipo de desarrollo del generador Angular aún no completa su desarrollo, por lo que lo que verás es más simple que el Work With de las aplicaciones Web (.NET/Java).

Mientras que para Web (.NET/Java) no existe como objeto ejecutable, para Angular y aplicaciones móviles sí.

- En Web (.NET/Java) se configura todo a partir del árbol que se muestra en el video.
- En Web Angular se programará todo como si se tratase de un objeto independiente.

Esto hace que la forma de implementar las pantallas del Work With en un caso y otro difieran.

Observarás que una vez que se le aplica el pattern “Work With” al objeto transacción (en este caso, Attraction) automáticamente se le prende la propiedad **Business Component**. Más adelante entenderás por qué, ahora sólo obsérvalo.

[“Tabla base y extendida”](#), [“Definición de subtipos”](#)

Valen exactamente.

#### [“Definición de atributos como fórmulas”](#)

Si bien podríamos pensar que esto no valdrá para Angular dado que allí no se genera objeto transacción (sino que ésta se ejecutará en modo silencioso como Business Component), y por tanto nada de lo que se muestra en ejecución en este video podrá verse en Angular, en verdad la definición de atributos fórmula aplica a nivel de la estructura de la transacción, lo que significa que aplica a nivel del modelo de datos. Por tanto las fórmulas sí valdrán en todas las

plataformas. La diferencia vendrá dada por el momento en que se calcularán. Pero la lógica es exactamente la aquí presentada, en todas las plataformas de desarrollo. De hecho cuando más adelante veamos cómo recorrer una tabla y listar su información, veremos cómo allí la fórmula se disparará automáticamente para cada registro sin que debamos hacer nada. La definición de un atributo fórmula es una definición lógica que tiene repercusiones en la base de datos y en toda la KB.

[“Eventos de disparo de reglas en transacciones”](#)

Si bien puede parecer a primera vista que esto no aplicará a Angular, en verdad las reglas se ejecutarán también cuando la transacción se ejecute sin su pantalla, es decir, cuando se manipule su información a través del Business Component.

[“Índices”](#), [“Normalización de Tablas: Un Caso de Estudio”](#), [“Relaciones entre actores de la realidad”](#), [“Relaciones 1 a 1 entre actores de la realidad”](#), [“Exportar e importar objetos GeneXus”](#), [“Análisis del modelo de diseño de transacciones”](#).

Valen exactamente

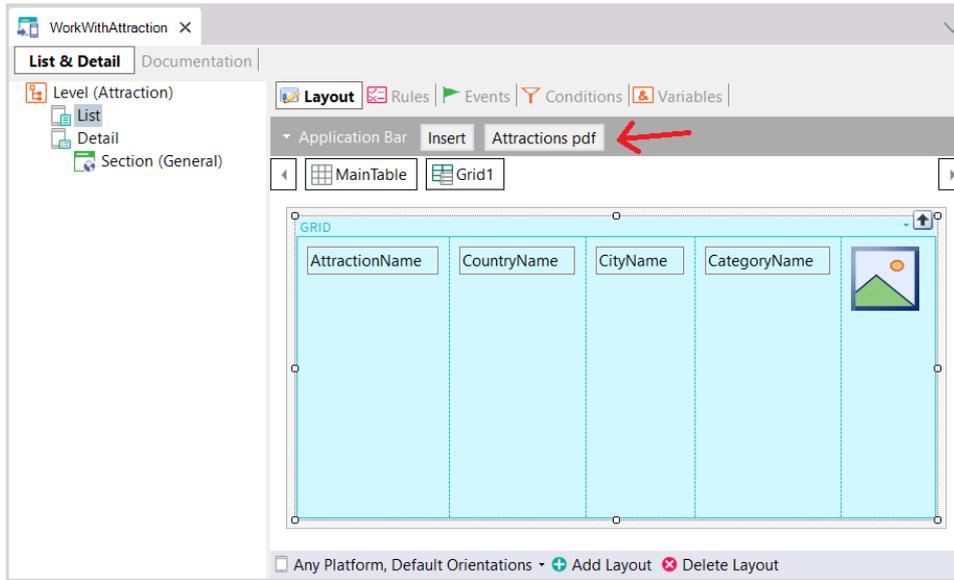
## Listados y acceso a los datos por código

[“Listados y comando For Each para consultar la base de datos”](#)

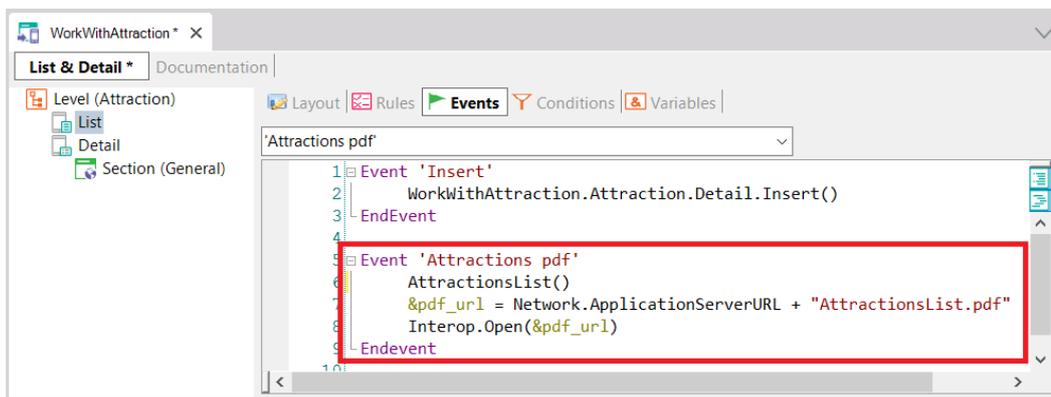
Si sigues el video tal cual, verás que el listado se abrirá en la url del front-end .NET, y no en el de Angular.

Para que el listado se ejecute dentro del front-end Angular hay que hacer lo siguiente:

- Dejar propiedad “Call protocol” con el valor default, Internal.
- Desde un **panel Angular** (puede ser el WorkWithAttraction) agregar un botón para invocar al procedimiento.



Y en la solapa Events agregar lo que se muestra:



Donde:

- La variable &pdf\_url debe ser del **dominio Url** (url, GeneXus:Domain)
- "AttractionsList.pdf" es el nombre que se le dio al pdf generado por el procedimiento AttractionsList, dado que allí se especificó la regla:  
Output\_file("AttractionsList.pdf", "pdf");

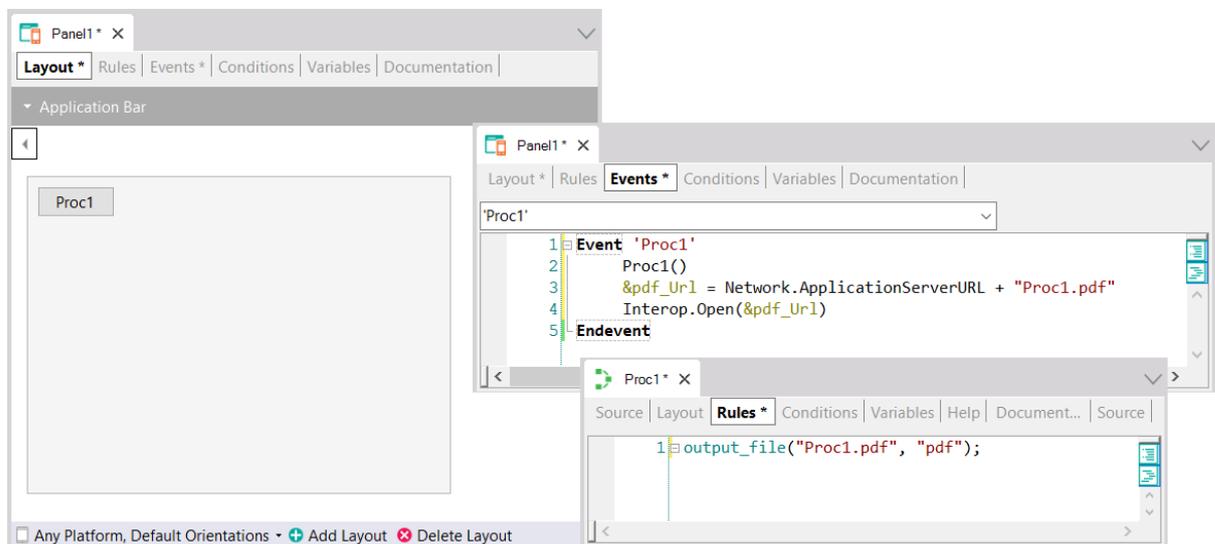
- Ejecutar este objeto panel -WorkWithAttraction- (que deberá tener la propiedad "Main program" en True) -con botón derecho / Run- y desde allí cuando se cargue el panel en el browser presionar el botón para imprimir el pdf.
- Explicación para quienes estén más avanzados:
  - La invocación al procedimiento, AttractionsList(), pide que el procedimiento sea ejecutado (y esto se hará en el server). GeneXus le habrá prendido la propiedad "Expose as Web Service" al procedimiento para exponerlo como servicio Rest, de modo que pueda ser invocado desde el cliente. Se comprenderá cuando se estudie la arquitectura de las aplicaciones Angular.

- Como el procedimiento tiene la regla **Output\_file**, el resultado se guardará en el servidor como archivo de nombre AttractionsList.pdf (el indicado en la regla output\_file).
- Como queremos que este archivo sea abierto por el front-end Angular, entonces debemos pedir al front-end que abra el recurso (en nuestro caso el pdf), para lo que debemos antes indicarle la url en este servidor, que es lo que armamos en la segunda sentencia del evento (la propiedad ApplicationServerURL del objeto externo Network devolverá la url donde se encuentran los servicios Rest).
- Para abrir un recurso, se cuenta con el método Interop.Open(...) -Interop es un objeto externo que viene en el módulo GeneXus (búscalo bajo el nodo References).

[“Cómo procesar información relacionada”](#), [“Cómo procesar información agrupada”](#), [“Fórmulas inline”](#)

Las mismas consideraciones: para ejecutar estos procedimientos tendrás que hacer lo mismo que en el caso anterior. Como no hay mayor diferencia entre usar el front-end Web (.Net) y el front-end Angular para esto, sugerimos ejecutar tal cual como se muestra en los videos (front-end Web .Net).

De todos modos, si quieres igualmente hacerlo en Angular: puedes crear un **objeto Panel** y en el Layout colocar tantos botones como Procedimientos necesites invocar. La variable que contendrá la url (&pdf\_Url) debe ser definida con la propiedad **Based on** con el dominio “Url”. Debes cambiar la propiedad del panel **Main Program** pasándola a **True**, y luego simplemente presionando **botón derecho/Run** sobre el panel ya lo ejecutarás.

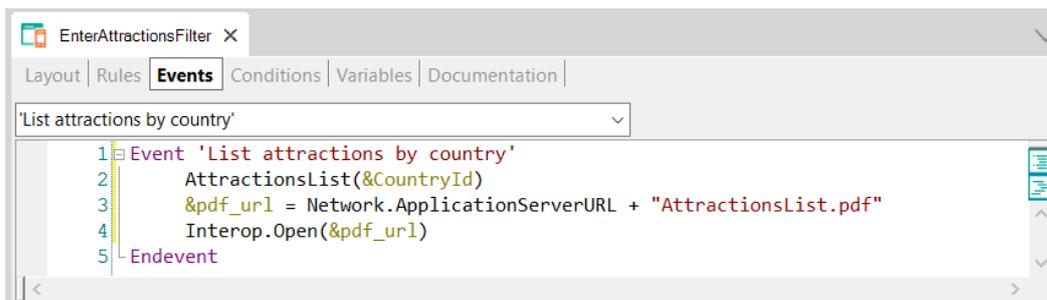
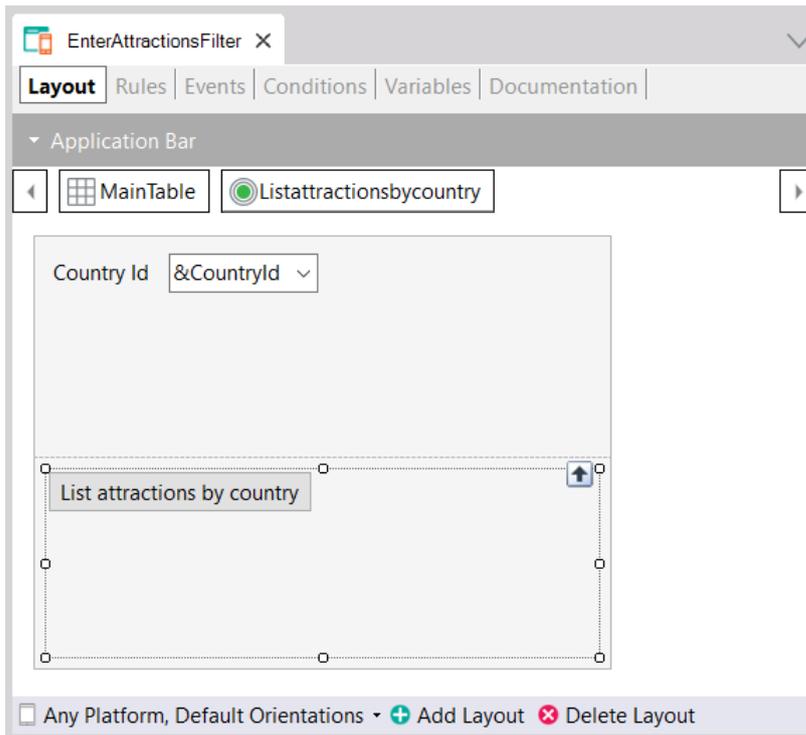


## Comunicación entre objetos

[“Invocaciones entre objetos”](#), [“Invocaciones entre objetos \(cont.\)”](#)

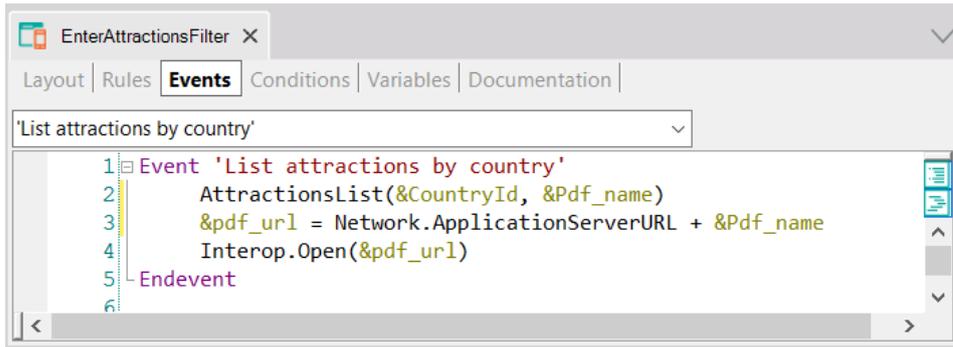
El análogo al Web Panel para Angular será el Panel. Recomendamos por simplicidad seguir los videos tal cual se muestran aquí, para Web Panels y no para Panels.

Una opción para ejecutar en Angular el primer panel del primer video, EnterAttractionsFilters:

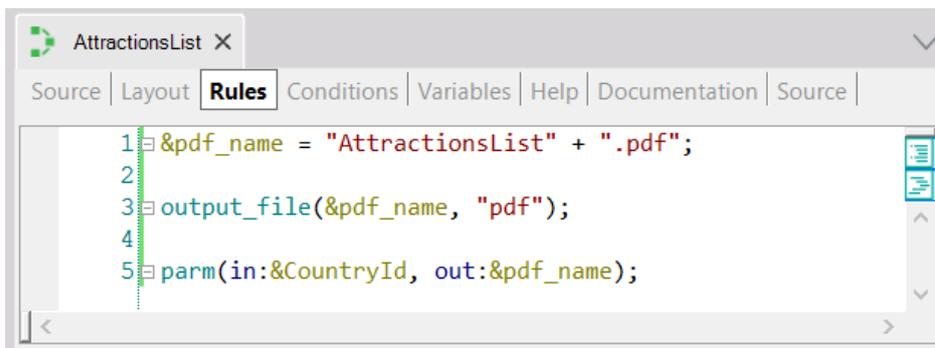


Colocando como “Main program” al panel y asegurándonos que el proc AttractionsList tenga el valor “Expose as web service” en True, “Rest Protocol” True y “Call protocol” Internal.

Una forma más prolija de implementar esto para no tener que recordar el nombre que se le dio al pdf dentro del procedimiento, es pasarlo también por parámetro:

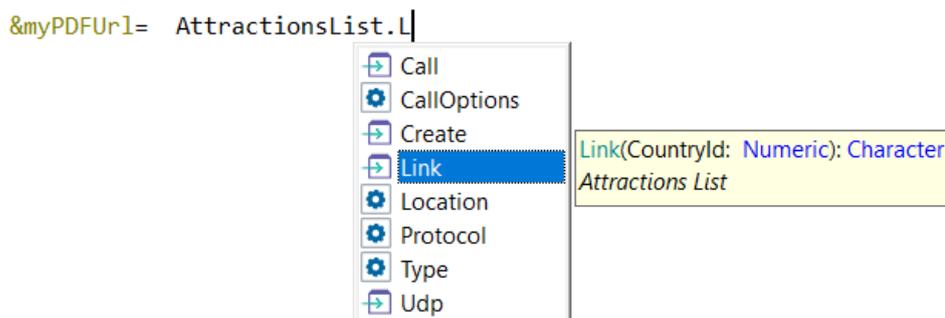


Así, en el procedimiento AttractionsList:



### Observaciones extras:

Cuando se llama directamente a un objeto por su nombre y entre paréntesis la lista de parámetros, es como si se estuviera escribiendo el método Call (o Udp si el objeto devuelve un parámetro). Pero hay otras formas de invocación, por ejemplo con el método Link. Ahora no nos detendremos en sus similitudes y diferencias, solo lo nombramos.



## Tipos de datos estructurados y Data Providers

[“Tipos de Datos Estructurados”](#), [“Variables que almacenan colecciones de datos en memoria”](#), [“Carga de Tipos de Datos Estructurados \(SDT\) mediante Data Providers”](#)

Todo esto es muy importante para todas las plataformas y vale exactamente igual. La consideración es la misma que vimos en los listados anteriormente (cómo poder visualizar un listado pdf en front-end Angular). Sugerimos, igual que antes, seguir los videos tal cual, en front-end Web (.NET).

## Actualización de la Base de Datos

Estos videos son fundamentales, más aún en Angular, donde no utilizamos la transacción como objeto de UI.

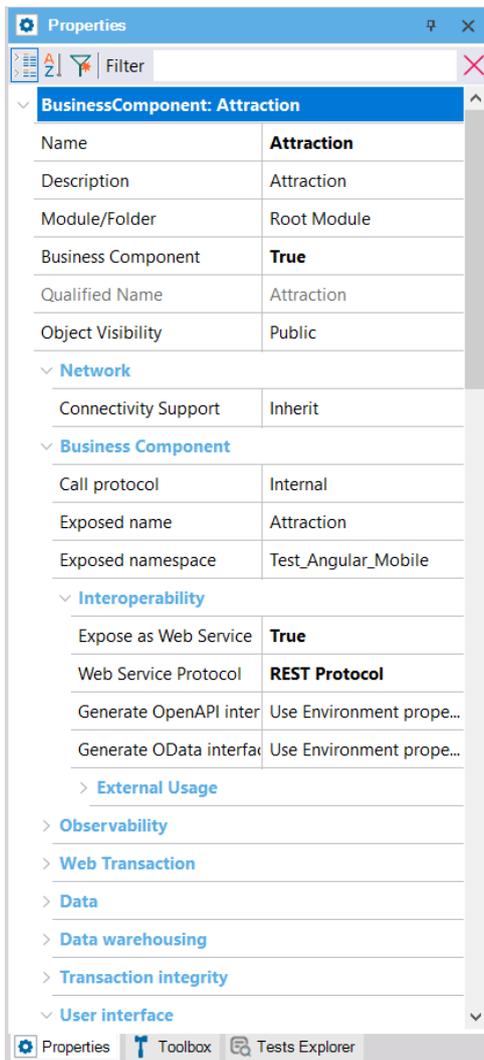
### [“Actualización de Business Components. Justificación”](#)

Cuando se mencionan las reglas que agregó el pattern Work With a la transacción Attraction, nos referimos al pattern for Web. No al pattern Work With genérico, que es el que vale para Angular o aplicaciones nativas. Es que, recordemos, en Angular y en aplicaciones nativas no se ejecutará el objeto transacción. De hecho, lo que se utilizará es lo que se empieza a introducir en este video, que es el Business Component que se crea a partir de la transacción.

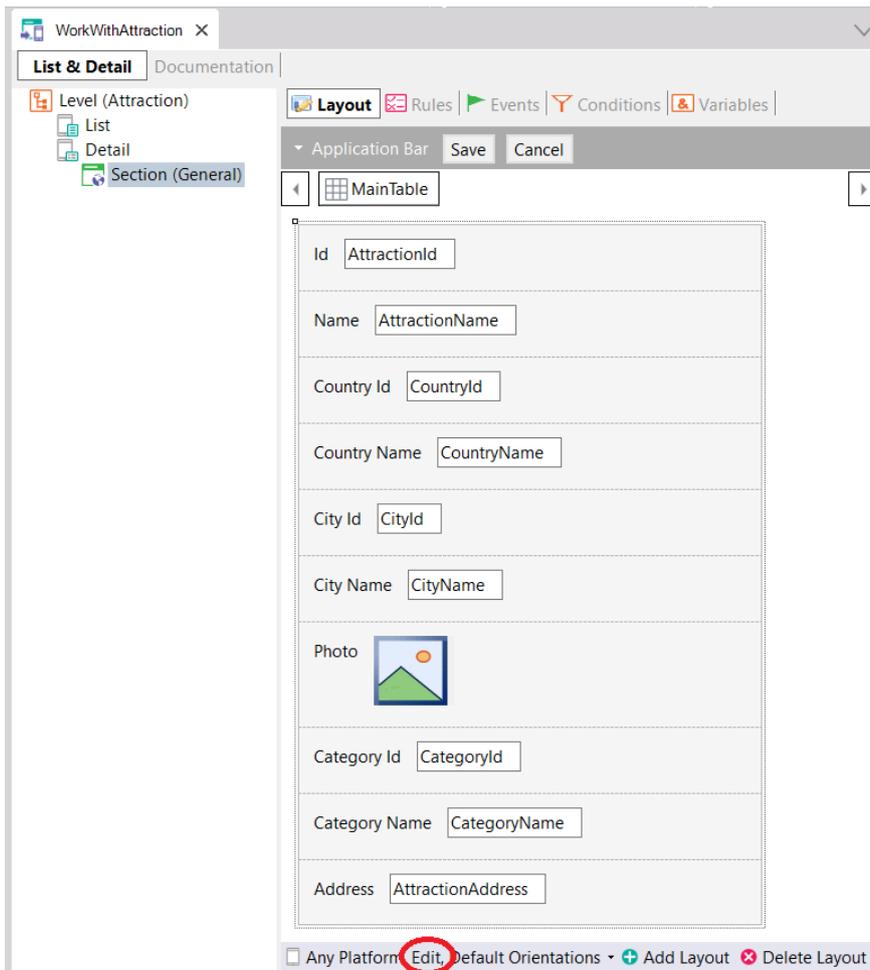
### [“Actualización con Business Component”](#)

Primero asegúrate de entender el video tal cual está, y recién después lee lo que sigue.

Si cuando estudiamos el pattern Work With creaste el pattern para Angular, entonces habrás observado que automáticamente GeneXus prendió la propiedad Business Component de la transacción, y las de interoperabilidad que se muestran en la imagen:



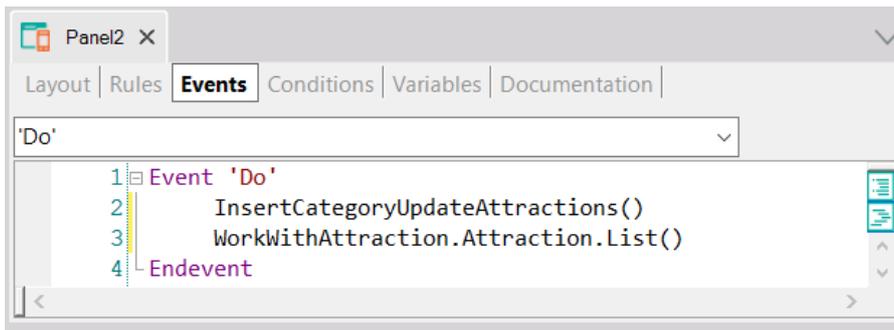
Esto sucedió porque para poder insertar atracciones, así como modificar o eliminar a través de las pantallas del Work With, en Angular no se llama a la transacción Attraction sino a la pantalla correspondiente a la sección General, layout Edit.



Y esos “atributos” que vemos en la pantalla en verdad no lo son. Son los elementos del Business Component. Cuando se presione el botón **Save** internamente lo que se hará será llamar al servicio Rest correspondiente al Business Component, publicado en el server, para que realice la operación correspondiente sobre la base de datos. En definitiva, desde un evento del cliente se estará llamando a un servicio Rest del server que es el que efectivamente intentará la actualización en la base de datos a través del business component, y de ser exitosa realizará el Commit. Este Commit el servicio Rest lo intenta realizar automáticamente.

Distinto es el caso si se quiere utilizar el BC desde un procedimiento, porque el procedimiento se ejecuta en el back-end y allí todo es idéntico a lo visto en el video.

Para ejecutar el procedimiento InsertCategoryUpdateAttraction crear un Panel con un botón Do en el layout y en el evento asociado escribir:



Otra vez: el procedimiento con “Call Protocol” Internal. Observar que al invocar al procedimiento desde el evento automáticamente GeneXus les prende las propiedades “Expose as Web Service” y “REST protocol” al procedimiento. Aprovechamos e invocamos al List del Work With para ver cómo aparece listada la nueva atracción.

Cuando el Business Component se utiliza dentro de un procedimiento, allí no es necesario ejecutarlo como servicio Rest (al business component, no al proc), porque el procedimiento se ejecutará en el server, y es allí que utilizará el Business Component. Es por eso que allí podrán encadenarse los commits como el desarrollador desee, explicitándolos. Todo tal cual se ve en el video.

La parte final del video, donde se habla de que pueden utilizarse los BCs no sólo en Procs sino también en eventos de Web panels es la que no será tan idéntica en Angular, donde tendrá limitaciones.

#### [“Actualización con Business Components. Un ejemplo”](#)

Para realizar esto en Angular, en lugar del Web Panel “CategoriesAndAttractions” crear el Panel análogo (y hacerlo main para poder ejecutarlo con Run).

Para la primera parte, cuando se invoca desde el evento ‘Do’ del botón del panel al proc, será todo idéntico (aunque observar que el proc automáticamente estará expuesto como servicio Rest).

**PERO:** cuando a mitad del video se copia el código del Source del procedimiento al evento ‘Do’ del Web panel aquí ya la cosa no funcionará igual. Es que al hacer esto en el Web Panel, aunque se trate de un evento en el cliente, GeneXus internamente mueve ese código al servidor, y lo ejecuta allí.

Para **Panels** esto no será igual. En el Panel ese código se ejecutará en el cliente, no en el servidor. Y, lógicamente, en el cliente no hay acceso a la base de datos. Sí funcionará asignar valor a los elementos del BC y hasta la operación de Insert o Update, porque para ello se invoca al BC como servicio Rest. Y se realiza en forma implícita el Commit. Pero no funcionará la consulta del resultado con If, ni siquiera If &category.Success().

Para chequear en evento del cliente el error del BC hay dos alternativas:

1. Si todo se realiza dentro de un comando Composite el chequeo del error se realiza automáticamente (se verá en el curso de Angular)
2. Consultar la variable &err. Ejemplo:

```
&bc.insert()
if &err = 0
    msg('Saved', nowait)
else
    msg(&errMsg, nowait)
endif
```

Para poder tener el control sobre cuándo realizar Commit o Rollback no habrá otra alternativa que invocar a un procedimiento (el que siempre se ejecutará del lado del Server).

Tampoco podrán utilizarse en eventos del cliente (ejecutados en el front-end) ni comandos for eachs, ni fórmulas que accedan a la base de datos, como por ejemplo find(CategoryId, CategoryName = "Monument").

Por tanto estas implementaciones en un Panel tendrán que hacerse invocando en los eventos a procs (que por ejecutarse en el server sí tienen a disposición todos los comandos de acceso a la base de datos).

Comprenderás mejor todo esto cuando visualices tres videos más adelante, en la sección de Arquitectura, el de nombre "[Aplicaciones GeneXus y su arquitectura](#)".

#### ["Población de datos con Business Component y Data Provider"](#)

No va a funcionar el método Insert para colección de BCs a nivel de evento del cliente, por lo que habrá que incluir todo el código del evento en un procedimiento.

#### ["Población automática de datos"](#)

El Data Provider que crea GeneXus para poblar con datos la tabla Category (y lo mismo para Attraction) se ejecutará en el back-end, es decir, será generado en .NET en nuestro caso. Por tanto es independiente del front-end. El front-end .NET en el video se está utilizando solamente para visualizar los datos ingresados.

#### ["Actualización con comandos específicos de procedimientos. Introducción"](#)

Vale tal cual, con la consideración de que estos comandos que acceden a la base de datos lógicamente solo pueden ejecutarse del lado del Server, es decir, en el back-end.

# Arquitectura

[“Aplicaciones GeneXus y su arquitectura”](#)

Es importante para entender las diferencias en la arquitectura de una aplicación con front-end que utiliza Web panels (.NET o Java) versus una que utiliza Panels (Angular o nativas).

## Pantallas Web con foco en Back-office

Todos los videos que aquí se incluyen son para front-end Web (.Net o Java). Aunque el objeto Web Panel no se utilizan para front-end Angular o nativo, parte de la lógica valdrá, con algunas diferencias, para los Panels, y aquí se explica desde cero, por lo que sugerimos visualizar estos videos con la misma atención que los anteriores.

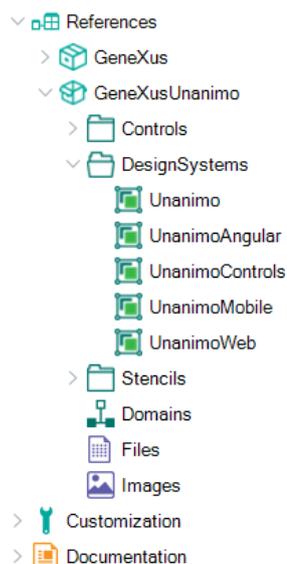
Luego, en el [curso de Angular](#) estudiarás las particularidades de los Panels.

## Diseño y modelado de pantallas

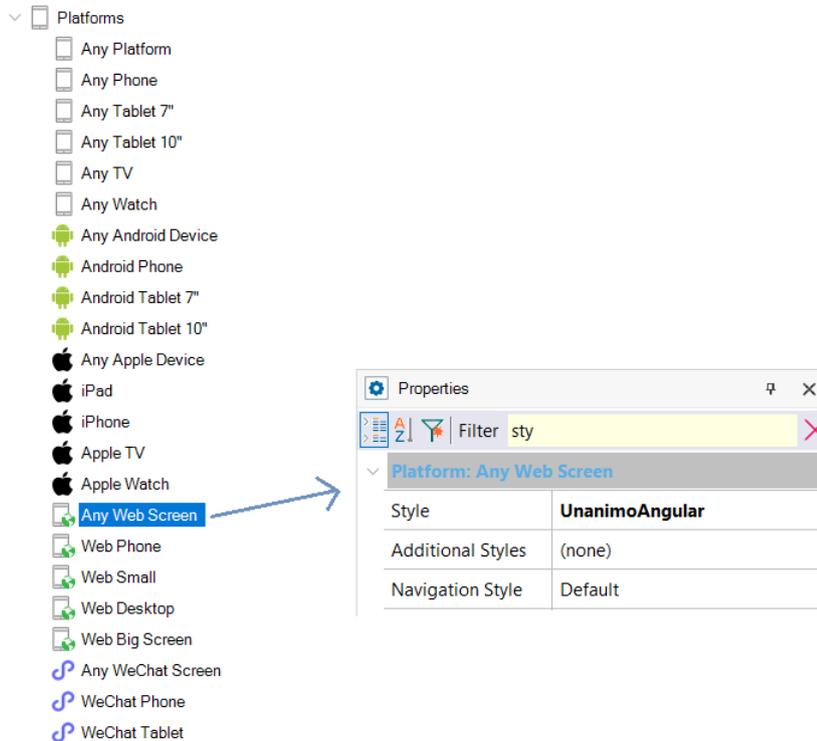
[UX Design. Introducción.](#)

Lo que mostramos en el video es para font-end Web (.Net/Java), pero conceptualmente vale prácticamente lo mismo para cualquier otro. Algunas diferencias destacables:

- Para front-end Web (.Net/Java) se define el Design System Object que aplicará por defecto a todas las pantallas a través de una propiedad de la versión, de nombre Default Style. Por defecto cuando se crea una KB se crea un DSO con su mismo nombre, y este DSO importará por defecto todo lo que venga del predefinido de nombre UnanimoWeb:



- Para front-end Angular la forma de indicar cuál será el DSO que comandará el diseño de las pantallas es en las propiedades de la plataforma. Aquí se puede ver que el valor default es el DSO UnanimoAngular, que viene predefinido en el módulo GeneXusUnanimo.



Te recomendamos seguir este video tal cual. Puedes aprovechar e investigar el DSO UnanimoAngular y probar con el WorkWithAttractions que habías aplicado para la transacción Attraction para Angular.

Más adelante (no ahora) puede interesarte visualizar [este webinar](#) donde explicamos los primeros pasos para diseñar de cero algunas pantallas para una aplicación customer-facing Angular, sin apoyarte en ningún DSO pre-existente sino creando uno de cero.

Todos los demás videos del curso Core valen tal cual.