Diseño de Aplicaciones

Diciembre, 2003

ARTech™
www.genexus.com

MONTEVIDEO – URUGUAY
CHICAGO – USA
MEXICO CITY – MEXICO
SAO PAULO – BRAZIL

Av. 18 de Julio 1645 P.4 - (5982) 402 2082
400 N. Michigan Ave. Suite 1600 - (312) 836 9152
Mariano Escobedo 353 A Of. 701 - (5255) 5255 4733
Rua Samuel Morse 120 Conj. 141 - (5511) 5502 6722

Copyright © ARTech Consultores S. R. L. 1988-2003.
Todos los derechos reservados. El presente documento no puede ser reproducido por ningún medio sin el consentimiento expreso de ARTech Consultores S.R.L. La información contenida en este documento es para uso personal del lector.

TRADEMARKS
ARTech y GeneXus son marcas o marcas registradas de ARTech Consultores S.R.L.
Todas las otras marcas citadas en este documento pertenecen a sus respectivos dueños.

Tabla de Contenido

Introducción 5
1. DESARROLLO DE UNA APLICACIÓN 7
Definir el objetivo 7
Definir el equipo de trabajo 7
Obtener una imagen global 8
Definir el alcance de la aplicación 8
Mantener el diseño simple 8
Orientarse a los datos 8
Metodología GeneXus 10
Objetos GeneXus 11
2. Objeto Transacción 12
Elementos 12
Estructura 13
Atributos 15
Dominios 17
Atributos Clave 18
Relación entre la estructura y el modelo de datos 18
Otras transacciones simples de nuestra aplicación 19
Definición de la transacción de pedidos 20
Integridad referencial en las transacciones 21
Niveles de una transacción 22
Tipos de relación entre los objetos 26
Forms 27
Diálogo campo a campo 28
Diálogo full-screen 28
Botones de posicionamiento 28
Atributos de entrada y atributos de salida 28
Facilidad de Prompt de Selección 29
Editor de forms 30
Controles 30
Tipos de controles atributo/variable 31
Personalizando el form de “Pedidos” 32
Paletas de herramientas 33
Propiedades de los controles 33
Editor de Propiedades, Métodos y Eventos 36
Fórmulas 37
Fórmulas Horizontales 37
Fórmulas Verticales 38
Fórmulas y Redundancia 40
Fórmulas de Fórmulas 40
Reglas 41
Default 41
Error 41
Asignación 42
Add y Subtract 42
Serial 43
Orden de evaluación 44
Call 44
Eventos y condiciones de disparo de reglas 44
Propiedades 46
3. Objeto Reporte 47
Elementos 47
Layout 48
Source 49
Comando FOR EACH 50
Reportes con varios FOR EACHs 58
Otros comandos 63
Condiciones 68
Reglas 69
Default 69
Parm 69
Report Wizard 70
Propiedades 74

4. Reportes Dinámicos – GeneXus Query 75
5. Objeto Procedimiento 78
Modificación de datos 78
Eliminación de datos 79
Creación de datos 80
Controles de integridad y redundancia 80
6. Objeto Work Panel 81
Elementos 81
Ejemplo 81
Form 84
Grid 84
Eventos 86
Evento Start 87
Evento Enter 87
Eventos definidos por el usuario (User Defined Events) 88
Evento Refresh 88
Carga de datos en el work panel 89
Condiciones 90
Orden de los datos en el grid 91
Reglas 92
Noaccept 92
Search 92
Bitmaps 93
Fixed Bitmaps 93
Dynamic Bitmaps 94
Propiedades 95
Diálogos Objeto-Acción 96
Browser de GeneXus 98
7. Objetos Web 99
Transacciones 99
Web Panels 99
Anexo A. Modelos de Datos Relacionales 100
Tablas 100
Clave primaria y Claves candidatas 100
Integridad Referencial 101
Índices 102
Normalización 103
Tabla extendida 104
Anexo B. Funciones, reglas y comandos 106
Indice de figuras 110
NOTAS 112

Introducción

El presente documento es una introducción al desarrollo de aplicaciones utilizando GeneXus. Está dirigido a profesionales de informática que se inician en el uso de GeneXus.

GeneXus es una herramienta para el desarrollo de aplicaciones sobre bases de datos. Su objetivo es ayudar a los analistas de sistemas a implementar aplicaciones en el menor tiempo y con la mejor calidad posible.

A grandes rasgos, el desarrollo de una aplicación implica tareas de análisis, diseño e implementación. La vía de GeneXus para alcanzar el objetivo anterior es liberar a las personas de las tareas automatizables (por ejemplo, la implementación), permitiéndoles así concentrarse en las tareas realmente difíciles y no automatizables (comprender el problema del usuario).

Desde un punto de vista teórico, GeneXus es una herramienta asociada a una metodología de desarrollo de aplicaciones. En cuanto a la metodología, tiene algunos puntos de contacto con las metodologías tradicionales, pero aporta enfoques bastante diferentes en otros.

En común con las metodologías tradicionales, se mantiene el énfasis del análisis y diseño de la aplicación sobre la implementación. Con GeneXus este enfoque se resalta más aún, ya que el programador GeneXus estará la mayor parte del tiempo realizando tareas de análisis[1] y GeneXus, en sí, tareas de diseño e implementación (por ejemplo, normalización de la base de datos, creación de la base de datos, generación de programas, etc.).

Por otro lado, algunos de los enfoques metodológicos son bastante diferentes que los habituales, como el comenzar el análisis de la aplicación por la interfaz con el usuario y la casi nula referencia a la implementación física del sistema, entre otros.

El punto de partida de la metodología GeneXus es describir las visiones de los usuarios para modelar el sistema. A partir de este modelo, GeneXus construye el soporte computacional (base de datos y programas) en forma totalmente automática.

Para presentar estos nuevos conceptos, y a los efectos de no realizar una presentación demasiado abstracta del tema, se ha elegido una aplicación que se irá desarrollando a través de los distintos capítulos. El primer capítulo presenta la aplicación y los aspectos iniciales de un desarrollo con GeneXus.

Los siguientes capítulos presentan algunos de los objetos con los que GeneXus cuenta para describir aplicaciones, los que serán utilizados para desarrollar la aplicación de ejemplo. Así, el segundo capítulo trata sobre el objeto Transacción, el tercero sobre el objeto Reporte, el quinto sobre el objeto Procedimiento, el sexto sobre el Work Panel, y por último se tratan los objetos para trabajar en ambiente Web.

Debido a que en todos los capítulos se asume cierto conocimiento sobre bases de datos relacionales, se ha incluido un anexo sobre el tema que describe los conceptos necesarios para poder entender este documento. Se recomienda su lectura antes de leer el segundo capítulo.

Por razones didácticas, en este documento no se tratan los siguientes temas:

[1] Es por ello que es común referirse al usuario de GeneXus como “analista GeneXus” en lugar de “programador GeneXus”

- ***Ciclo de vida de una aplicación:*** Una aplicación tiene un ciclo de vida, que comienza cuando se planifica la misma y termina cuando la aplicación sale de producción. GeneXus acompaña todo este ciclo.
- ***Uso de GeneXus***.

La versión de GeneXus utilizada en este libro es la 8.0.

Puede obtenerse una versión de prueba de GeneXus accediendo a: http://www.genexus.com/trial

Con el objetivo de brindar a los desarrolladores una fuente única de información potenciada con una "ingeniería de búsqueda" que les permita acceder rápida y fácilmente a la información técnica (release notes, manuales, tips, whitepapers, etc) sobre los diferentes productos que desarrolla ARTech, se creó la GXDL (GeneXus Developer Library), una biblioteca que centraliza toda esta información técnica.

Puede ser consultada online o puede ser utilizada en forma local mediante una sencilla instalación. (http://www.gxtechnical.com/gxdl)

Por sugerencias o comentarios acerca de la presente edición, le solicitamos nos los haga llegar a la dirección training@genexus.com

1. DESARROLLO DE UNA APLICACIÓN

La aplicación que se ha elegido es una simplificación de una aplicación real: *Sistema de compras para una cadena de farmacias*.

En una cadena de farmacias se desea implementar un sistema de compras que permita a los *analistas de compras* realizar dicha tarea con la mayor cantidad de información posible. La función de un *analista de compras* es decidir los *pedidos* que se deben efectuar a los *proveedores*, de los distintos *artículos*.

Funcionamiento del sistema
Se desea que el *analista de compras* utilice el computador para definir los *pedidos* de *artículos* a los distintos *proveedores*. Una vez que el pedido esté hecho y aprobado, se quiere que el computador emita las *órdenes de compra*. En el momento de hacer un pedido es necesario hacer varias consultas; por ejemplo cuánto hay en stock del artículo, cuál fue el precio de la última compra, etc.

Los siguientes puntos describen los pasos seguidos para el desarrollo de esta aplicación.

Definir el objetivo

No se debe olvidar que los computadores son meras herramientas. Los usuarios de los sistemas tienen objetivos específicos. Ellos esperan que la aplicación los ayude a alcanzarlos más rápidamente, más fácilmente, o a un menor costo. Es parte del trabajo de análisis el conocer esos propósitos y saber por medio de qué actividades los usuarios quieren alcanzarlos.
Estos objetivos deben ser expresados en pocas palabras y ser conocidos por todas las personas involucradas con el sistema.
En el ejemplo, algunos de los propósitos posibles son:

- “El sistema de compras debe disminuir la burocracia existente para la formulación de un pedido.”
- “El sistema de compras debe asistir a usuarios no entrenados en la formulación de pedidos de tal manera que su desempeño se asemeje al de un experto.”
- “El sistema de compras debe permitir la disminución del stock existente en las farmacias.”

De todos los objetivos posibles se debe elegir uno como el objetivo principal o prioritario. Esto es muy importante para el futuro diseño de la aplicación. Basta con observar cómo los distintos objetivos anteriores conducen a diseños diferentes.
Para nuestro ejemplo elegiremos el primer objetivo, dado que en la situación real el analista de compras no tenía toda la información que necesitaba, por lo cual debía consultar una serie de planillas manuales y llamar por teléfono a los empleados del depósito para que realizaran un conteo manual del stock.

No se debe confundir el objetivo de la aplicación -el QUE- con la funcionalidad de la misma -COMO se alcanzará el objetivo.

Definir el equipo de trabajo

A continuación se debe definir cuál será el equipo de personas encargado de la implementación del sistema. Dicho equipo debe tener como mínimo dos personas:

- El analista de sistemas, y
- Un usuario.

Los analistas de sistemas que trabajen en el desarrollo de la aplicación deben cumplir dos condiciones:

- Haber sido entrenados en el uso de GeneXus
- Tener una orientación a las aplicaciones.

Se recomienda que dichos analistas pasen algún tiempo trabajando con los usuarios en el comienzo del proyecto a los efectos de familiarizarse con los conceptos, vocabulario, problemas existentes, etc.

Como una aplicación con GeneXus se desarrolla de una manera incremental, es muy importante la participación de los usuarios en todas las etapas del desarrollo. Se recomienda tener un usuario principal disponible para la prueba de los prototipos y tener acceso a los demás usuarios de una manera fluida.

Dado que con GeneXus los miembros del equipo estarán la mayor parte del tiempo trabajando en tareas de diseño y no de codificación, se debe tomar como criterio general el trabajar en equipos pequeños, por ejemplo, de no más de cinco personas.

Obtener una imagen global

Se deben tener entrevistas con el nivel gerencial más alto posible, de modo de obtener información sobre la posición relativa e importancia de la aplicación dentro de toda la organización.

Definir el alcance de la aplicación

Luego de un estudio primario se debe decidir cuál será el alcance de la aplicación para poder cumplir con el objetivo. Para ello se recomienda seguir el *Principio de Esencialidad*:

"Solo lo imprescindible, pero todo lo imprescindible"

En el ejemplo, una vez que una orden de compra es enviada a un proveedor, se debe controlar cómo y cuándo se fueron entregando efectivamente los productos. Sin embargo vemos que esto no es imprescindible para cumplir el objetivo de la aplicación y por lo tanto no será tratado.

Mantener el diseño simple

Se debe planificar pensando en un proceso de diseño y desarrollo incremental. Comenzando por pequeños pasos y verificando la evolución del modelo frecuentemente con el usuario.

Orientarse a los datos

En esencia, una aplicación es un conjunto de mecanismos para realizar ciertos procesos sobre ciertos datos.
Por lo tanto, en el análisis de la aplicación, se puede poner mayor énfasis en los procesos o en los datos.
En las metodologías tradicionales -como el *Análisis Estructurado*- el análisis se basa en los procesos. En general, este análisis es top-down: se comienza con la definición más abstracta de la función del sistema y luego ésta se va descomponiendo en funciones cada vez más simples, hasta llegar a un nivel de abstracción suficiente como para poder implementarlas directamente.

Sin embargo, este enfoque tiene ciertos inconvenientes:

- Las funciones de un sistema tienden a evolucionar con el tiempo, y por tanto, un diseño basado en las funciones será más difícil de mantener.
- La idea de que una aplicación se puede definir por una única función es muy controvertida en aplicaciones medias o grandes.
- Frecuentemente se descuida el análisis de las estructuras de datos.
- No facilita la integración de aplicaciones.

Si, por el contrario, el diseño se basa en los datos se pueden obtener las siguientes ventajas:

- ***Más estabilidad.*** Los datos tienden a ser más estables que los procesos y en consecuencia la aplicación será más fácil de mantener.

- ***Facilidad de integración con otras aplicaciones.*** Difícilmente una aplicación sea totalmente independiente de las otras dentro de una organización. La mejor forma de integrarlas es tener en cuenta los datos que comparten.

Por lo tanto, orientarse a los datos parecería ser más beneficioso. La pregunta natural que surge es, entonces, cómo analizar los datos. La respuesta de GeneXus es: analizar directamente los datos que el usuario conoce, sin importar como se implementarán en el computador.

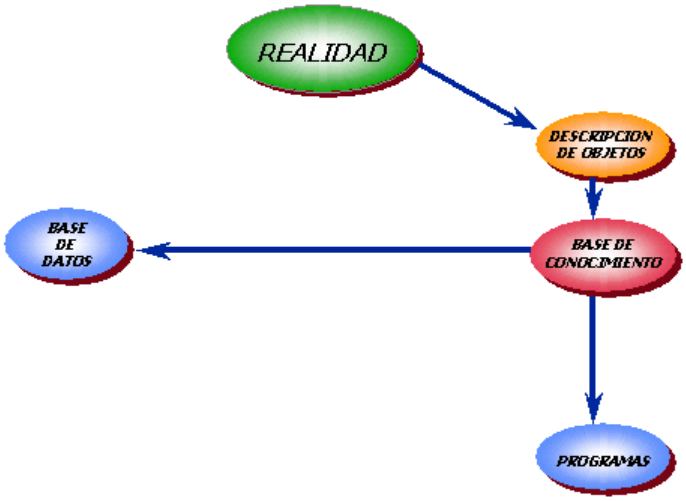
El diseño comienza -como veremos en más detalle en el siguiente capítulo- estudiando cuáles son los objetos que el usuario manipula. Para cada uno de estos objetos se define cuál es su estructura de datos y posteriormente cuál es su comportamiento.

De esta manera se alcanzan dos objetivos importantes: el análisis se concentra en hechos objetivos, y éste puede ser evaluado directamente por los usuarios, utilizando la facilidad de prototipación de GeneXus.

Metodología GeneXus

Para comenzar el desarrollo de una aplicación con GeneXus, el primer paso consiste en crear un nuevo proyecto o **base de conocimiento**. El siguiente paso es describir las visiones de los usuarios, a partir de las cuáles GeneXus captura y sistematiza el conocimiento. Para ello, se deben identificar los objetos de la realidad y pasar a definirlos mediante **objetos GeneXus**.

Con la definición de estos objetos, GeneXus puede extraer el conocimiento, y diseñar la base de datos y los programas de la aplicación en forma **totalmente automática**.



Si un objeto de la realidad cambia, se identifican nuevas o diferentes características acerca del mismo, o si se encuentran objetos aún no modelados, el analista GeneXus debe reflejar dichos cambios en los objetos GeneXus que corresponda, y la herramienta se encargará automáticamente de realizar las modificaciones necesarias tanto en la base de datos como en los programas asociados.

La metodología GeneXus es una **metodología incremental**, pues parte de la base de que la construcción de un sistema se realiza mediante aproximaciones sucesivas. En cada momento el analista GeneXus define el conocimiento que tiene y luego cuando pasa a tener más -o simplemente diferente- conocimiento, GeneXus se ocupará de hacer automáticamente todas las adaptaciones en la base de datos y programas. Si GeneXus no fuera capaz de realizar automáticamente estas modificaciones conforme se realicen cambios que así lo requieran, el desarrollo incremental sería inviable.

GeneXus puede generar la aplicación en diversas plataformas. Así, por ejemplo, se puede implementar la aplicación para un DBMS como Sql Server, Oracle, Informix, etc., en un lenguaje como .NET, C/SQL, Visual Basic, Visual FoxPro, etc. Las tablas, índices y programas serán generados por GeneXus en la plataforma elegida, y el analista no tendrá que contar con conocimientos profundos del lenguaje ni del DBMS.

Concluyendo, si bien GeneXus eleva en mucho la productividad en el desarrollo de aplicaciones en comparación con las metodologías tradicionales, el punto más fuerte es sin duda la enorme disminución en los costos de mantenimiento.

Objetos GeneXus

A continuación se describen los objetos GeneXus más importantes (no siendo los únicos):

- **Transacciones**[2]
  Permiten definir objetos de la realidad –reales o imaginarios- que el usuario manipula (ej: analistas de compras, artículos, proveedores, pedidos, etc.). Son los primeros objetos en definirse, ya que a través de las transacciones, GeneXus infiere el diseño de la base de datos.
  Cada transacción tiene asociada una pantalla para ambiente windows y otra para ambiente web, que permititen al usuario dar altas, bajas y modificaciones en forma interactiva a la base de datos. El analista GeneXus decidirá si trabajar en ambiente windows, web, o ambos.

- **Reportes**
  Permiten recuperar información de la base de datos, y desplegarla ya sea en la pantalla, en un archivo o impresa en papel. Son los típicos listados o informes. No permiten actualizar la información de la base de datos.

- **Procedimientos**
  Tienen las mismas características que los reportes, pero a diferencia de éstos, permiten además la actualización de la información de la base de datos.

- **Work Panels**
  Permiten al usuario realizar interactivamente consultas a la base de datos, a través de una pantalla, en ambiente windows. Ejemplo: un work panel permite al usuario ingresar un rango de caracteres, y muestra a continuación todos los clientes cuyos nombres se encuentran dentro del rango.
  Son objetos muy flexibles que se prestan para múltiples usos.
  No permiten la actualización de la base de datos, sino solo consultarla.

- **Web Panels**
  Son similares a los work panels, salvo que son usados a través de navegadores en ambiente Internet/Intranet/Extranet.

[2] No debe confundirse el lector con el término "transacción de base de datos" que se refiere al conjunto de operaciones sobre la base de datos que se ejecutan todas o ninguna. En GeneXus ese concepto recibe el nombre de Unidad Transaccional Lógica (UTL).

2. Objeto Transacción

El análisis de toda aplicación GeneXus comienza con el diseño de las transacciones. Las transacciones permiten definir los objetos de la realidad. Para identificar cuáles transacciones deben crearse, se recomienda prestar atención a los sustantivos que el usuario menciona cuando describe la realidad.

Además de tener por objetivo la definición de la realidad y la consecuente creación de la base de datos normalizada[3], por cada transacción se generarán programas para ambiente windows y para ambiente web, para permitir dar altas, bajas y modificaciones en forma interactiva a la base de datos. El analista GeneXus decidirá si generar solo los programas para trabajar en ambiente windows, en web, o en ambos.

Por tanto, además de determinar una estructura de tablas, al igual que el resto de los objetos GeneXus que veremos, las transacciones provocan la generación de programas.

Es importante tener en cuenta que todo el proceso de desarrollo es incremental y por lo tanto no es necesario definir en esta etapa todas las transacciones y cada una de ellas en su mayor detalle. Por el contrario, lo importante aquí es reconocer las más relevantes y para cada una de ellas identificar y definir su estructura.

Para poder definir cuáles son las transacciones se recomienda estudiar cuáles son los objetos -reales o imaginarios- que el usuario manipula. Es posible encontrar la mayor parte de las transacciones a partir de:

- **La descripción del sistema.** Cuando un usuario describe un sistema se pueden determinar muchas transacciones si se presta atención a los sustantivos que utiliza. En el ejemplo:

  Analistas de compras
  Pedidos
  Proveedores
  Artículos
  Órdenes de Compra

- **Formularios existentes.** Por cada formulario que se utilice en el sistema es casi seguro que existirá una transacción para el ingreso del mismo.

Elementos

Para cada transacción se definen, entre otros, los siguientes elementos:

Estructura
Permite definir los atributos (campos) que componen la transacción y la relación entre ellos. A partir de la estructura de las transacciones, GeneXus inferirá el diseño de la base de datos: tablas, claves, índices, etc.

Form GUI-Windows
Cada transacción contiene un form (pantalla) GUI-Windows (Graphical User Interface Windows) mediante el cuál se realizarán las altas, bajas y modificaciones en ambiente Windows.

[3] GeneXus diseña la base de datos en tercera forma normal.

Form Web
Cada transacción contiene un form (pantalla) Web mediante el cuál se realizarán las altas, bajas y modificaciones en ambiente Web.

Reglas
Las reglas permiten definir el comportamiento particular de las transacciones. Por ejemplo, permiten definir valores por defecto para los atributos, definir chequeos sobre los datos, etc.

Eventos
Las transacciones soportan la programación orientada a eventos. Este tipo de programación permite definir código ocioso, que se activa en respuesta a ciertas acciones provocadas por el usuario o por el sistema.

Subrutinas
Permiten definir rutinas locales a la transacción, que se ejecutan al ser invocadas desde la propia transacción.

Propiedades
Son características a ser configuradas para definir ciertos detalles referentes al comportamiento general de la transacción.

Style Asociado
Styles (estilos) son objetos GeneXus utilizados para definir estándares. No se tratarán en la presente exposición.

Ayuda
Permite la inclusión de texto de ayuda, para ser consultado por los usuarios en tiempo de ejecución de la transacción.

Documentación
Permite la inclusión de texto técnico, para ser utilizado como documentación del sistema.

Al definir / editar un objeto GeneXus (tanto transacción como cualquier otro) es posible acceder a los distintos elementos que lo componen utilizando las solapas que aparecen en la parte inferior de la ventana de edición del objeto.
En caso de tratarse de una transacción, son los siguientes:

Fig. 1: Solapas que figuran en la parte inferior de la ventana de edición de una transacción

La solapa *Web Form* está disponible para el caso en el que queramos generar la transacción para aplicaciones web de forma tal de ser ejecutadas desde un navegador. Esta solapa es para editar el form Web diseñado. Volveremos sobre este punto en el capítulo que trata sobre Objetos Web.

Estructura

La estructura define qué atributos (campos) integran la transacción y cómo están relacionados.

Fig. 2: Estructura transacción "Proveedores"

En el ejemplo, la transacción "Proveedores" posee los siguientes atributos, que componen la información que se quiere tener de un proveedor:

PrvCod	Código de proveedor
PrvNom	Nombre del proveedor
PrvDir	Dirección del proveedor
PrvTotPed	Importe total pedido al proveedor

A partir de esta estructura GeneXus creará una tabla:

PROVEEDORES	*PrvCod*	*PrvNom*	*PrvDir*	*PrvTotPed*

Y creará para la transacción un form GUI por defecto:

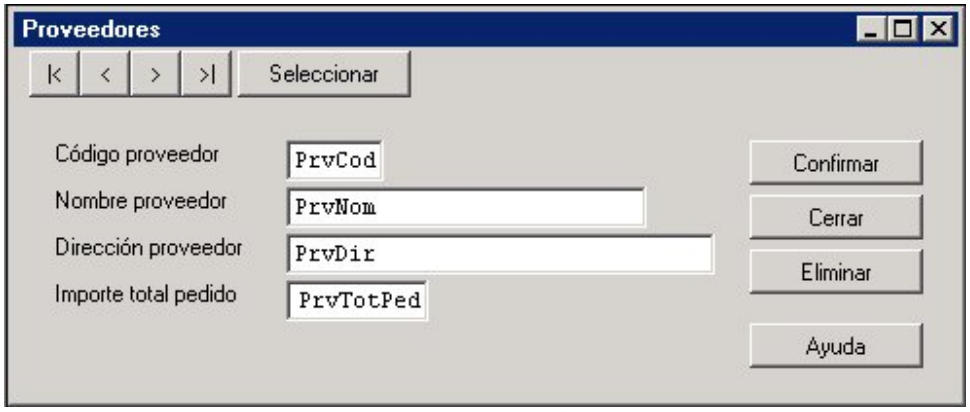


Fig. 3: Form GUI de la transacción "Proveedores"

así como uno Web:

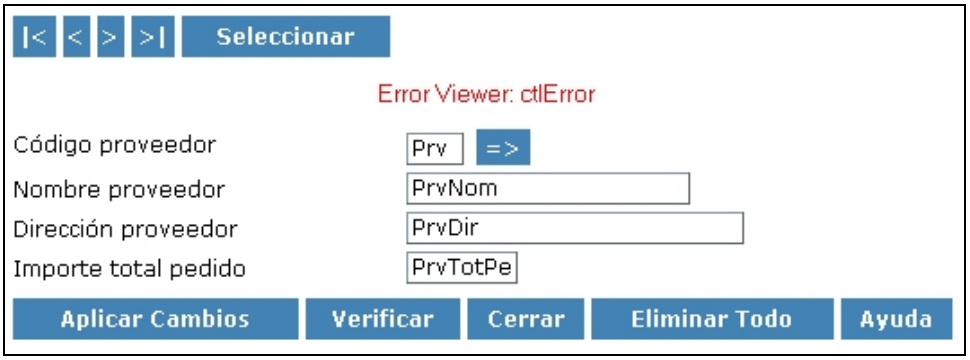


Fig. 4: Form Web de la transacción "Proveedores"

que podrán luego ser modificados por el analista.

A través de este form (ya sea del GUI o del Web, esto dependerá del ambiente para el que se genere la aplicación), el usuario podrá ingresar nuevos proveedores, que serán dados de alta como nuevos registros en la tabla asociada. También podrá modificar los datos de un proveedor dado. Para ello, la transacción "Proveedores" tiene encapsulada la lógica que le permite acceder a la tabla asociada, recuperar el registro solicitado, y mostrar sus datos en el form, de manera tal que el usuario pueda verlos y modificarlos. Al realizar una modificación sobre los datos que aparecen en el form, por ejemplo cambiar el nombre del proveedor, la transacción en el momento adecuado, deberá realizar ese cambio en el registro físico correspondiente.
Algo similar ocurre con respecto a las eliminaciones de registros.

El analista GeneXus no deberá programar nada de esto, pues ya está implícito en la lógica de toda transacción. La codificación la realiza GeneXus, y es absolutamente transparente para el analista, quien solo debe encargarse de aspectos de alto nivel que hacen al objeto.
Como veremos, al representar transacciones en GeneXus estamos:

- Explícitamente: definiendo la interfaz con el usuario en la presentación y captura de los datos
- Implícitamente: definiendo la relación entre los datos (tablas, índices, etc.)

Atributos

Cada atributo de la aplicación tiene propiedades asociadas, que pueden editarse para ser modificadas. Por ejemplo, la siguiente pantalla corresponde a las propiedades del atributo *PrvCod* definido:

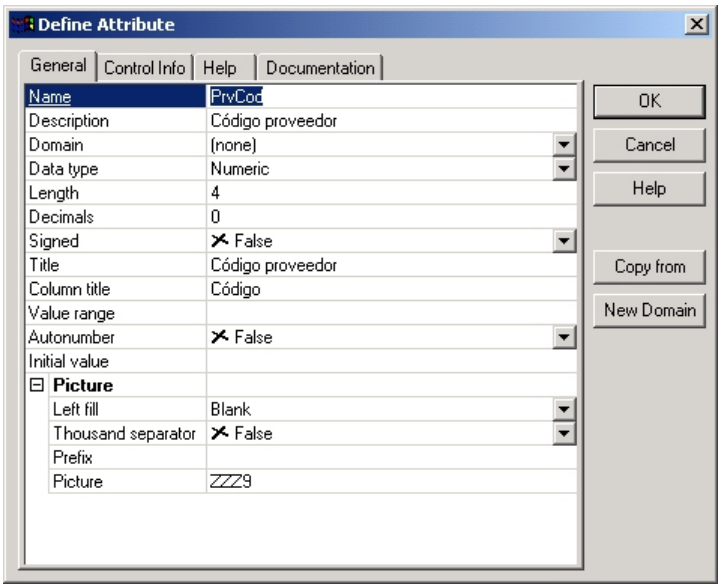


Fig. 5: Diálogo "Define Attribute"

Como podemos ver, esta pantalla tiene cuatro solapas:

1. **General**
   Permite configurar características generales del atributo, como:

   Name: Es el nombre del atributo. Se utiliza para identificarlo.

Description: Es un "nombre largo" o descripción ampliada del atributo. Debe identificarlo bien, con independencia del contexto, y principalmente debe ser entendible por el usuario final.
Debe ser un identificador global del atributo, es decir, no se debe asignar a dos atributos de la aplicación la misma descripción, ya que como se verá en el capítulo que trata sobre *Reportes Dinámicos*, será a través de esta descripción que el usuario final podrá, utilizando la herramienta *GeneXus QUERY,* seleccionar atributos para definir sus propias consultas a la base de datos.

Relacionadas con la **Description**, están las propiedades **Title** y **Column Title** que aparecen más abajo en la pantalla de edición de propiedades. Por defecto toman el mismo valor que se especifica en **Description**, puediendo luego ser modificadas por el analista. Estas propiedades se describen a continuación:

Title: La descripción que se ingrese aquí, será colocada por defecto al lado del atributo cada vez que éste se utilice en salidas "planas".
Si observamos las figuras 3 y 4 que corresponden a los forms Win y Web de la transacción "Proveedores" – forms planos - aparece "Código proveedor" a la izquierda del atributo *PrvCod*, que es justamente el valor que figura en la propiedad "Title" del atributo, como podemos ver en la fig.5.

Column Title: La descripción que se ingrese aquí será colocada por defecto como título del atributo, cada vez que se lo incluya como columna de un grid (grilla).
Si observamos la fig.5, en la propiedad "Column Title" del atributo *PrvCod* modificamos el valor por defecto, acortando la descripción. Así, cuando este atributo se incluya como columna de un grid, en lugar de aparecer con la descripción "Código proveedor", aparecerá simplemente con "Código" (puede referirse a las figuras 9 y 64 para ver dos ejemplos de ello)

Estos dos "títulos" permiten definir las descripciones que queremos que acompañen a los atributos cada vez que éstos se incluyan en las pantallas "default" de los objetos GeneXus.

Domain: Permite asociar un dominio al atributo. Al asignar un dominio a un atributo, ciertas propiedades del atributo se deshabilitarán (por ejemplo *Data Type* y *Length*) y tomarán los valores del dominio. En caso que el atributo no pertenezca a un dominio, se dejará el valor "(none)" en esta propiedad, y de esta forma las propiedades correspondientes al tipo de datos del atributo aparecerán habilitadas, para que el analista especifique sus valores.

Data Type: Permite indicar el tipo de datos del atributo. Aquí se podrá elegir uno de los tipos de datos soportados por GeneXus. Entre ellos se encuentran los tipos *Numeric*, *Character*, *VarChar*, *Long VarChar*, *Date*, *DateTime*, *Blob*. Dependiendo del tipo de datos que se seleccione, habrán ciertas propiedades, u otras, para configurar.

Length: Permite indicar el largo del atributo. Si en la propiedad **Data Type** se indica que el atributo es numérico, entonces, se deberá tener en cuenta que el largo incluya las posiciones decimales, el punto decimal y el signo. Esta propiedad estará deshabilitada cuando el tipo de datos elegido no requiera establecer un largo (por ejemplo, si se trata del el tipo de datos Date).

Decimals: Si en la propiedad **Data Type** se indica que el atributo es numérico, se ofrecerá esta propiedad, para que se especifique la cantidad de decimales. El valor 0 en este campo, indicará que se trata de un entero.

Signed: Si en la propiedad *Data Type* se indica que el atributo es numérico, se ofrecerá esta propiedad, para que se indique si manejará signo, o no. El valor por defecto es "False", lo que indica que no se representarán valores negativos.

Value Range: Permite indicar un rango de valores válidos para el atributo. Ej:
1:20 30: - significa que los valores válidos son aquellos entre 1 y 20; y 30 o mayor.
1 2 3 4 - significa que los valores válidos son 1, 2, 3 y 4.
'S' 'N' - significa que los valores válidos son 'S' y 'N'.

Autonumber: Permite indicar que el atributo debe ser numerado automáticamente por el sistema. Solo aplica si el atributo es de tipo numérico y clave primaria de la tabla. Si el atributo no es clave primaria, la autonumeración puede resolverse mediante la regla "Serial" que veremos, o por procedimiento.

Initial Value: Cuando el atributo se está agregando a una tabla que ya existe y tiene registros cargados, esta propiedad permite especificar qué valor "inicial" se le dará a este atributo para los registros pre-existentes.

Picture: Permite indicar el formato de edición para la entrada y salida del atributo. Dependiendo del tipo de datos del atributo, aparecerán determinadas propiedades bajo esta categoría. Por ejemplo, en el caso de tipo de datos numérico, se permite especificar si se utilizará separador de miles o no, en el caso de character se permite especificar si se desea que todos los caracteres se acepten y desplieguen automáticamente en mayúsculas, etc.

2. **Control Info**
   Permite indicar el tipo de control que se desea asociar al atributo. Esta información es utilizada por GeneXus cuando crea los forms por defecto. Dependiendo del tipo de control que se seleccione, se solicitará cierta información relacionada u otra. Volveremos sobre este punto un poco más adelante cuando estudiemos los forms de transacciones.

3. **Help**
   A todo atributo puede asociársele una descripción larga para ayudar al usuario final en tiempo de ejecución. Si el usuario final solicita ayuda presionando F1 sobre el atributo, entonces esta descripción será desplegada.

4. **Documentation**
   Permite definir documentación técnica del atributo, útil para otros desarrolladores.

Dominios

Es común cuando estamos definiendo la base de datos, tener atributos que comparten definiciones similares, y para los que no se puede establecer ninguna relación directa. Por ejemplo, es común almacenar todos los nombres en atributos de tipo character y largo 25.

El uso de dominios permite utilizar definiciones de atributos genéricos. Por ejemplo, en la transacción de proveedores tenemos el nombre, *PrvNom*, y más adelante vamos a definir el nombre del analista, entonces podemos definir un dominio *Nombres* de tipo character con largo 25 y asociarlo a estos atributos. De esta forma, si en el futuro cambia la definición de este dominio, los cambios serán propagados automáticamente a los atributos que pertenecen a él.

Los dominos no son excluivos de los atributos. También podremos definir una variable como perteneciente a determinado dominio y valen las mismas consideraciones mencionadas.

En la fig.2 correspondiente a la estructura de la transacción “Proveedores”, se puede ver que el atributo *PrvNom* pertenece al dominio “Nombres” (al que asignaremos el tipo de datos *Character(25)*), y que el atributo *PrvTotPed* pertenece al dominio “Importes” (al que asignaremos el tipo de datos *Numeric(9,2)*, donde el 2 corresponde a las posiciones decimales)

Atributos Clave

Es necesario definir cuál es el atributo o conjunto de atributos que identifican a la transacción, es decir, aquel o aquellos atributos cuyos valores son únicos.

Si en la realidad que estamos modelando un proveedor se identifica por su código, entonces el atributo *PrvCod* será el identificador de la transacción “Proveedores”.
Esto queda expresado en GeneXus mediante el símbolo de llave a la izquierda del atributo en la estructura de la transacción, como puede observarse claramente en la fig.2.

Es común encontrar como notación para representar en papel la estructura de una transacción (tanto en libros, manuales de GeneXus, help, etc.) una lista de atributos, donde aquellos que constituyen el identificador aparecen sucedidos del símbolo asterisco. Así, por ejemplo, denotaremos a la estructura de la transacción “Proveedores” de la siguiente manera:

*PrvCod**
PrvNom
PrvDir
PrvTotPed

Aquí estamos representando que el identificador de la transacción “Proveedores” es el atributo *PrdCod*, es decir, no podrán haber dos proveedores con el mismo código.
En tiempo de ejecución, se realizará este chequeo de unicidad automáticamente.

Con respecto a los identificadores:

- Toda transacción debe tener un identificador.
- Los identificadores tienen valor desde un principio (por ejemplo cuando se crea un proveedor nuevo se debe saber cuál será su *PrvCod*)
- No cambian de valor. Por ejemplo al proveedor con código 123 no se lo puede cambiar para que tenga código 234.

En los casos en los cuales no se puede determinar un identificador a partir de los atributos que representan la información relevante del objeto de la realidad, se debe optar por crear un atributo artificial (no existente en la realidad), cuyo valor sea asignado automáticamente por el sistema.

Relación entre la estructura y el modelo de datos

GeneXus utiliza la estructura de la transacción para capturar el conocimiento necesario para definir cuál es el modelo de datos correspondiente.

En particular de la estructura anterior GeneXus infiere que:

- No existen dos proveedores con el mismo *PrvCod*.
- Para cada *PrvCod* existe solo un valor de *PrvNom*, *PrvDir* y *PrvTotPed*.

y con esta información va construyendo el modelo de datos.

En este caso GeneXus determina que debe crear una tabla que tendrá por defecto el mismo nombre que la transacción, y que estará conformada por los cuatro atributos anteriores, siendo *PrvCod* la clave primaria de la tabla:

PROVEEDORES	*PrvCod*	*PrvNom*	*PrvDir*	*PrvTotPed*

A lo largo de este libro utilizaremos esta notación para las tablas, donde los atributos que conforman la clave primaria aparecen subrayados.

Diremos que la transacción "Proveedores" tiene asociada la tabla PROVEEDORES en el entendido de que cuando se ingresen, modifiquen o eliminen datos por medio de la transacción, éstos se estarán ingresando, modificando o eliminando físicamente en la tabla asociada.

Asimismo GeneXus creará automáticamente un índice por clave primaria, al que llamará IPROVEEDORES, que será utilizado para realizar eficientemente el control de unicidad de la clave primaria y controles de integridad referencial, como veremos un poco más adelante.

El nombre de la tabla y el del índice son asignados automáticamente por GeneXus utilizando el nombre de la transacción, pero pueden ser modificados por el analista cuando así lo desee.

Otras transacciones simples de nuestra aplicación

Además de la transacción de proveedores, tendremos transacciones para representar y manejar la información relevante de los analistas de compras, de los artículos y de los pedidos.

Las estructuras de las dos primeras serán:

Analistas:

*AnlNro**	Número de analista
AnlNom	Nombre de analista

Artículos:

*ArtCod**	Código de artículo
ArtDsc	Descripción del articulo
ArtCnt	Cantidad en stock
ArtFchUltCmp	Fecha última compra
ArtPrcUltCmp	Precio última compra
ArtUnd	Unidad del artículo
ArtTam	Tamaño
ArtFlgDsp	Disponibilidad

que tendrán asociadas las tablas ANALISTAS y ARTICULOS respectivamente:

ANALISTAS	*AnlNro*	*AnlNom*

ARTICULOS	*ArtCod*	*ArtDsc*	*ArtCnt*	*ArtFchUltCmp*
	ArtPrcUltCmp	*ArtUnd*	*ArtTam*	*ArtFlgDsp*

Como podemos apreciar en estas transacciones, todos los atributos de la estructura están presentes en la tabla asociada. Sin embargo esto no es lo más frecuente, como veremos en

breve, cuando definamos la transacción correspondiente a los pedidos.

A partir de la estructura de una transacción, GeneXus crea por defecto un form GUI-Windows y un form Web para la transacción, los cuales constituirán su interfaz con el usuario en ambiente Windows y Web respectivamente.

Definición de la transacción de pedidos

Consideremos ahora la transacción para manejar la información relativa a los pedidos. El formulario preexistente de pedidos es:

Pedido :	1		Fecha :	02/02/01
Analista :	21	Juan Gómez		
Proveedor :	125	ABC Inc.		
Código	Descripción	Cantidad	Precio	Importe
321	Aspirinas	100	30	3000
567	Flogene	120	50	6000
			Total	9000

Los atributos que integran el cabezal de la transacción son (dejando momentáneamente de lado la información de los artículos del pedido):

*PedNro** Número del pedido
PedFch Fecha del pedido
AnlNro
AnlNom
PrvCod
PrvNom
PedTot Total del pedido

donde *PedNro* es el identificador de un pedido.

Esta transacción tiene algunas características interesantes a destacar: tanto *AnlNro* y *AnlNom*, como *PrvCod* y *PrvNom* son atributos que están presentes en otras transacciones. De esta manera estamos indicando que existe una relación entre los “Analistas” y los “Pedidos” y también entre los “Proveedores” y los “Pedidos”. En particular aquí estamos indicando que un pedido solo tiene UN analista y UN proveedor.

Se tiene así que la forma de indicar a GeneXus la relación entre las distintas transacciones se basa en los nombres de los atributos.

Reglas para nombres de atributos

Se debe poner el mismo nombre al mismo atributo, en todas las transacciones en las que éste se encuentre[4].

De esta forma, al nombre del proveedor se lo denomina *PrvNom* tanto en la transacción de proveedores como en la de pedidos.

[4] En el curso GeneXus se estudian casos en los que esto no es posible, y se introduce entonces el concepto de “subtipo” para poder nombrar de otra forma a un atributo, pero especificando que se trata del mismo concepto. No trataremos este tema en el presente libro.

Se deben poner nombres distintos a atributos conceptualmente distintos, aunque tengan dominios iguales. Por ejemplo, el nombre del proveedor y el nombre del analista tienen el mismo dominio (son del tipo Character de largo 25), pero se refieren a datos diferentes, por lo tanto se deben llamar diferente: *PrvNom* y *AnlNom*.

Integridad referencial en las transacciones

Cuando se define la estructura de una transacción no se está describiendo exactamente la estructura de una tabla, sino los datos que se necesitan en la **pantalla** (form), en las **reglas** o en los **eventos** de la transacción.

Por ejemplo, si bien *AnlNom* aparece en la estructura de la transacción "Pedidos", este atributo no pertenecerá a la tabla correspondiente a la transacción. Lo incluimos en la estructura pues queremos que esté presente en la pantalla (form) de "Pedidos".

La tabla asociada al cabezal del "Pedido" será:

PEDIDOS	*PedNro*	*PedFch*	*AnlNro*	*PrvCod*	*PedTot*

Esta tabla será creada automáticamente por GeneXus, junto con los siguientes índices:

- IPEDIDOS (*PedNro*) Índice por clave Primaria
- IPEDIDOS1 (*AnlNro*) Índice por clave Foránea
- IPEDIDOS2 (*PrvCod*) Índice por clave Foránea

Observar que *PrvNom* no se encuentra en la tabla PEDIDOS, así como tampoco *AnlNom*. Diremos que GeneXus *normalizó* y que existe una relación entre la tabla PROVEEDORES, que tiene a *PrvCod* como clave primaria, y la tabla PEDIDOS que lo tiene como clave foránea. La relación, que llamaremos de integridad referencial, determina que:

- Para insertar o actualizar un registro en la tabla PEDIDOS debe existir el *PrvCod* correspondiente en la tabla PROVEEDORES.
- Cuando se elimina un registro de la tabla PROVEEDORES no deben haber registros en la tabla PEDIDOS con el valor del *PrvCod* a eliminar.

Estos controles de integridad son **generados automáticamente** por GeneXus en las transacciones.

Así, en la transacción "Pedidos", cuando en ejecución el usuario asigne un valor al atributo *PrvCod*, se validará que éste exista en la tabla PROVEEDORES.

Para ayudar al usuario de manera tal que no tenga que recordar de memoria los valores válidos, GeneXus genera una pantalla de selección, conocida como "Prompt" o "Lista de selección", que permite visualizar los proveedores existentes y seleccionar uno de ellos, para asociarlo al pedido, como veremos un poco más adelante (fig.9).

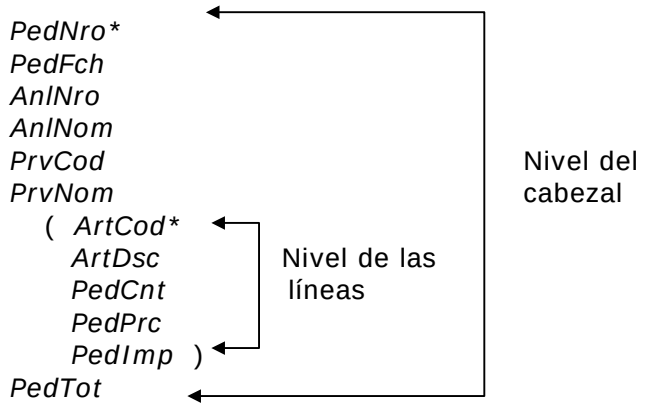
Niveles de una transacción

Volviendo al ejemplo, para terminar de diseñar la transacción “Pedidos” se debe incorporar en la estructura la información relativa a los artículos del pedido (las líneas). Si definimos la estructura de la siguiente manera:

*PedNro**
PedFch
PrvCod
PrvNom
AnlNro
AnlNom
ArtCod
ArtDsc
PedCnt Cantidad pedida
PedPrc Precio pedido
PedImp Importe por el artículo
PedTot

no estaremos modelando correctamente la realidad, ya que aquí se está representando que para cada pedido existe solo UN artículo. Sin embargo, si observamos el formulario preexistente, vemos claramente que un pedido está compuesto de muchos artículos.

La estructura correcta es, pues:



donde el identificador es *PedNro* y para cada número de pedido existe solo una fecha, un número y nombre de analista, un código y nombre de proveedor, y un solo total, pero existen **muchos** artículos, con su descripción, cantidad pedida, precio e importe.

En esta notación, los paréntesis indican que para cada pedido existen muchos artículos. Diremos que cada grupo de atributos que se encuentra encerrado entre paréntesis pertenece a un **NIVEL** de la transacción.

Cabe destacar que el primer nivel queda implícito y no necesita un juego de paréntesis.

Así, la transacción “Pedidos” tiene dos niveles:

- *PedNro*, *PedFch*, *AnlNro*, *AnlNom, PrvCod*, *PrvNom* y *PedTot* pertenecen al primer nivel, y
- *ArtCod*, *ArtDsc*, *PedCnt*, *PedPrc* y *PedImp* pertenecen al segundo nivel.

En GeneXus el segundo nivel queda representado como se muestra en la siguiente figura:

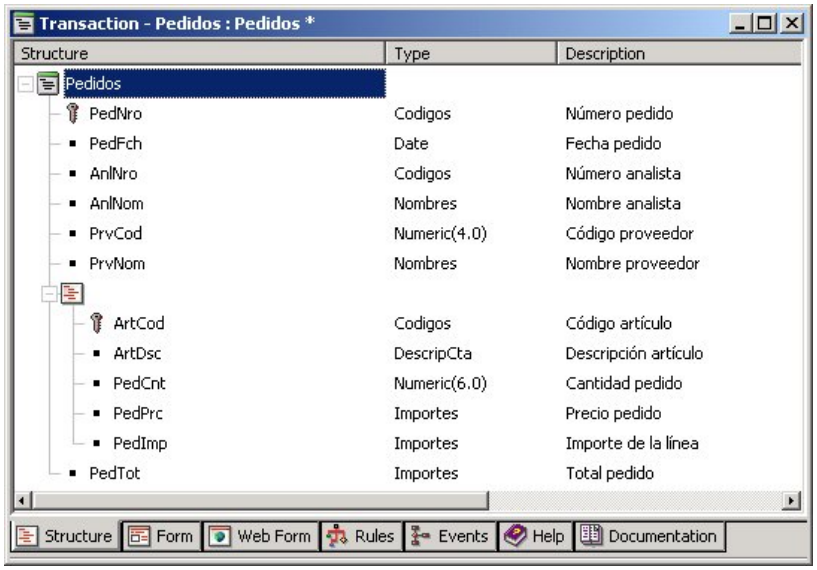


Fig. 6: Estructura de "Pedidos" en GeneXus

Una transacción puede tener varios niveles y ellos pueden estar anidados o ser paralelos entre sí. Por ejemplo, si el pedido tuviera varias fechas de entrega podemos definir:

```
PedNro*
PedFch
AnlNro
AnlNom
PrvCod
PrvNom
  ( ArtCod*
    ArtDsc
    PedCnt
    PedPrc
    PedImp  )
  ( PedFchEnt*          Fecha de entrega
    PedImpPag  )        Importe a pagar
PedTot
```

Con esta estructura se define que un pedido tiene muchos artículos y muchas entregas, pero no hay relación directa entre ellos (a no ser el hecho de pertenecer al mismo pedido).

Si por el contrario, las fechas de entrega son por artículo (cada artículo tiene sus fechas de entrega particulares), la estructura correcta es:

*PedNro**
PedFch
AnlNro
AnlNom
PrvCod
PrvNom
(*ArtCod**
ArtDsc
PedCnt
PedPrc
PedImp
(*PedFchEnt**
PedImpPag))
PedTot

Cada nivel de la transacción tiene un identificador, es decir, un conjunto de atributos que determinan la unicidad de los datos del nivel.

Así, en el pedido se definió que *ArtCod* es el identificador del segundo nivel, lo cual significa que dentro de cada pedido no pueden existir dos líneas con el mismo valor de *ArtCod.*

En consecuencia, el siguiente pedido no se podría ingresar:

Pedido :	1		Fecha :	02/02/01
Analista :	21 Juan Gómez			
Proveedor :	125 ABC Inc.			
Código	Descripción	Cantidad	Precio	Importe
321	Aspirinas	100	30	3000
321	Aspirinas	120	50	6000
			Total	9000

porque en él existen dos líneas del pedido con el mismo *ArtCod*.

Si se desea representar pedidos como el anterior, en los que puedan ingresarse varias líneas correspondientes al mismo artículo, entonces el atributo *ArtCod* ya no puede ser identificador del segundo nivel. Se debe definir, pues, otro identificador. Para ello agregamos un atributo artificial, *PedLinNro* en el segundo nivel, que contendrá el número de línea:

*PedNro**
PedFch
AnlNro
AnlNom
PrvCod
PrvNom
(*PedLinNro**
ArtCod
ArtDsc
PedCnt
PedPrc
PedImp)
PedTot

donde el valor de *PedLinNro* puede ser digitado por el usuario o puede incluirse una regla en la transacción para que sea asignado automáticamente por el programa (regla "Serial" que veremos más adelante)[5].
De esta forma *ArtCod* deja de ser identificador y podrá repetirse para dos líneas distintas del mismo pedido.

Cada nivel de la transacción tiene una tabla asociada. Para la transacción "Pedidos" con la última estructura vista, GeneXus creará 2 tablas:

PEDIDOS	*PedNro*	*PedFch*	*AnlNro*	*PrvCod*	*PedTot*

PEDIDOS1	*PedNro*	*PedLinNro*	*ArtCod*	*PedCnt*	*PedPrc*	*PedImp*

La clave primaria de las tablas asociadas a los niveles subordinados es la concatenación de los identificadores de los niveles superiores (*PedNro*) más el identificador del nivel (*PedLinNro*). Para el caso del primer nivel, no existen niveles superiores, por lo que la clave primaria se corresponde exactamente con el identificador del nivel.

Con solo definir las estructuras de las transacciones, tal como lo hemos hecho, GeneXus determina las tablas que debe crear, las claves primarias (a partir de los identificadores) y foráneas (a partir de los nombres de atributos) de tales tablas, y seguidamente, para cada una, define y crea índices por esas claves encontradas, para poder hacer eficientes los controles de integridad referencial.

Además, GeneXus provee la posibilidad de definir **claves candidatas**, es decir, conjuntos de atributos de una tabla, cuyos valores deben ser únicos. Toda clave primaria es, en particular, una clave candidata. La forma de definir claves candidatas es creando un *índice de usuario* para ese conjunto de atributos, y definiéndolo como 'unique' (en lugar de 'duplicate'). Con esta definición, GeneXus no solo realizará en las transacciones el control de unicidad para la clave primaria (así como los controles de integridad referencial), sino que además controlará la unicidad de cualquier clave candidata definida mediante estos índices.

La elección de los identificadores es una tarea importante, que puede simplificar o complicar la implementación de un sistema. Supongamos, por ejemplo, que en el pedido no se admite que exista más de una línea con el mismo *ArtCod*. La forma natural de implementar esto es definiendo a *ArtCod* como identificador del segundo nivel. Ahora bien, si por alguna razón se definió a *PedLinNro* como el identificador, ya no tendremos ese control automático y deberemos recurrir a una de dos soluciones:

- escribir un procedimiento para realizarlo, o
- crear un índice de usuario compuesto por los atributos *PedNro* y *ArtCod* en ese orden, y definirlo como 'unique'.

La creación de todo índice tiene costos en cuanto a la performance y al almacenamiento, puesto que hay que mantenerlo actualizado.

Cuantas más reglas de la realidad se puedan reflejar en la estructura, menor será la cantidad de procesos que se deben implementar, ganando así en simplicidad y flexibilidad.

Para el resto de la exposición, nos quedaremos con la primer estructura vista para la transacción "Pedidos", en la que el código de artículo no podía repetirse (es decir, el identificador del segundo nivel es *ArtCod*)

[5] Aquí no es válido setear la propiedad Autonumber del atributo *PedLinNro* en True, para que sea autonumerado por el sistema, pues este atributo no es por sí solo una clave primaria

Tipos de relación entre los objetos

Cuando los usuarios están describiendo la aplicación, es común preguntarles qué tipo de relación existe entre los distintos objetos. Por ejemplo, cuál es la relación entre los pedidos y los proveedores:

- Un proveedor tiene muchos pedidos
- Un pedido tiene un solo proveedor

A una relación que cumple con los requerimientos anteriores le llamamos N-1 (y leemos: "Pedidos y Proveedores tienen una relación N a 1").

El primer requerimiento podría representarse como:

*PrvCod**
PrvNom
....
(*PedNro**
....
)

y el segundo, con las dos transacciones siguientes:

*PedNro**	*PrvCod**
....	*PrvNom*
PrvCod	*PrvDir*
PrvNom	*PrvTotPed*
....	

Observar que el primer diseño representa el primer requerimiento, pero no el segundo, ya que permite que un mismo número de pedido esté asociado a distintos códigos de proveedor. Con este diseño se está representando una relación N-M, que no es la que refleja la realidad de nuestra aplicación de ejemplo.

El segundo diseño, en cambio, representa ambos requerimientos, y por tanto es el adecuado y es el que hemos presentado, por tanto, en la exposición anterior.

De la misma forma, si recurrimos a la realidad, obtenemos que la relación entre los pedidos y los artículos es la siguiente:

- Un pedido tiene muchos artículos.
- Un artículo puede estar en muchos pedidos.

En este caso decimos que la relación entre pedidos y artículos es N-M.

Las dos estructuras posibles que modelan esta relación son:

*PedNro**		*ArtCod**
PedFch	ó	*ArtDsc*
PrvCod		*(PedNro**
PrvNom		*PedFch*
AnlNro		*PrvCod*
AnlNom		*PrvNom*
*(ArtCod**		*AnlNro*
ArtDsc		*AnlNom*
PedCnt		*PedTot*
PedPrc		*PedCnt*

Se debe definir solo una de las dos estructuras. En este caso la correcta es la primera, porque el usuario ingresa los pedidos con sus artículos y no para cada artículo qué pedidos tiene.

Forms

Para cada transacción, GeneXus crea un form GUI-Windows y un form Web, los cuales serán la interfaz con el usuario, en ambiente windows y web respectivamente.

Ambos forms son creados por defecto por GeneXus al momento de salvar la estructura de la transacción, y contienen todos los atributos incluidos en la misma, con sus respectivas descripciones, además de algunos botones.

Si bien son creados por defecto, es posible modificarlos para dejarlos más vistosos, cambiar por ejemplo controles de tipo edit a otros tipos de controles, agregar y/o quitar botones, etc.

El Form GUI Windows por defecto para la transacción “Proveedores” es:

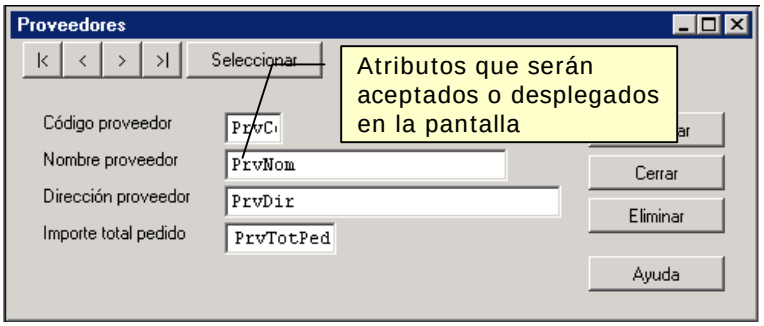


Fig. 7: Form GUI - transacción “Proveedores”

Utilizando esta pantalla el usuario final puede ingresar, modificar y eliminar proveedores. Para ello la pantalla tiene un **modo**, que indica cuál es la operación que se está realizando (Insert, Update, Delete o Display) y el usuario puede pasarse de un modo a otro sin tener que abandonar la pantalla.

GeneXus genera varios tipos de diálogo para las transacciones. Entre ellos: diálogo campo a campo y full-screen.

El tipo de diálogo utilizado dependerá tanto del ambiente (Windows o Web), como de la plataforma de implementación elegida.

Diálogo campo a campo

En este diálogo, cada vez que se digita un valor en un campo, se controla inmediatamente su validez. Por ejemplo, si se trata de un atributo de tipo Date, una vez que el usuario ingresa un valor en el mismo, y abandona el campo, inmediatamente se controlará que el valor corresponda a una fecha válida.

Diálogo full-screen

En este diálogo a pantalla completa, primero se aceptan todos los campos de la pantalla y luego se realizan todas las validaciones. En ambiente Web no hay otra opción que trabajar de esta forma, ya que como los datos viajan del navegador a un servidor Web, y viceversa, es inviable trabajar con el diálogo campo a campo, por los factores tiempo y tráfico.

Botones de posicionamiento

Tanto los forms GUI-Win como Web son inicializados por defecto con botones de posicionamiento.
Si observamos el form GUI de la transacción "Proveedores" (fig.7), vemos en el borde superior izquierdo botones que permiten seleccionar el primer proveedor: |<, el siguiente a partir del que se está mostrando en pantalla: >, elegir uno en particular: Seleccionar, etc.
También en el form Web de la figura 4 vemos estos controles, aunque con otro aspecto.

Atributos de entrada y atributos de salida

Consideremos nuevamente la transacción "Pedidos". El form GUI por defecto es:

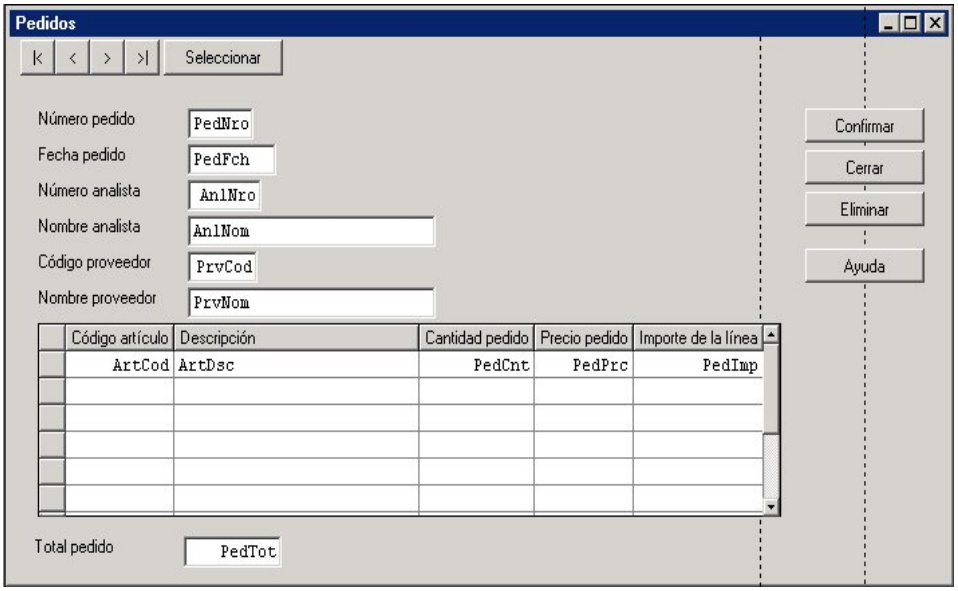


Fig. 8: Form GUI por defecto de la transacción "Pedidos"

Observemos que aparece una grilla que corresponde a los datos del segundo nivel, dado que por cada pedido, se deben poder ingresar varias líneas.

Aquí hay algunos atributos que deben ser digitados (son atributos de entrada), por ejemplo *PedNro* y *PrvCod* y otros que no, como *PrvNom* (son atributos de salida, es decir, solo se muestra su valor).

GeneXus automáticamente determina qué atributos son aceptados y de cuáles solo se muestra su valor, siguiendo las reglas:

- Solo pueden ser aceptados los atributos que se encuentran físicamente en la **tabla asociada a cada nivel** (en este caso son los atributos de la tabla PEDIDOS: *PedNro*, *PedFch*, *PrvCod*, *AnlNro* y *PedTot* y los de la tabla PEDIDOS1: *ArtCod*, *PedCnt*, *PedPrc* y *PedImp*)
- Los atributos definidos como fórmulas no son aceptados (estudiaremos los atributos fórmula un poco más adelante en este capítulo)
- En modo Update no se aceptan los identificadores.
- Los atributos que tienen una regla "Noaccept" no son aceptados (estudiaremos esta y otras reglas más adelante en este capítulo).

Facilidad de Prompt de Selección

Para los atributos que están involucrados en la integridad referencial de la tabla asociada (atributos que son claves foráneas) se genera la facilidad de *Prompt*, que permite visualizar el conjunto de valores válidos para esos atributos, de forma tal de poder seleccionar uno sin tener que recordarlo de memoria.

Por ejemplo, en la transacción "Pedidos" generada en ambiente Win, presionando sobre el atributo *PrvCod* una tecla especial (por defecto F4) se abre la siguiente pantalla que permite visualizar todos los proveedores, para seleccionar uno:

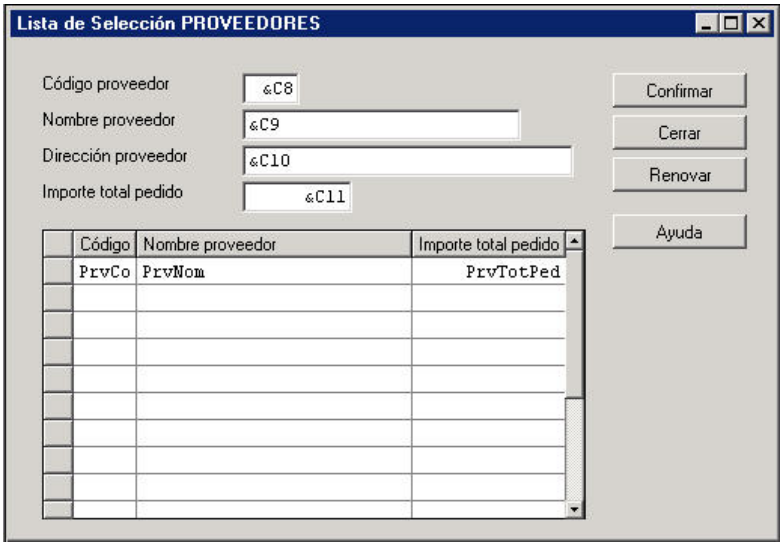


Fig. 9: "Lista de Selección PROVEEDORES"

Los campos que aparecen en la parte superior de la pantalla corresponden a variables que se utilizan para poder posicionarse en la grilla sobre el primer proveedor que tenga alguno de los valores ingresados por el usario en esas variables, y, de esta forma, seleccionarlo.
Esta "lista de selección" es un objeto creado automáticamente por GeneXus, de tipo work panel, pues hemos supuesto que estamos trabajando en ambiente Win. Si el ambiente fuera Web, GeneXus hubiera creado un objeto análogo, pero específico para este otro ambiente: el web panel. Luego de creado puede ser modificado por el analista, como lo haría con cualquier work panel (web panel) creado por él.
En toda transacción en la que el atributo *PrvCod* sea clave foránea, el usuario podrá acceder a esta pantalla, para seleccionar un valor válido. Esta pantalla es abierta también cuando el usuario presiona el botón "Seleccionar" en la transacción "Proveedores" (ver figuras 3 y 4).

Editor de forms

Los objetos GeneXus que implementan una interacción con el usuario, tanto para ingresar como para desplegar datos, lo hacen a través de una pantalla. Las transacciones y los work panels –en ambiente Win- y las transacciones y los web panels –en ambiente Web- son objetos de este tipo, y por tanto tienen forms asociados.
A los efectos de permitir diseñar tales forms se provee de un editor para ambiente Win y otro para Web, que permiten insertar y modificar los distintos controles que componen esas pantallas.

Controles

Un form está compuesto de controles. Podemos definir a un **control** como un área de la interfaz con el usuario, que tiene una forma y un comportamiento determinado.

Existen distintos controles, entre ellos:

- **Form**: Un form puede verse en sí mismo como un control.
- **Texto**: Permite colocar texto fijo (por ejemplo las descripciones de los atributos: "Nombre proveedor", "Código de artículo", o cualquier otro texto).
- **Atributo/Variable**: Permite colocar atributos o variables.
- **Línea**: Con este control se dibujan líneas horizontales o verticales.
- **Recuadro**: Permite definir recuadros de distintos tamaños y formas.
- **Grid**: Permite definir grillas de datos.
- **Botón**: Permite incluir botones en los forms.
- **Bitmap**: Permite definir bitmaps estáticos.

Los controles anteriores están disponibles tanto para ser utilizados en interfaz GUI-Windows como Web.
A su vez el "tab control", es un control que solo está disponible para utilizarlo en interfaz GUI-Windows, mientras que otros controles solo pueden utilizarse en interfaz Web (text blocks, tablas, grids freestyle, etc.).

La edición del form GUI-Windows se realiza seleccionando la solapa "Form" del objeto. En la figura siguiente mostramos el form GUI-Windows de la transacción "Articulos" que hemos personalizado, cambiando el **tipo de control** de algunos de los **controles atributo**:

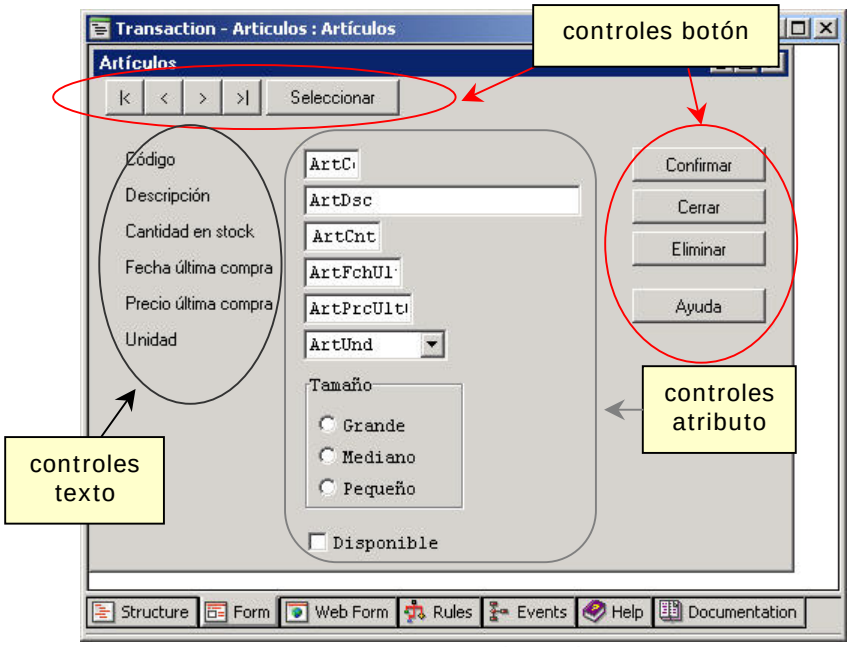


Fig. 10: Form transacción "Artículos"

Los atributos *ArtUnd*, *ArtTam y ArtFlgDsp* no aparecen en el form como los atributos convencionales (de tipo *edit*). *ArtUnd* lo definimos como un **Combo Box**, *ArtTam* como un **Radio Button** y *ArtFlgDsp* como un **Check Box**.

Tipos de controles atributo/variable

Edit

Normalmente los atributos tienen un rango de valores muy grande, por ejemplo: un nombre, un precio, etc. En estos casos se le permite al usuario entrar el valor del atributo y el sistema se encarga de validarlo. A estos tipos de controles se los llama "Edit Box", o más simplemente "Edit". *ArtCod*, *ArtDsc*, etc. en la figura anterior son ejemplos de controles Edit.
Sin embargo, existen atributos que tienen un rango de valores pequeño y que pueden ser desplegados de antemano para que el usuario seleccione uno. De esta forma controlamos que se ingresen solo valores válidos. Estos tipos de controles son los que veremos a continuación.

Check Box

Es usado para aquellos atributos que tienen solo dos valores posibles. En nuestro ejemplo, para señalar si existe disponibilidad del artículo, el atributo tomará uno de los valores 'S' o 'N'. Existe una única descripción (Disponible) y en caso que este campo esté seleccionado, el valor será el especificado por el analista para este caso ('S' en nuestro ejemplo) y en caso contrario será el otro valor especificado ('N' en el ejemplo).

Radio Button

Los 'Radio Button', en cambio, pueden tener más de dos valores. Todos los valores se despliegan en el form (en realidad se despliegan sus descripciones, el valor que se almacena es manejado internamente) y solo se puede seleccionar uno. En el ejemplo, al "tamaño del artículo", *ArtTam*, lo definimos como un Radio Button.

Combo Box

Es generalmente usado para aquellos atributos que tienen un rango grande de valores, que hace poco práctico utilizar un Radio Button para su manejo. Se despliega un campo de tipo Edit y presionando un botón que se encuentra a la derecha del campo se despliega una lista con todos los valores válidos. No es recomendable utilizar este tipo de control como lista de selección de atributos que puedan ser leídos de una tabla. Para estos casos se usan los Dynamic Combobox.

Dynamic Combobox

Un Dynamic Combobox es un tipo de control similar al Combo Box. La forma de operación es similar, salvo que los valores desplegados no son estáticos (ingresados por el analista como valores fijos) sino que son descripciones leídas de una determinada tabla de la base de datos.

List Box

Este tipo de control tiene asociada una colección de ítems. Cada ítem tiene asociado un par <valor, descripción>. El control da la posibilidad de seleccionar un solo ítem a la vez. El atributo o variable toma el valor en el momento que se selecciona el ítem. La selección se realiza dando clic con el mouse en un ítem o con las flechas del teclado. El control *List Box* puede verse como un *Combo Box* abierto, es decir, que los valores (ítems) se muestran desplegados en una lista, siendo la definición y funcionalidad del List Box totalmente análoga a la del *Combo Box.*

Dynamic List Box

Este tipo de control tiene asociada una colección de ítems. Cada ítem tiene asociado un par <valor, descripción>. La colección de ítems se carga desde una tabla de la base de datos en tiempo de ejecución. En tiempo de diseño se asocian dos atributos al Dynamic List Box, uno al valor que tendrá el ítem y el otro a la descripción que éste tomará. Ambos atributos deben pertenecer a la misma tabla.
En tiempo de especificación se determina la tabla desde la cual se traerán los valores y las descripciones.

Personalizando el form de "Pedidos"

En la fig.8 vimos el form que crea por defecto GeneXus para la transacción "Pedidos".

En dicho form podemos ver que por cada atributo del primer nivel de la estructura, se inserta un control texto y un control atributo. El control texto corresponde a lo que tenga configurado el atributo en la propiedad "Title", como mencionamos cuando estudiamos los atributos.

Para los atributos del segundo nivel se diseña una grilla (grid), donde aparece una columna por cada atributo de este segundo nivel, siendo el título de la columna el valor de la propiedad "Column title" del atributo correspondiente.

Trabajando con el editor de forms, podemos personalizar la pantalla anterior, de manera tal que nos quede:

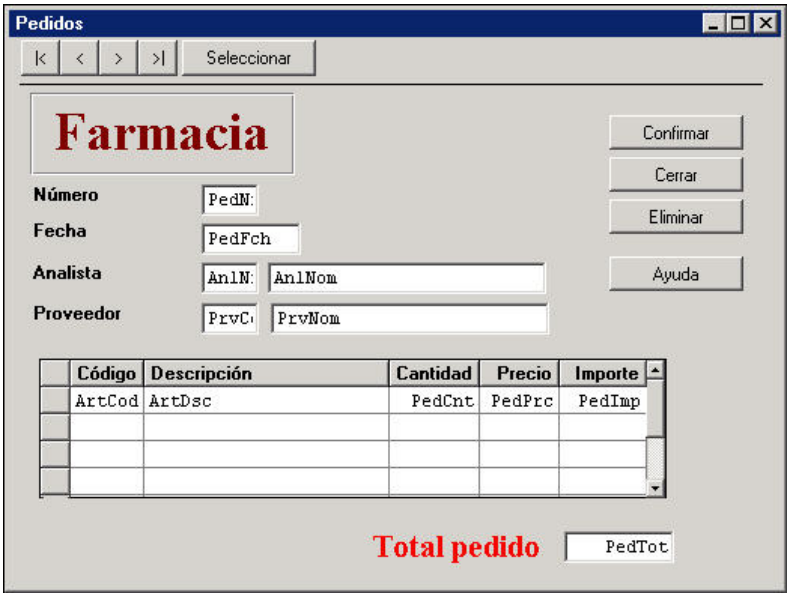


Fig. 11: Form personalizado de transacción "Pedidos"

El form está delimitado por un marco de ventana (frame) cuyo título es la descripción de la transacción. Puede verse como el control Form. Dentro figurarán los distintos tipos de controles de edición del form, que acabamos de mencionar.

Cada control tiene una serie de propiedades (ancho de línea, color, color de fondo, font, etc.), que dependen del tipo de control, y cuyos valores pueden fijarse en forma independiente. Unas páginas más adelante estudiaremos las propiedades de algunos de los controles mencionados.

Paletas de herramientas

Mientras se está editando un form, están disponibles varias paletas de herramientas, en forma de ventanas adicionales.
Tenemos una paleta que permite insertar controles:

Fig. 12: Paleta de herramientas "Controls"

y otra que permite alinearlos, copiar el tamaño de uno en otro, etc:

Fig. 13: Paleta de herramientas "Form Editor"

Uso de las Herramientas

Sobre los objetos seleccionados con el puntero vamos a poder realizar varias operaciones:

- Cambiar el tamaño y forma de un control.
- Edición de propiedades.
- Move Forward, Move Back, Move to Front y Move to Back. Estas operaciones nos van a permitir cambiar la capa en que se encuentra un control. Cada control se encuentra en una capa distinta del form, por lo que algunos están "más arriba" que otros. Cuando dos objetos se superpongan, el de más arriba será visible totalmente, mientras que el otro sólo en aquellos puntos en que no se encuentre "tapado".
- Move, Copy y Delete.

Propiedades de los controles

Vamos a ver las principales propiedades de algunos de los controles que podemos insertar en un form.

Atributo/Variable

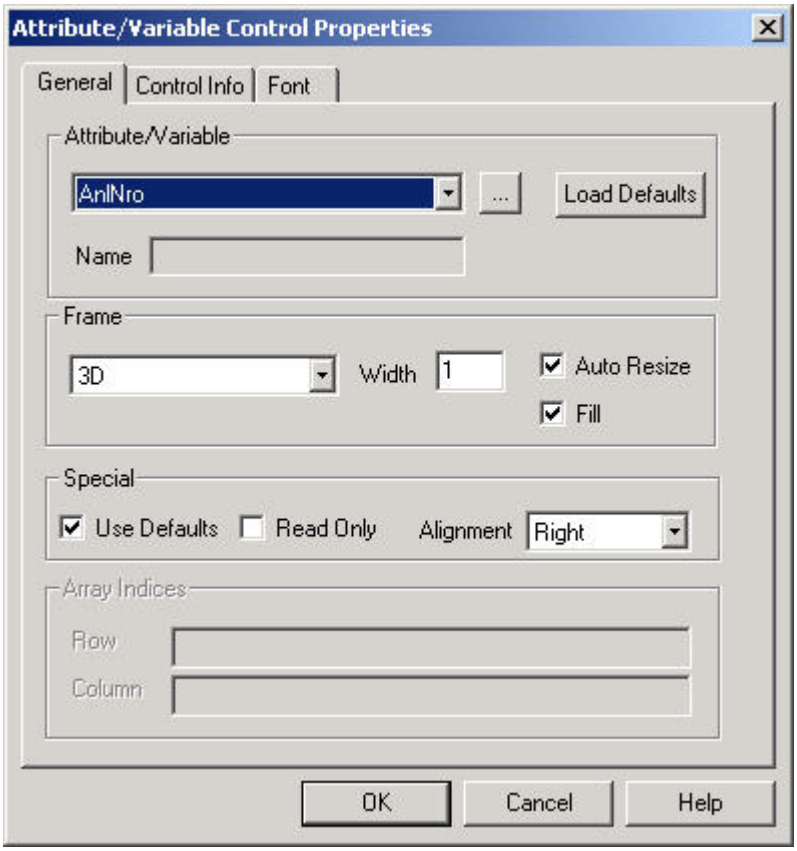


Fig. 14: Propiedades control atributo/variable

Solapa *General*

Atribute/Variable: Permite indicar el atributo o variable asociado al control.

Name: En esta opción se permite asignar un nombre al control que se está editando. Este nombre será usado luego para asociarle propiedades, eventos o métodos al control. Cuando se trata de un atributo, o de una variable escalar, aparece deshabilitado, dado que se le asocia al control el mismo nombre que tiene el atributo o variable. Esta propiedad se habilita en caso de tratarse de una variable no escalar (o sea, un vector o matriz).

En este último caso, se debe utilizar un control por cada elemento atómico de la variable no escalar. Es decir, si tenemos una variable llamada &numeros que es un vector de tipo numérico, con 3 elementos, para mostrar en el form a cada uno de ellos, se deberán insertar 3 controles. En la propiedad Atribute/Variable, de los 3 controles, se seleccionará la misma variable (&numeros), pero en la propidad Name de cada control, se pondrá diferente nombre (por ejemplo: uno, dos, tres). Además, en cada control se deberá indicar de qué elemento del vector se trata. Esto se hace con las propiedades Row y Column del grupo Array Indices. Aquí se indican expresiones para la fila y la columna de manera de que quede determinado el elemento.

Frame: Se puede indicar que el atributo o variable esté rodeado por un marco. Es posible indicar el tipo: None, Single o 3D; el ancho de la/s línea/s (Width), si se pinta dentro de él o no (Fill) y si el tamaño y forma deben determinarse a partir del atributo (Auto Resize).

Array Indices: En caso de utilizar variables no escalares, se habilitan estas propiedades para indicar la fila y la columna de la variable no escalar, que determinan el elemento cuyo

contenido se quiere mostrar.

Solapa *Control Info*
Nos da la posibilidad de seleccionar el tipo de control, y de acuerdo a él, nos pide que seleccionemos otras características del mismo, así como nos permite seleccionar el *Fore Color* y el *Back Color* del control.

Control Type: En los forms generados un atributo podrá asociarse a distintos tipos de controles. Se puede seleccionar uno de los siguientes: Combo Box, Dynamic Combobox, Radio Button, Edit, List Box, Dynamic List Box y Check Box. El botón de *Setup* que aparece luego de seleccionar el *control type* permite definir características propias del tipo de control elegido.

Fore Color: Este botón es para seleccionar el color con el que se desplegará el valor del atributo y la/s línea/s del frame, si tuviera.

Back Color: Con este botón se puede seleccionar el color con el que se pinta el área asignada al atributo en la pantalla. Sólo tiene efecto si está presente la propiedad Fill.

Solapa *Font*
Permite seleccionar el font que se utilizará para el atributo o variable en la pantalla.

Texto

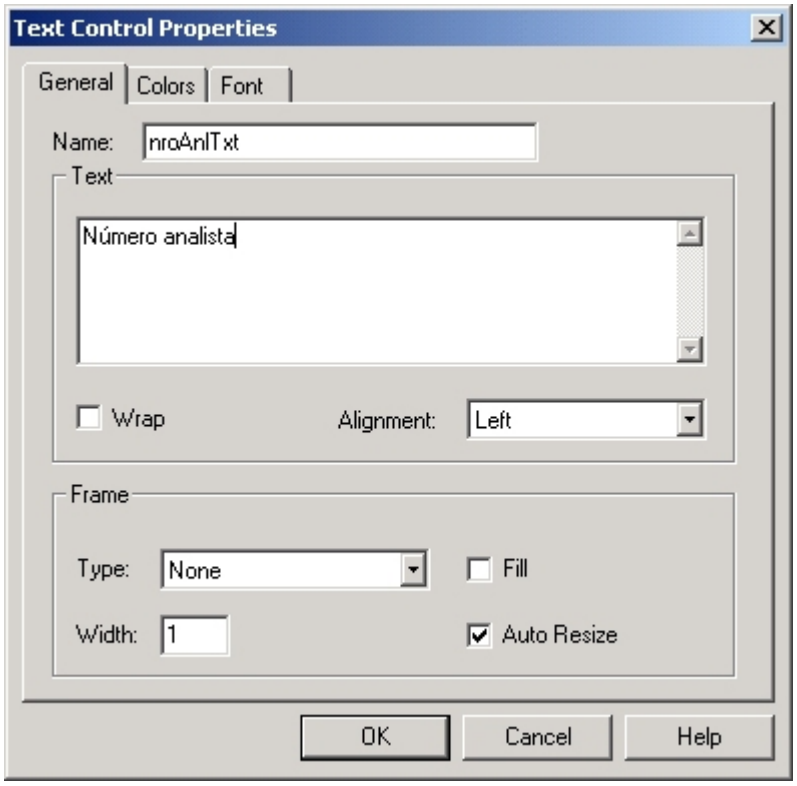


Fig. 15: Propiedades control texto

Text: Indica el texto que se quiere insertar en la pantalla. Puede consistir de una o más líneas.

Wrap: Indica si queremos que se ajuste el texto al tamaño del control, de forma tal que cuando se llegue al tope derecho, lo que siga se coloque en el siguiente renglón.

Alignment: El texto puede estar justificado a la izquierda (Left), derecha (Right) o centrado (Center).

Resto de las propiedades: Análogas al control Atributo/Variable.

Recuadro

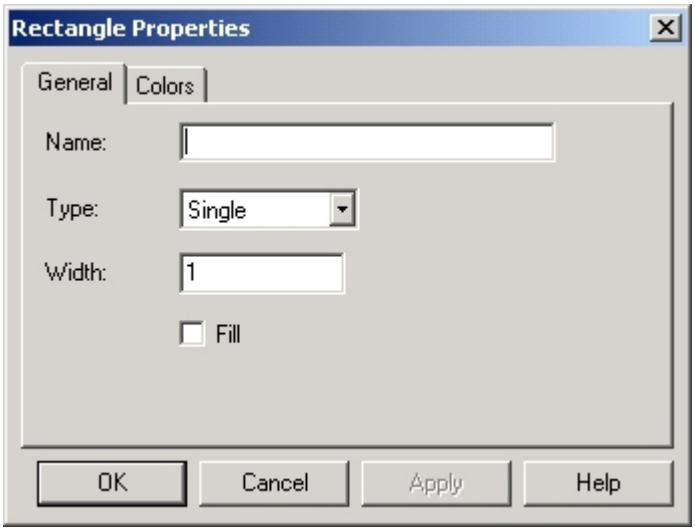


Fig. 16: Propiedades control rectángulo

Del combo box se selecciona el tipo del borde del recuadro: single, none (sin borde) o 3D.

Línea

Utiliza el mismo diálogo de edición de propiedades con la salvedad de que la opción Fill aparece deshabilitada.

Editor de Propiedades, Métodos y Eventos

De todos los controles utilizados en un form, a aquellos que tengan nombre asignado se les pueden asociar propiedades, métodos o eventos dinámicamente, por programación dentro del objeto. Por ejemplo: cambiarle el font a un texto, hacer un texto visible o invisible, deshabilitar un botón, etc. El tipo de propiedades/eventos/métodos que se pueden asociar a un control dependen del tipo de control.

Para acceder al diálogo donde poder asociar propiedades a los controles se usa la opción de menú **Insert/Property** (**Insert/Events** e **Insert/Methods** para los eventos y métodos respectivamente) cuando se están editando los eventos, reglas o subrutinas de la transacción. El diálogo que aparece es el siguiente:

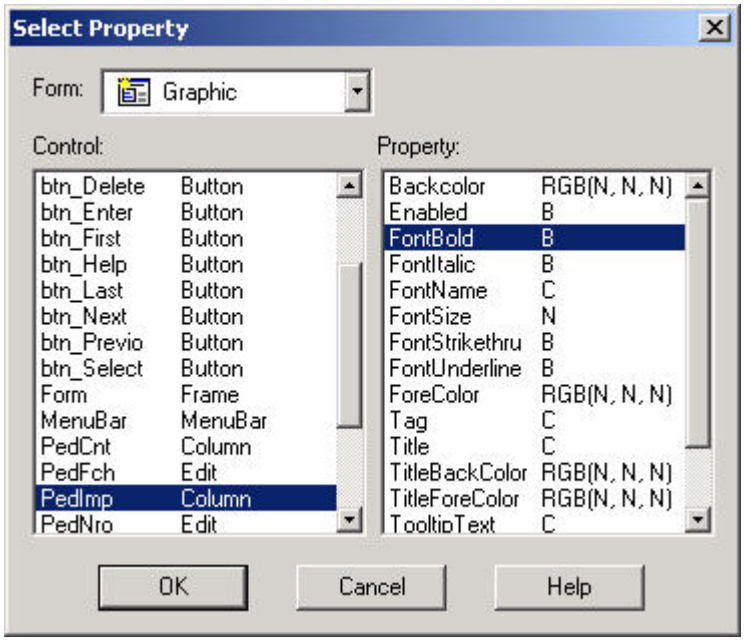


Fig. 17: Diálogo de selección de propiedades de controles

Así, por ejemplo, si en la transacción "Pedidos" queremos que el importe de una línea aparezca en negrita si supera los $1000, podemos escribir las siguientes reglas de la transacción:

PedImp.FontBold = 1 if *PedImp* > 1000;
PedImp.FontBold = 0 if *PedImp* <= 1000;

Fórmulas

Se dice que un atributo es una fórmula si su valor se puede calcular a partir del valor de otros atributos.

En el ejemplo, el atributo *PedImp* de la transacción "Pedidos" se puede calcular a partir de la cantidad pedida, *PedCnt* y del precio del producto, *PedPrc*:

*PedImp = PedCnt * PedPrc*

En cada transacción se puede definir qué atributos son fórmulas, pero esta definición es global, es decir, toda vez que se necesite el atributo se calcula la fórmula, tanto en transacciones como en los otros objetos GeneXus (reportes, work panels, etc.).

Existen distintos tipos de fórmulas:

- Horizontales
- Verticales
- Aggregate/Select

Fórmulas Horizontales

Una fórmula horizontal se define como una o varias expresiones aritméticas condicionales. Por ejemplo, si agregarmos un atributo en el primer nivel de la transacción "Pedidos", *PedDto*, que represente el descuento que se aplica sobre el total del pedido y sabemos que este descuento se calcula de la siguiente manera:

- Si el total del pedido es menor a 100 no se aplica descuento
- Si el total del pedido está en el rango 100-1000 entonces se aplica sobre éste un 10% de descuento
- Si el total del pedido es mayor a 1000 entonces se aplica un 20% de descuento

El atributo *PedDto* será definido como la fórmula horizontal:

PedDto =	*PedTot* * 0.10	if *PedTot* >= 100 and *PedTot* <= 1000;
	PedTot * 0.20	if *PedTot* > 1000;
	0	OTHERWISE;

En las expresiones se pueden utilizar los operadores aritméticos (+, -, *, /, ^), funciones del sistema (por ejemplo "today()" que devuelve la fecha del día), atributos, variables y constantes, y para los casos en donde el cálculo es muy complicado se puede llamar a una rutina que lo realice. Por ejemplo:

PedDto = udp('Pdesc', PedTot)

En donde *udp* es una función que implementa la comunicación entre objetos GeneXus, 'Pdesc' es el nombre del procedimiento invocado, *PedTot es un* parámetro que se envía y *PedDto* es el atributo que recibe el resultado que calcula el procedimiento.

El importe del pedido también es una fórmula horizontal:

*PedImp = PedCnt * PedPrc*

Vemos que en el cálculo aparecen involucrados dos atributos: *PedCnt* y *PedPrc*, pero no se indica en qué tablas se encuentran. GeneXus se encarga de buscar la manera de relacionar *PedCnt* y *PedPrc* para poder calcular la fórmula (en este caso es muy fácil porque *PedCnt* y *PedPrc* se encuentran en la misma tabla). En los casos en los que no es posible relacionar a los atributos GeneXus da el mensaje "No Triggered Actions" en el listado de navegación, del que hablaremos más adelante.

Ahora bien, ¿cuáles son los atributos que pueden estar involucrados?. En una fórmula horizontal los atributos de los cuales depende la fórmula deben estar en la **tabla extendida** del atributo que se está definiendo como fórmula (ver el concepto de tabla extendida en el anexo sobre modelos de datos relacionales).

Como con cualquier otro atributo, cuando se define uno de los atributos de la estructura de una transacción como fórmula, GeneXus, siguiendo los criterios de normalización, puede determinar directamente la tabla a la que está "asociado" ese atributo. A partir de ello encuentra su tabla extendida y si todos los atributos utilizados en la definición de la fórmula se encuentran en esa tabla extendida, entonces la fórmula estará correctamente definida, y será del tipo horizontal.

Fórmulas Verticales

Consideremos ahora el total del pedido, *PedTot.* Este se debe calcular sumando los importes, *PedImp*, de cada una de las líneas del pedido. Esta fórmula se expresa como:

PedTot = **SUM**(*PedImp*)

Es un caso de fórmula vertical. En este caso los atributos involucrados no se encuentran dentro de la tabla extendida de la fórmula. En el caso de *PedTot*, si observamos el modelo de datos –usando el diagrama de tablas de GeneXus, conocido como diagrama de Bachman:

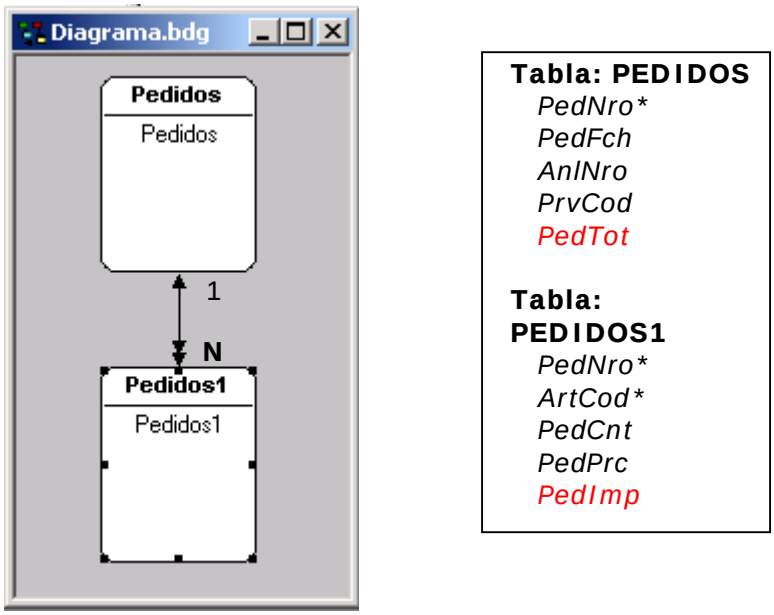


Fig. 18: Diagrama de Bachman de tablas PEDIDOS y PEDIDOS1

tenemos que *PedTot* está asociada a la tabla PEDIDOS y *PedImp* a la tabla PEDIDOS1, que es una tabla directamente **subordinada** a la tabla PEDIDOS.
Podemos utilizar también el navegador (browser) de GeneXus para determinar qué tablas están subordinadas y superordinadas a PEDIDOS (eligiendo la opción Subordinated Tables o Superordinated Tables del diálogo del browser, respectivamente):

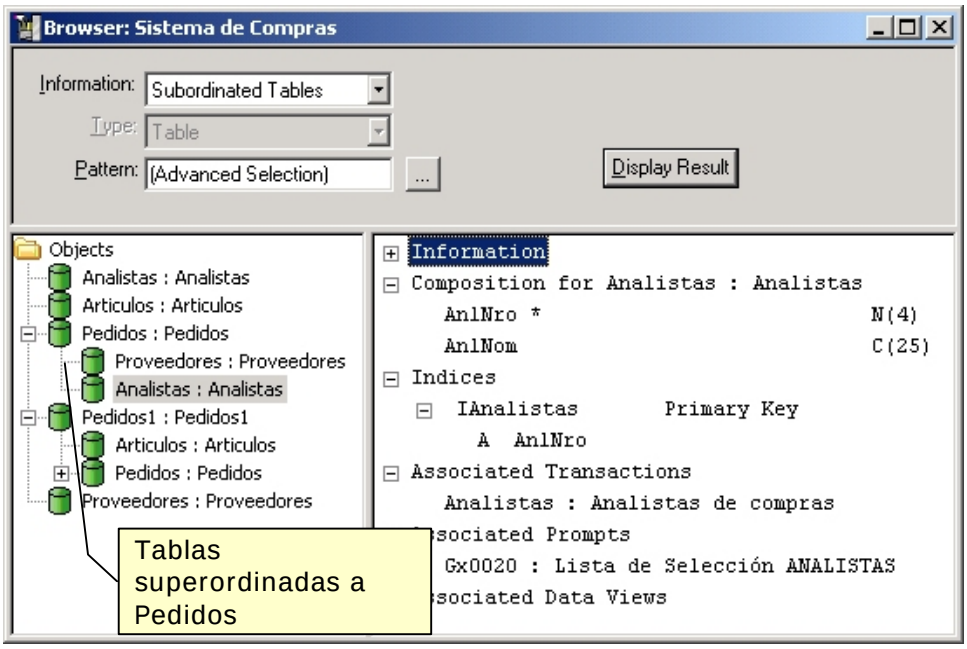


Fig. 19: Browser: Sistema de Compras

Además de ver qué tablas tiene subordinadas y superordinadas, podemos consultar la estructura de una tabla, que índices tiene (composición de los índices), fórmulas, etc.

Existen dos operadores para fórmulas verticales: **SUM** que suma todos los datos y **COUNT** que cuenta cuántos datos hay.

Fórmulas y Redundancia

En base a los criterios de normalización y dado que por definición una fórmula siempre puede ser calculada, no es necesario que la misma esté almacenada y basta con recalcularla cada vez que sea necesario.

Sin embargo el hecho de que la fórmula no esté almacenada puede ocasionar problemas de performance, debido al tiempo que puede demorar el recálculo.

Para evitar este inconveniente se puede definir la fórmula como **redundante**. En este caso la fórmula se almacena en la base de datos y no es necesario recalcularla cada vez que se use. Una vez que la fórmula es definida como redundante, GeneXus se encarga de agregar todas las subrutinas de mantenimiento de redundancia a todas las transacciones que utilicen esa fórmula, de forma tal de manetenerla actualizada en la base de datos.

Tenemos entonces que la definición de fórmulas redundantes implica un balance de performance por un lado y almacenamiento y complejidad de los programas generados en otro, que el analista debe decidir. Si bien en teoría esta decisión puede ser bastante difícil de tomar, en la práctica vemos que la mayor parte de las fórmulas que se definen redundantes son las verticales y pocas veces las horizontales.

La razón de ello es que el cálculo de las fórmulas verticales puede insumir un número indeterminado de lecturas. Por ejemplo, para calcular el *PedTot* se deben leer todas las líneas del pedido, que pueden ser una cantidad bastante grande.

Fórmulas de Fórmulas

Una fórmula se calcula a partir del valor de otros atributos. Dichos atributos pueden estar almacenados o ser otras fórmulas. Por ejemplo, si en la transacción de proveedores definimos:

PrvTotPed = SUM(*PedTot*) Total pedido del proveedor

tenemos que para calcularlo se necesita:

PedTot = SUM(*PedImp*)

que a su vez necesita:

PedImp = *PedCnt* * *PedPrc*

Para fórmulas de fórmulas GeneXus determina cuál es el orden de evaluación correspondiente:

PedImp = *PedCnt* * *PedPrc*
PedTot = SUM(*PedImp*)
PrvTotPed = SUM(*PedTot*)

Las fórmulas aparecen en la estructura de la transacción, como se muestra en la siguiente figura:

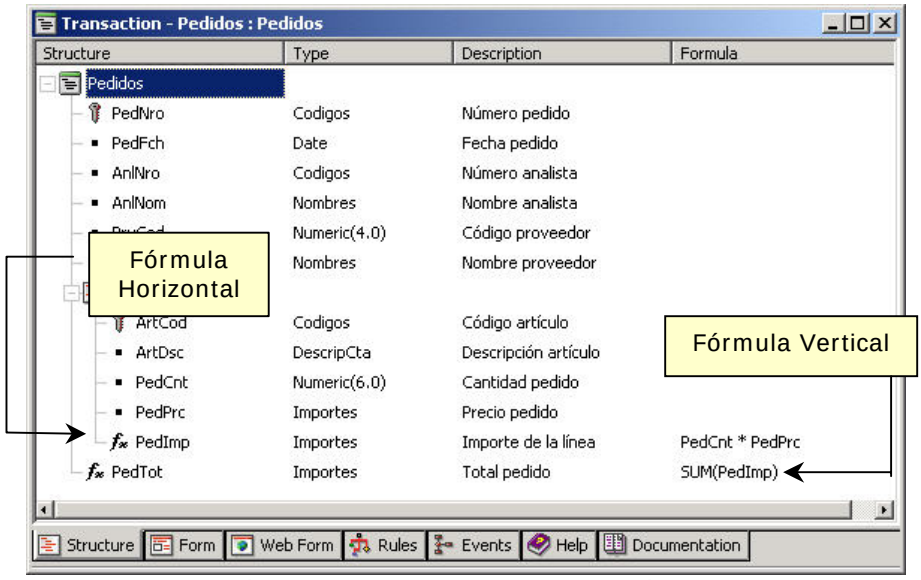


Fig. 20: Fórmulas en la estructura de transacción "Pedidos"

Reglas

Para definir el comportamiento de las transacciones se utizan las reglas.
Usando reglas se definen valores por defecto, controles, etc. Veremos algunas de las reglas en lo que sigue.

Default

Se utiliza para definir los valores por defecto de algunos atributos, por ejemplo:

default(*PedFch*, today());

El funcionamiento de la regla default varía según el tipo de diálogo (full-screen o campo a campo). En el diálogo full-screen se asigna el valor por defecto si el usuario no digitó nada en el campo. En el diálogo campo a campo primero se asigna el valor por defecto y luego se permite modificarlo.

Error

Es la regla para definir los controles que deben cumplir los datos. Por ejemplo, si queremos impedir que quede en un pedido un artículo con cantidad pedida cero, escribimos la regla:

error('La cantidad pedida debe ser mayor que 0') if *PedCnt* <= 0;

Cuando se cumple la condición (*PedCnt* <= 0) se despliega el mensaje (' La cantidad pedida debe ser mayor que 0') en pantalla y no se permite continuar hasta que el usuario ingrese un valor correcto o abandone la transacción.

Una utilización relativamente común es incluir una regla de error para prohibir alguna de las modalidades de la transacción. Por ejemplo:

error('No se permite eliminar Pedidos') if delete;

Con esta regla se prohibe que el usuario entre en el modo eliminación y así se prohibe la eliminación de Pedidos.

Asignación

Dentro de las reglas de la transacción se permiten definir asignaciones a atributos y variables. La asignación a atributos implica una actualización en la tabla base del nivel o en alguna de las tablas que pertenecen a la tabla extendida del nivel. Veamos un ejemplo. En la transacción de Artículos:

Artículos:

*ArtCod**
ArtDsc
ArtCnt
ArtFchUltCmp
ArtPrcUltCmp
ArtUnd
ArtTam
ArtFlgDsp

En el atributo *ArtPrcUltCmp* se desea almacenar cuál fue el precio del último pedido realizado para ese artículo, y en *ArtFchUltCmp* en qué fecha fue realizado. Esto se logra definiendo en la transacción "Pedidos" las siguientes reglas:

ArtPrcUltCmp = *PedPrc* if insert;
ArtFchUltCmp = *PedFch* if insert;

Así cada vez que se ingrese una línea del pedido se actualiza la tabla de artículos con la fecha y el precio correspondiente.
Existe una similitud entre fórmulas y asignaciones, incluso la sintaxis de definición es similar. La diferencia entre ambas es que una fórmula es GLOBAL, es decir, vale para todos los objetos GeneXus que la utilicen, mientras que una asignación es LOCAL, vale solo para la transacción en la cuál esté definida.

Cuando un atributo es fórmula éste no está almacenado (a no ser que se lo defina como redundante) y cuando su valor se asigna mediante la regla de asignación, por ser ésta local, sí lo está.

Add y Subtract

Las asignaciones que vimos en la sección anterior eran relativamente fáciles, pero existen casos más sofisticados. Por ejemplo, en la misma transacción de artículos tenemos el atributo *ArtCnt* en donde se quiere mantener cuál es el stock que tenemos de cada artículo.
Sin duda la transacción "Pedidos" debe modificar ese valor, porque cada pedido nuevo aumenta el stock. También existirá alguna otra transacción (ej: "Facturas") que hace disminuir el stock cada vez que se vende el artículo. Como esta transacción está fuera del alcance del proyecto solo estudiaremos la actualización del stock relacionada con la transacción "Pedidos".

En dicha transacción se debe actualizar *ArtCnt*. Esto se podría hacer con la regla de asignación:

ArtCnt = *ArtCnt* + *PedCnt*;

Pero no debemos olvidar que en la misma transacción se permite crear, modificar o eliminar un pedido, y la asignación anterior solo es correcta si se está creando uno nuevo, ya que si por ejemplo se está eliminando, la asignación correcta es:

ArtCnt = *ArtCnt* - *PedCnt*;

Entonces, para actualizar *ArtCnt* correctamente se necesitaría también considerar los casos de modificación y eliminación, pero para evitar todo esto GeneXus provee la regla add, que lo hace automáticamente:

add(*PedCnt*, *ArtCnt*);

Con esta regla si se está insertando una línea del pedido se suma *PedCnt* a *ArtCnt;* si se está eliminando se resta y si se está modificando se resta el valor anterior de *PedCnt* (que se define como old(*PedCnt*)) y se suma el nuevo valor.
Existe una dualidad entre la fórmula vertical SUM y la regla add; más concretamente se puede decir que un SUM redundante es equivalente a la regla add. Por ejemplo, el *PedTot* se puede definir como:

PedTot = SUM(*PedImp*) y redundante
o
add(*PedImp*, *PedTot*);

Dado que es común que las fórmulas verticales tengan que estar redundantes para tener buena performance se recomienda el uso de la regla add y no la fórmula SUM.
La regla "Subtract" es equivalente pero realiza las operaciones contrarias, es decir, cuando se está insertando resta, en caso de eliminación suma, etc.

Serial

Esta regla es útil cuando se quiere asignar automáticamente valores a algún atributo de la transacción. Por ejemplo, en la transacción "Pedidos", si consideramos la segunda opción vista, en la que un artículo puede repetirse para un mismo pedido, teníamos como identificador del segundo nivel a *PedLinNro* y si queremos que este número sea asignado automáticamente por el sistema se puede usar la regla:

serial(*PedLinNro*, *PedUltLin*, 1);

en donde el primer parámetro define cuál es el atributo que se está numerando, el segundo indica cuál es el atributo que tiene el último valor asignado (aquí se trata del último número de línea asignado) y el tercer parámetro el incremento (en este caso se incrementa de a uno). El atributo *PedUltLin* (última línea asignada) debe ser incluido en la estructura de la transacción en el nivel de *PedNro* (un pedido solo tiene UNA última línea asignada):

*PedNro**
....
PedUltLin ←
(*PedLinNro**
....
)
PedTot

De esta manera para cada nueva línea del pedido el programa asigna automáticamente su número.
En el diálogo campo a campo (donde la modalidad era inferida automáticamente), se debe digitar un valor inexistente en *PedLinNro* (usualmente 0) y el programa asigna el valor correspondiente.
En el diálogo full-screen el valor se asigna cuando el usuario presiona Enter en modo Insert.

Orden de evaluación

La definición de reglas es una forma DECLARATIVA de definir el comportamiento de la transacción. El orden en el cual fueron definidas no necesariamente coincide con el orden en que se encuentran en el programa generado, y por tanto en el que son ejecutadas.
GeneXus se encarga de determinar el orden correcto según las dependencias existentes entre atributos.

Veamos un ejemplo:

Supongamos que definimos en la transacción "Artículos" un nuevo atributo: *ArtCntMax*, del mismo tipo que *ArtCnt*. Este atributo indicará el stock máximo que se desea tener de ese artículo.
Cuando se ingresa un pedido debemos emitir un mensaje si se sobrepasa el stock máximo de un artículo. Para ello definimos las siguientes reglas en la transacción de pedidos:

msg('Se sobrepasa el stock máximo del artículo') if *ArtCnt* > *ArtCntMax*;
add(*PedCnt*, *ArtCnt*);

El orden de evaluación será:

add(*PedCnt*, *ArtCnt*);
msg('Se sobrepasa el stock máximo del artículo') if *ArtCnt* > *ArtCntMax*;

ya que la regla add modifica *ArtCnt* y la regla msg consulta ese atributo.
Esto implica que en la regla msg se debe preguntar por *ArtCnt* mayor que *ArtCntMax* y no por *ArtCnt + PedCnt mayor que ArtCntMax*.

Cada vez que se especifica una transacción se puede pedir la opción **"Detailed Navigation"** que lista cuál es el orden de evaluación de las reglas. En este listado se explicita en qué orden se generarán -usualmente decimos "se dispararán"-, no solo las reglas, sino también las fórmulas y lecturas de tablas.

Call

Call es una regla -y a la vez un comando- con la cual se permite llamar a otro programa. Este último puede ser cualquier otro objeto GeneXus (por ejemplo, de una transacción se puede llamar a un reporte o a otra transacción, etc.), o un programa externo.
En el caso de la regla call es necesario precisar en qué momento se debe disparar el llamado. Por ejemplo, si en la transacción "Pedidos" se quiere llamar a un reporte que imprima el pedido que se acaba de ingresar, se utiliza la regla:

call(RImpRec, *PedNro*) On AfterComplete ;

donde 'RImpRec' es el programa llamado y *PedNro* es el parámetro que se le pasa. Al tener el **evento de disparo** *AfterComplete*, el call se realizará después de haber completado todo el Pedido.
El uso del evento *AfterComplete* no es privativo de la regla call y puede ser utilizado en otras reglas.

Eventos y condiciones de disparo de reglas

Existen eventos del sistema en una transacción, que están relacionados con los distintos momentos que se van sucediendo en el ingreso de los datos. Por ejemplo, existe un evento que ocurre inmediatamente después de que se inserta el registro correspondiente al cabezal en la

tabla asociada. Existe otro que ocurre un instante antes. Otro evento ocurre inmediatamente luego de insertados todos los registros (cabezal y líneas), etc.
A estos eventos se les conoce como **eventos de disparo** ya que se utilizan para condicionar el momento de ejecución –"disparo"- de una regla.

Asimismo, la mayoría de las reglas permiten especificar una **condición de disparo**, que es una condición booleana que puede involucrar atributos, variables, funciones y que puede ser compuesta utilizando los operadores lógicos *or*, *and* y *not*.

Resumiendo: puede condicionarse el disparo de una regla especificando una condición booleana y/o uno o varios eventos de disparo.

Además, puede condicionarse una regla para que se dispare solo en alguno de los niveles de la transacción. La forma de especificar el nivel es nombrando uno o varios atributos de ese nivel, a través de la cláusula "**level**". Si no se menciona dicha cláusula, se asume el primer nivel.

Con lo visto hasta el momento, la sintaxis de las reglas nos queda:

regla [if <**cond**>] [On <**evento**>] [level <**att**>];

Los eventos de disparo tienen que ver con los momentos en los que ocurren las grabaciones/actualizaciones/elminaciones físicas de los registros de la base de datos.

Veremos a continuación los distintos eventos que ocurren.

AfterValidate

Ocurre después de que se confirman los datos del nivel de la transacción en el que se está trabajando, pero antes de realizarse la actualización física correspondiente. Se usa por ejemplo, en algunos casos de numeración automática cuando no se quiere utilizar la regla serial para evitar problemas de control de concurrencia.

AfterInsert | AfterUpdate | AfterDelete

Una regla condicionada a alguno de estos eventos de disparo, se ejecutará inmediatamente después de haber insertado, actualizado o eliminado el registro de la tabla base del nivel. Se usa fundamentalmente para llamar a procesos que realizan actualizaciones.

AfterLevel

Se dispara después de haber entrado todos los datos de un nivel. Se usa muchas veces para controles de totales. Por ejemplo, si cuando se entra el cabezal del pedido se digita un total (llamémosle *PedTotDig*) y se quiere verificar que éste sea igual que el total calculado escribiremos las reglas:

error('El total digitado no cierra con el calculado') if (*PedTotDig* <> *PedTot*) On **AfterLevel Level** *ArtCod*;

en donde el control recién se realizará cuando se terminen de entrar todas las líneas del pedido.

AfterComplete

Se dispara después de haber entrado todos los datos de una transacción. Se usa fundamentalmente para llamar a programas que imprimen los datos, o a otras transacciones. En ambientes con integridad transaccional este evento ocurre luego de que se realiza el commit de la transacción.

Propiedades

En el editor de propiedades de las transacciones se pueden seleccionar configuraciones específicas para definir el comportamiento general del objeto. A continuación vemos la pantalla para la edición de las mismas:

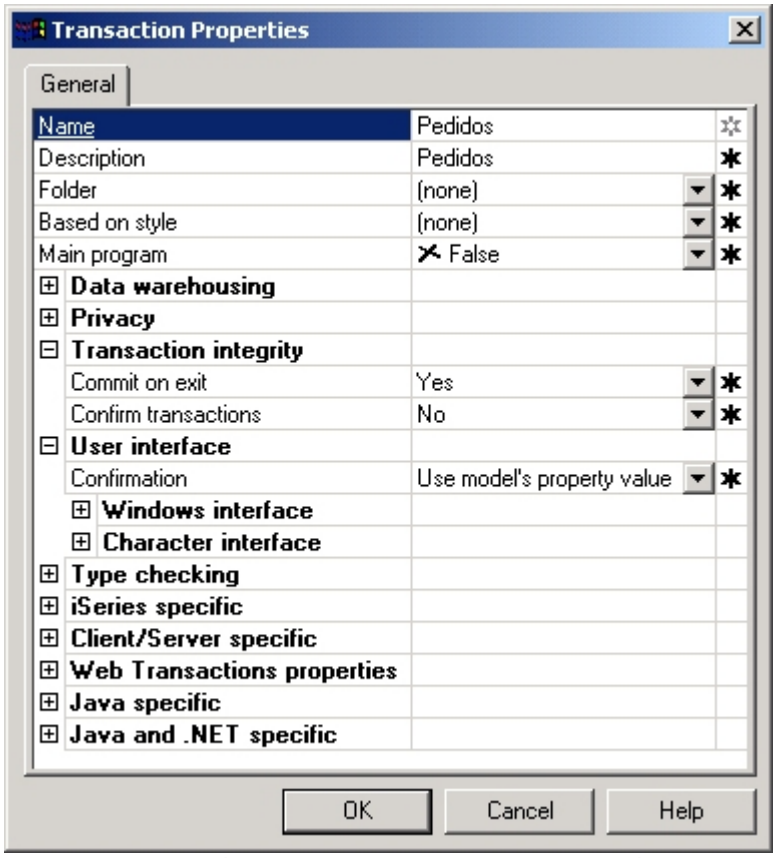


Fig. 21: Diálogo de propiedades de una transacción

Por ejemplo se pueden configurar propiedades que están asociadas con el manejo de la integridad transaccional y con la interface. Podemos indicar si deseamos que la transacción haga un commit al final; asimismo es posible configurar si deseamos que se soliciten confirmaciones cada vez que se actualiza un nivel, etc.

3. Objeto Reporte

Con reportes se definen procesos no interactivos de extracción de datos. Los reportes son listados cuya salida puede: emitirse por impresora, visualizarse por pantalla, o enviarse a un archivo.

Es un proceso no interactivo, porque primero se extraen los datos y luego se muestran, no habiendo interacción con el usuario durante el proceso de extracción, y es solo de extracción ya que no se pueden actualizar los datos leídos.

Elementos

Para cada reporte se definen, entre otros, los siguientes elementos:

Layout
Así como las transacciones tienen una pantalla (form), los reportes tienen un "layout" de la salida. En esta sección se define la presentación del reporte: los datos que se quieren listar y el formato de la salida.

Source
Aquí se escribe el código correspondiente a la lógica del reporte. También pueden definirse al final del código subrutinas que podrán ser invocadas desde el propio código mediante el comando adecuado.

Reglas – Propiedades
Definen aspectos generales del reporte, como su nombre, descripción, tipo de salida (impresora, archivo, pantalla), parámetros que recibe el objeto, etc.

Condiciones
Condiciones que deben cumplir los datos para ser recuperados (filtros).

Ayuda
Texto para la ayuda a los usuarios en el uso del reporte.

Documentación
Texto técnico para ser utilizado en la documentación del sistema.

Solapas de acceso a los elementos

Como para todo objeto GeneXus, se puede acceder a varios de los elementos que lo componen utilizando las solapas que aparecen bajo la ventana de edición del objeto:

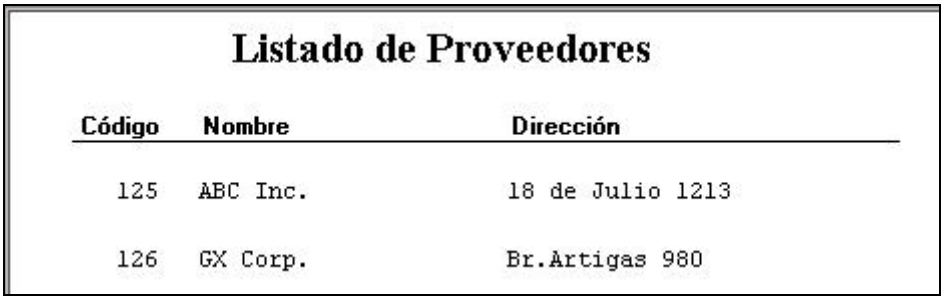
Fig. 22: Solapas que figuran en la parte inferior de la ventana de edición de un reporte

Layout

En esta sección se definen los datos que se quieren listar y el formato de la salida.

El Layout es una sucesión de bloques conocidos como **print blocks** (bloques de impresión). Estos bloques son áreas que serán desplegadas en el listado resultante y que contendrán controles que indicarán qué es lo que se quiere imprimir, dando forma a la salida.

Por ejemplo, supongamos que queremos implementar un reporte para imprimir el código, nombre y dirección de todos nuestros proveedores y queremos que el listado luzca como sigue:



Listado de Proveedores

Código	Nombre	Dirección
125	ABC Inc.	18 de Julio 1213
126	GX Corp.	Br.Artigas 980

Fig. 23: Listado de Proveedores en ejecución

Para implementar este listado creamos un objeto reporte, y definimos en su sección Layout todos los print blocks necesarios. Para este ejemplo, podríamos dividir la salida en 2 áreas, una con datos fijos y otra con datos variables, cada una implementada con un print block. El Layout correspondiente se muestra a continuación:

Fig. 24: Layout del "Listado de Proveedores"

El orden de los print blocks dentro del Layout es irrelevante: en esta sección simplemente se **declaran**. El orden en el que serán desplegados en la salida queda determinado en la sección Source, que es la que contiene la lógica del reporte. Desde allí serán invocados para ser impresos, mediante un comando específico para tal fin (el comando print).
Por esta razón cada print block deberá tener un nombre, de forma de poder ser referenciado luego desde el Source. Este nombre es el que aparece a la izquierda de cada uno de ellos en el Layout de la figura 24.

El print block es un nuevo tipo de control, solo válido en reportes y procedimientos, que indica que debe mostrarse en la salida la información especificada dentro de él. Esta información es la representada por otros controles, de los ya estudiados (atributos, textos, recuadros, líneas, etc.).

Para listar el código, nombre y dirección de todos los proveedores, el print block de nombre "Proveedor" deberá ser invocado dentro de una estructura repetitiva en el Source, de forma tal que se recorran todos los proveedores y se impriman cada vez los datos del proveedor en el que

se está posicionado en ese momento. Esta estructura repetitiva es el comando for each que introduciremos luego.

Como cualquier otro control, el print block tiene asociadas ciertas propiedades, que pueden ser configuradas desde un diálogo que se abre al hacer doble-clic sobre el control. Algunas de estas propiedades son el tipo de papel, el tipo de letra, etc. El diálogo es el siguiente:

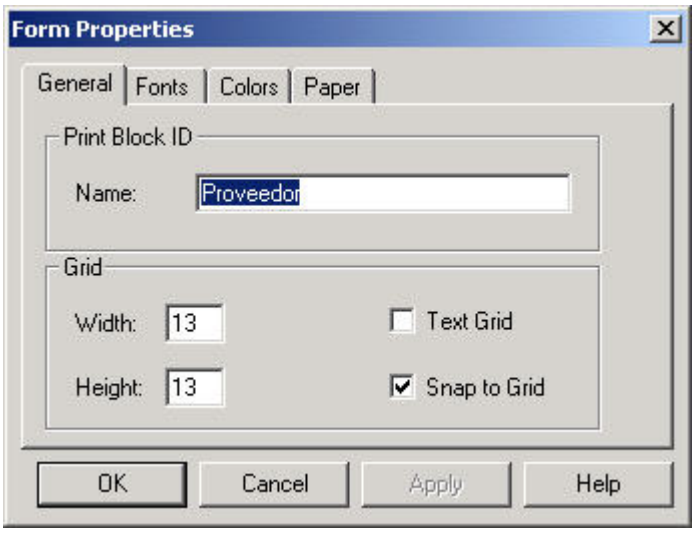


Fig. 25: Propiedades del control Print Block

Para el diseño de los print blocks tenemos disponible la misma paleta de herramientas "Controls" de las transacciones, teniendo habilitadas las opciones que aplican a este tipo de objeto:

Fig. 26: Paleta de herramienta "Controls" en reportes

Esta paleta permite insertar atributos, textos, líneas, recuadros, bitmaps y nuevos print blocks. Dentro del reporte, los controles son insertados y configurados de la misma forma que en el editor del form de las transacciones.

Source

En esta sección se programa la lógica del reporte. El lenguaje que se utiliza es muy simple, y consta de algunos comandos.

El estilo de programación es procedural (imperativo), por lo que el Source será una sucesión de comandos para los que importa el orden, dado que éste será el orden de ejecución.
Existen, como en todo lenguaje imperativo, comandos de control para la ejecución condicional (if, do case), la repetitiva (do while, for), para invocar a otro objeto (call), para cortar las iteraciones dentro de un bucle (exit) o abandonar el programa (return), así como también comandos específicos de este lenguaje: para imprimir un print block del Layout (print), para acceder a la base de datos (for each), para invocar a una subrutina (do), etc.

Al final del source pueden definirse subrutinas (mediante el comando sub) que serán locales al objeto, y que podrán ser invocadas más arriba, dentro del mismo source, en el código de ejecución, mediante el comando do.

El siguiente es el Source que implementa la lógica del reporte del ejemplo:

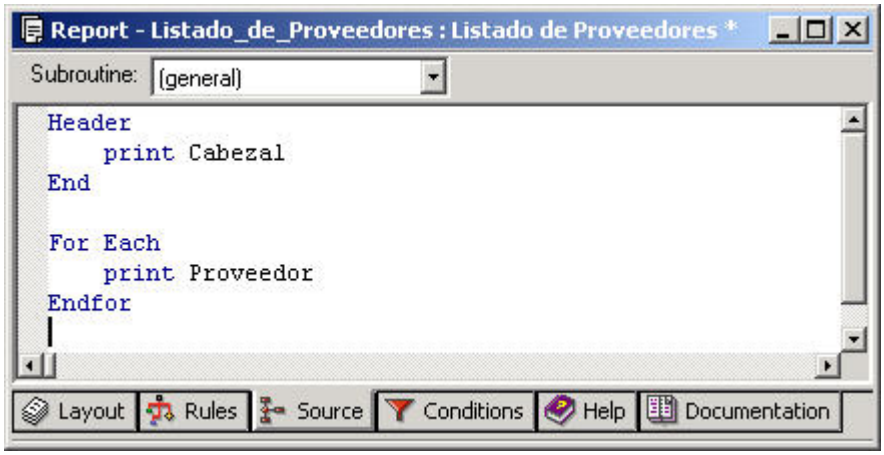


Fig. 27: Source del "Listado de Proveedores"

donde tenemos los comandos Header, Print y For each.
Como resulta evidente, con el comando print mandamos a imprimir un print block en la salida. Todos los print blocks que se manden a imprimir entre los comandos **Header** y **end** son las líneas que se imprimen en el cabezal de la página. También existe el comando **Footer** que permite imprimir un pie de página en cada hoja.

El comando **For each** indica que se consultarán cada uno de los registros de cierta tabla, mostrando la información que se indique en el/los print block/s que se mande/n a imprimir dentro del comando.
En este caso, GeneXus infiere que debe navegar la tabla PROVEEDORES, mostrando el código, nombre y dirección de cada proveedor.

Comando FOR EACH

Este comando es fudamental, ya que es el único comando que permite recuperar información de la base de datos. Usando el for each se define la información que se va a leer. La forma de hacerlo se basa en nombrar los atributos a utilizar y NUNCA se especifica de qué tablas se deben obtener, ni qué índices se deben utilizar.

Con este comando se define QUE atributos se necesitan y en qué ORDEN se van a recuperar, y GeneXus se encarga de encontrar COMO hacerlo. Obviamente esto no siempre es posible, y en esos casos GeneXus da una serie de mensajes de error indicando por qué no se pueden relacionar los atributos involucrados.

La razón por la cuál no se hace referencia al modelo físico de datos es porque de esta forma la especificación del reporte es del más alto nivel posible, de tal manera que ante cambios en la estructura de la base de datos la especificación del mismo se puede mantener válida la mayor parte de las veces.

El acceso a la base de datos queda especificado por los atributos que son utilizados dentro de un grupo FOR EACH - ENDFOR. Para ese conjunto de atributos GeneXus buscará la tabla extendida que los contenga (el concepto de tabla extendida es muy importante y sugerimos repasar su definición en el Anexo sobre Modelos de Datos Relacionales).

Por ejemplo, en el reporte anterior, dentro del FOR EACH - ENDFOR se utilizan los atributos *PrvCod, PrvNom y PrvDir*. GeneXus "sabe" que estos atributos se encuentran en la tabla

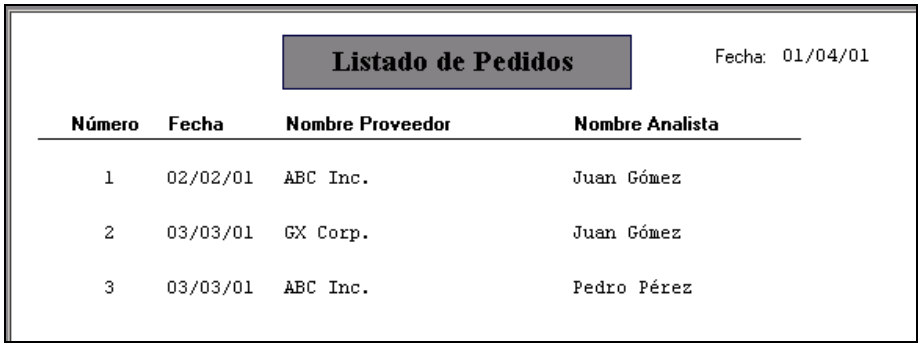
PROVEEDORES y por lo tanto el comando:

```
For each
    print Proveedores
endfor
```

es interpretado por GeneXus como:

For each record in table **PROVEEDORES**
    Imprimir *PrvCod*, *PrvNom* y *PrvDir*
  endfor

Para clarificar el concepto hagamos un reporte que involucre datos de varias tablas. Por ejemplo, listemos el cabezal de todos los pedidos:



Listado de Pedidos

Fecha: 01/04/01

Número	Fecha	Nombre Proveedor	Nombre Analista
1	02/02/01	ABC Inc.	Juan Gómez
2	03/03/01	GX Corp.	Juan Gómez
3	03/03/01	ABC Inc.	Pedro Pérez

Fig. 28: "Listado de Pedidos" en ejecución

El Layout para este listado es:

Fig. 29: Layout "Listado de Pedidos"

El diagrama de Bachman de las tablas del modelo es:

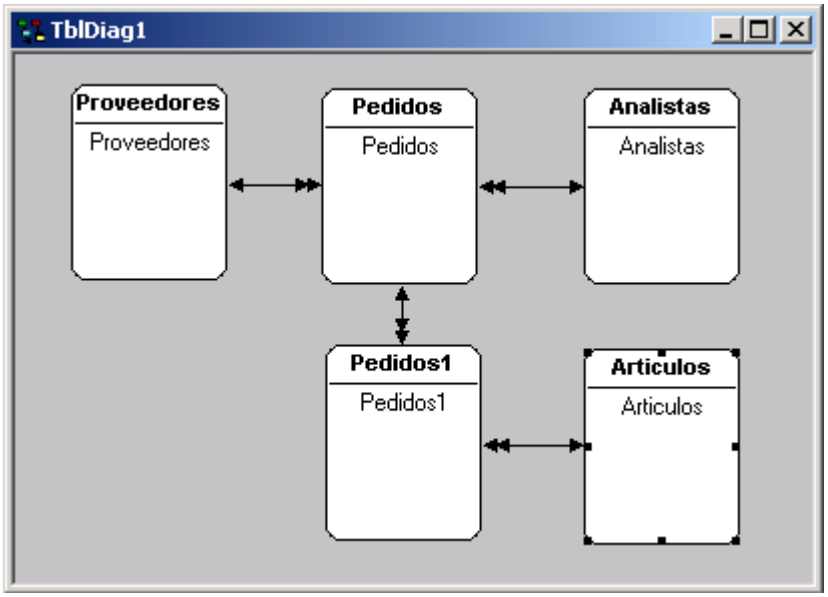


Fig. 30: Diagrama de Bachman del modelo

En el listado anterior se requieren datos de las tablas PROVEEDORES (*PrvNom*), ANALISTAS (*AnlNom*) y PEDIDOS (*PedNro* y *PedFch*). La tabla extendida de PEDIDOS contiene a todos estos atributos y por lo tanto el comando FOR EACH se interpretará como:

For each record in tabla **PEDIDOS**
  Find the corresponding *PrvNom* in table **PROVEEDORES**
  Find the corresponding *AnlNom* in table **ANALISTAS**
    Imprimir *PedNro*, *PedFch*, *PrvNom* y *AnlNom*
  endfor

Cuando se especifica un reporte, GeneXus brinda un listado, que llamamos **Listado de Navegación**, donde se indica cuáles son las tablas que se están accediendo:

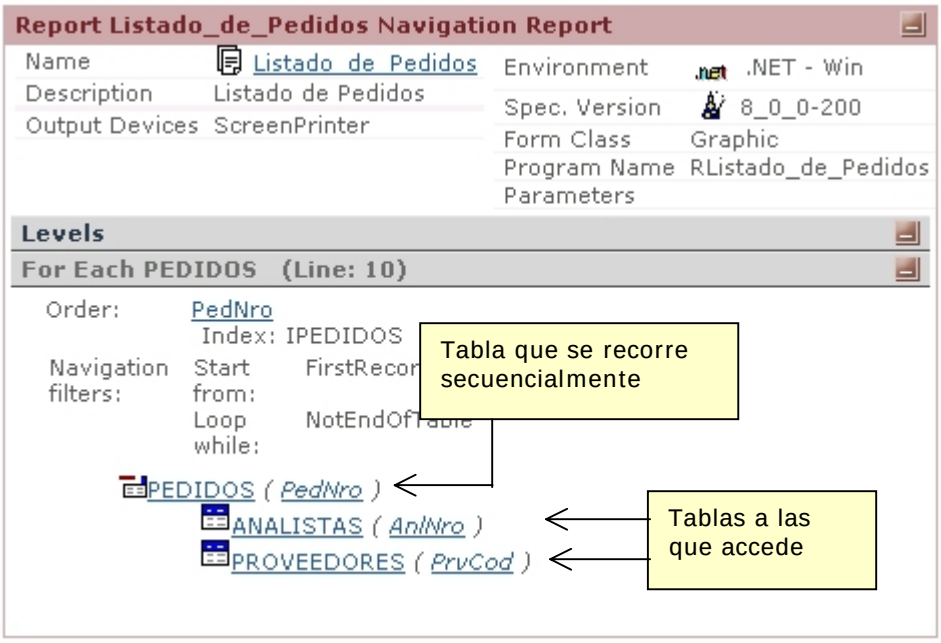


Fig. 31: Listado de Navegación del reporte "Listado de Pedidos"

Donde la primer tabla representada indica cuál es la **tabla base** del for each y las tablas que figuran debajo de ésta, e indentadas, indican las tablas que deben accederse a partir de cada registro de la tabla base en el que se esté posicionado en cada momento. Una interpretación muy práctica de este diagrama es: para la tabla del primer nivel de indentación se leen muchos registros y para cada tabla del siguiente nivel se lee un solo registro: el asociado al del nivel anterior.
En este diagramita de tablas que aparece en el listado, la relación entre la tabla de un nivel de indentación, con cualquier tabla "anidada" es N-1.

Orden

En los reportes anteriores no se tuvo en cuenta en qué orden se deben listar los datos. Cuando no se indica el orden, GeneXus utiliza como orden el de la clave primaria de la tabla base del For Each (existen algunas excepciones a esta regla).

En el listado de navegación, para cada for each se indica cuál es su tabla base, e inmediatamente el orden elegido para recuperar la información. Como podemos ver en la fig.31, GeneXus indica que va a ordenar por *PedNro*, es decir, por el atributo que es clave primaria de la tabla base del for each.

Para los casos en donde se quiere definir el orden de los datos de forma explícita, se utiliza la cláusula ORDER en el comando For each. Por ejemplo, para listar los pedidos ordenados por código de proveedor, *PrvCod*:

```
For each ORDER PrvCod
    print Pedidos
endfor
```

En este caso el índice a utilizar será el de nombre IPEDIDOS2 (*PrvCod*) de la tabla PEDIDOS, que fue definido automáticamente por GeneXus, ya que *PrvCod* es una clave foránea. La navegación para este reporte es la siguiente:

Fig. 32: Fragmento del listado de navegación reporte "Listado de Pedidos"

Si queremos que el listado de pedidos salga ordenado por fecha haremos:

```
For each ORDER PedFch
    print Pedidos
endfor
```

En este caso no hay ningún índice definido en PEDIDOS para satisfacer el orden, dado que *PedFch* no es clave en ningún lado. Como no existe un índice definido, GeneXus indicará en el listado de navegación mediante una advertencia ("warning") que **no existe un índice para satisfacer el orden**, lo que podría ocasionar problemas de performance, dependiendo de la plataforma de implementación, de la cantidad de registros que deben ser leídos de la tabla, etc.

En algunas plataformas, como VFP o iSeries (AS/400), si no existe índice para satisfacer el orden especificado se crea uno temporal cada vez que se ejecute el reporte y se elimina al finalizar la ejecución.
Este tipo de índice es el que llamamos **índice temporal**.

En otras plataformas, como VB con Access o en ambientes cliente/servidor, si no existe índice para satisfacer el orden, el For Each se traduce en una consulta sql ("select") que es resuelta por el motor del DBMS de la plataforma.

El listado de navegación mostrará en este caso:

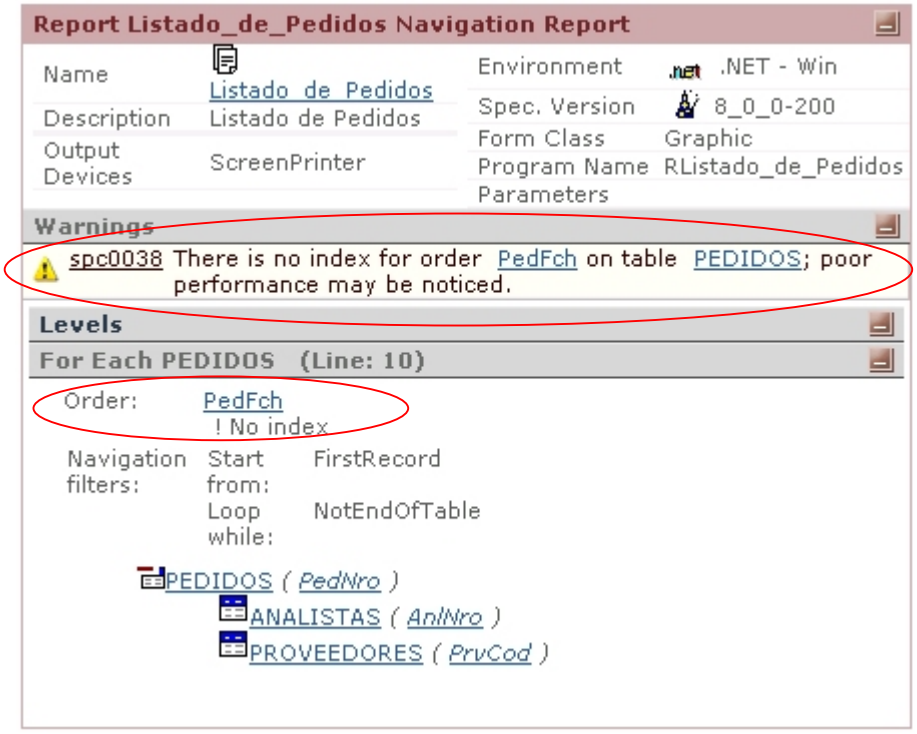


Fig. 33: Listado de navegación del reporte "Listado de Pedidos" ordenado por fecha

Evidentemente este proceso es más lento si lo comparamos con el listado anterior en donde había un índice definido. Dada esta situación el analista debe tomar una decisión:

- Crear un **índice de usuario**. En este caso el reporte será bastante más eficiente, pero se debe mantener un índice más (lo que implica mayor almacenamiento y mayor procesamiento en las transacciones para mantener actualizado el índice).
- No crear el índice de usuario, y dejar que la navegación sea resuelta por el dbms o mediante la creación de un índice temporal.

No es posible recomendar, a priori, cuál de las dos soluciones es la mejor, por lo tanto se debe estudiar, caso por caso, la solución particular, siendo fundamental la frecuencia con que se ejecutará el reporte. De cualquier manera si al comienzo no se definió el índice y posteriormente se decide definirlo, alcanza con regenerar el reporte (sin modificar nada del mismo) y éste pasará a utilizarlo.

Los criterios de ordenamiento utilizados por GeneXus son:

Cada grupo FOR EACH puede estar ordenado por CUALQUIER conjunto de atributos pertenecientes a la **tabla base** del grupo, pero no se puede ordenar por atributos que NO se

encuentren en la misma. Por ejemplo el listado anterior no se puede ordenar por *PrvNom* porque la tabla base del grupo es PEDIDOS en la que no está almacenado el atributo *PrvNom*. Sin embargo, si la plataforma de implementación es cliente/servidor, el criterio es más amplio y podrá también ordenarse por atributos de la tabla extendida (por lo tanto en este caso sí se podrá ordenar por *PrvNom*).

Condiciones en el FOR EACH

Para restringir los datos que se quieren listar se utiliza la cláusula WHERE. Por ejemplo, para listar sólo los pedidos de hoy:

```
For each
WHERE PedFch = Today()
    print Pedidos
endfor
```

Se pueden definir varios WHERE dentro del FOR EACH:

```
For each
WHERE PedFch = Today()
WHERE PedNro > 100
   print Pedidos
endfor
```

Lo que significa que se deben cumplir AMBAS condiciones, es decir, se interpreta como que ambas cláusulas están relacionadas por un AND. Si se quiere utilizar un OR (es decir, se desea que se cumpla alguna de las dos condiciones) se debe utilizar un solo WHERE:

```
For each
WHERE PedFch = Today() OR PedNro > 100
   print Pedidos
endfor
```

GeneXus posee una serie de mecanismos para optimizar el reporte en función del orden del FOR EACH y de las condiciones del mismo.

Cuando la condición ha sido optimizada, en el Listado de Navegación aparece como **'Navigation Filter'** y cuando no, como **'Constraint'**.

Por ejemplo, la parte del listado de navegación que corresponde al siguiente for each de un reporte:

```
For each order PedFch
WHERE PedFch = &today
   print Pedidos
endfor
```

siendo *&today* una variable del sistema que contiene la fecha de hoy, es:

Fig. 34: Fragmento listado de navegación de for each optimizado

Observar que los "Navigation filters" especifican el rango de la tabla que se va a recorrer para luego seleccionar los registros que cumplan con los filtros que aparecen como "Constraints". En el listado anterior no hay "Constraints", lo que significa que se van a imprimir todos los registros del rango especificado. En este caso el rango a recorrer no es toda la tabla. Como se está ordenando por *PedFch* y filtrando por *PedFch*, no tiene que recorrerse toda la tabla. Decimos que se eligió un orden compatible con los filtros, y por ello GeneXus puede optimizar el acceso a las tablas.

El mismo reporte, pero esta vez sin especificar cláusula ORDER en el for each:

```
For each
WHERE PedFch = &today
   print Pedidos
endfor
```

muestra en el listado de navegación:

Fig. 35: Fragmento listado de navegación de for each NO optimizado

Aquí, como no especificamos orden, GeneXus eligió la clave primaria de la tabla base. Este listado nos está diciendo que deberá recorrerse toda la tabla PEDIDOS, y para cada registro, si cumple las "Constraints" lo imprimirá.
Este reporte no optimiza el acceso a las tablas, pues no hemos ordenado de forma compatible con los filtros.

Cláusula WHEN NONE de un For Each

En muchos casos es necesario ejecutar determinado código cuando en un For Each no se encuentra ningún registro. Para esto se usa la cláusula WHEN NONE.

Ejemplo:

```
For each
where PedFch = Today()
    print Pedidos
WHEN NONE
   Msg('No se realizó ningún pedido en el día de hoy')
endfor
```

Es importante aclarar que si se incluyen For Eachs dentro del When None no se infieren Joins ni filtros de ningún tipo con respecto al For Each que contiene el When None.

Determinación de la tabla extendida del grupo FOR EACH

Resulta importante saber cómo determina GeneXus la tabla extendida que se debe utilizar. El criterio básico es:

Dado el conjunto de atributos que se encuentran dentro de un grupo FOR EACH - ENDFOR, GeneXus determina la MINIMA tabla extendida que los contiene.

Supongamos que queremos listar el nombre del analista y proveedor de cada pedido.
Para ello escribiremos en el Source:

```
For each
   print NombreProveedorAnalista
endfor
```

donde "NombreProveedorAnalista" será el nombre de un print block del Layout que contendrá los atributos *PrvNom* y *AnlNom*. Observemos que la única diferencia con el listado de pedidos realizado antes es que aquí no nos interesa listar ni el nro. de pedido ni la fecha del mismo, sino solo analista y proveedor.
Ahora los atributos que participan en la determinación de la tabla base del For Each son *PrvNom* y *AnlNom*. La mínima tabla extendida que contiene a los atributos del For Each sigue siendo PEDIDOS.

Sin embargo, podría darse el caso de que existiera más de una tabla extendida mínima que contuviera a los atributos de un grupo FOR EACH - ENDFOR. Ante esta ambigüedad GeneXus escoge la PRIMER tabla extendida que cumpla con las condiciones anteriores.

Para ver un ejemplo de ello, supongamos que agregamos a nuestra aplicación una transacción "Contactos" para representar los contactos efectuados por un analista a un proveedor pidiendo cotizaciones. Nos interesa mantener un histórico de tales contactos. Para ello, definimos la siguiente estructura:

*CtoNro** Nro de contacto
PrvCod
PrvNom
AnlCod
AnlNom
CtoFch Fecha del contacto

Si hacemos un diagrama de Bachman que represente las tablas PEDIDOS, ANALISTAS, PROVEEDORES y CONTACTOS, tenemos:

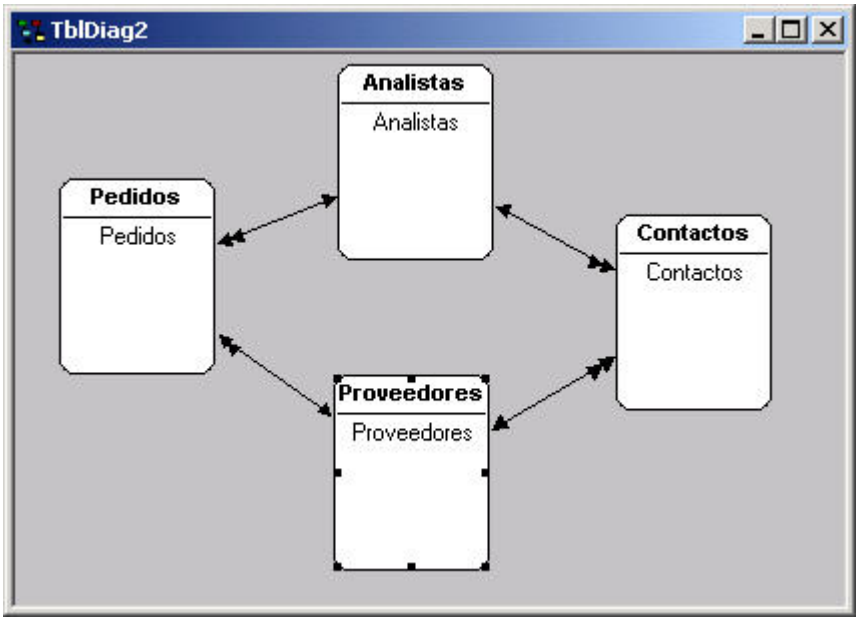


Fig. 36: Diagrama de Bachman

Con la nueva tabla CONTACTOS, aparece una ambigüedad en la resolución de la tabla base del for each mencionado antes.

Ahora existen dos tablas extendidas mínimas que contienen a los atributos *PrvNom* y *AnlNom* del For Each: la que tiene como tabla base PEDIDOS y la que tiene como tabla base CONTACTOS.

Pero nosotros queremos listar los nombres de proveedor y analista de cada pedido, y no de cada contacto. Para resolver la ambigüedad que aparece, se utiliza la cláusula DEFINED BY dentro del comando FOR EACH. Así, en el ejemplo agregaremos:

```
For each
DEFINED BY PedFch
   print NombreProveedorAnalista
Endfor
```

En el DEFINED BY se debe hacer referencia a por lo menos un atributo de la tabla base deseada.
Aún en los casos en los que no se da el problema anterior, se recomienda el uso del DEFINED BY para reportes más o menos complejos porque mejora bastante el tiempo de especificación del reporte.

Reportes con varios FOR EACHs

For Eachs anidados

Hasta ahora hemos definido reportes en donde existía un solo FOR EACH, pero es posible utilizar más de un FOR EACH para realizar reportes más sofisticados. Supongamos que queremos listar para cada proveedor cuáles son sus pedidos. Por ejemplo:

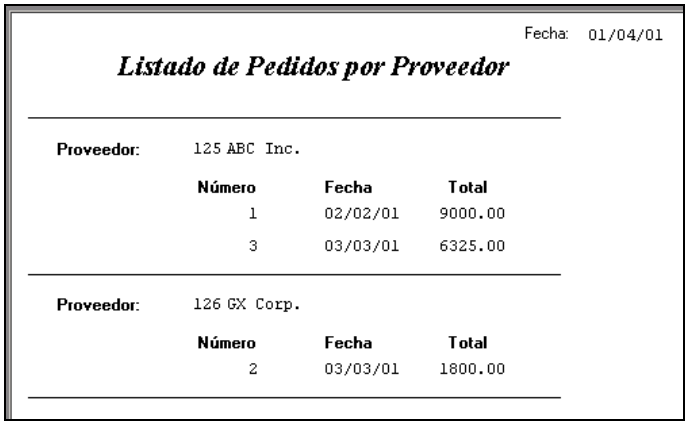


Fig. 37: Listado de Pedidos por Proveedor

Para ello creamos un reporte con el siguiente Layout:

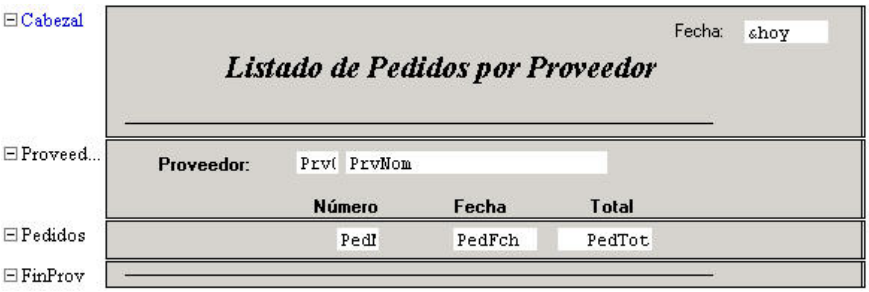


Fig. 38: Layout de "Listado de Pedidos por Proveedor"

Y en el Source escribiremos:

```
Header
   print Cabezal
end
For each
   print Proveedor
   For each
      print Pedidos
   endFor
   print FinProv
endFor
```

En este reporte tenemos dos FOR EACHs anidados. Si analizamos el primer FOR EACH vemos que utiliza sólo los atributos *PrvCod* y *PrvNom* y por lo tanto su tabla extendida será la de la tabla PROVEEDORES. El segundo FOR EACH utiliza los atributos *PedNro, PedFch* y *PedTot* por lo que su tabla extendida será la de PEDIDOS. Pero el segundo FOR EACH se encuentra anidado respecto al primero y a su vez la tabla PEDIDOS está subordinada a PROVEEDORES (por *PrvCod*) por lo tanto GeneXus interpretará el reporte como:

```
For each record in table PROVEEDORES
  ....
        For each record in table PEDIDOS
        with the same PrvCod
        ....
        endfor
  ....
endfor
```

Y así se listarán, para cada registro en la tabla de proveedores, todos sus pedidos (de la tabla de pedidos).
Más formalmente, cuando hay dos FOR EACHs anidados, GeneXus busca cuál es la relación de subordinación existente entre la **tabla base del segundo FOR EACH** y la **tabla extendida del primer FOR EACH** (en el caso anterior la relación era que la tabla PEDIDOS estaba subordinada a la tabla PROVEEDORES por *PrvCod*) y establece de esta manera la condición que deben cumplir los registros del segundo FOR EACH (en el ejemplo: todos los registros de la tabla PEDIDOS con el mismo *PrvCod* leído en el primer FOR EACH).

Hay otro caso más en el que GeneXus encuentra relación entre los datos que debe recuperar. Por ejemplo, supongamos que agregamos a la aplicación una transacción "Paises", y luego representamos que cada proveedor pertenece a un país. El siguiente diagrama de Bachman representa las relaciones entre las tablas relevantes:

Si nos interesa hacer un listado de los pedidos por país, escribiremos en el Source:

```
For each
   print Pais
   For each
      print Pedido
   endfor
endfor
```

donde "Pais" es el nombre de un print block que contiene atributos de PAISES (por ejemplo, código y nombre del país) y "Pedido" es el nombre de un print block que contiene atributos de PEDIDOS (por ejemplo *PedNro*, *PedFch* y *PedTot*)

Aquí las tablas base serán PAISES y PEDIDOS respectivamente y GeneXus encuentra que hay una especie de subordinación indirecta entre ellas, de manera tal que para cada registro de la tabla base del primer for each, recuperará de la tabla base del segundo for each solo aquellos pedidos que correspondan a un proveedor de ese país.
Formalmente, este caso se da cuando la tabla base del primer for each está contenida en la tabla extendida del segundo.

Pero puede darse el caso de que GeneXus no encuentre relación entre ambos FOR EACHs, por ejemplo, en el siguiente caso:

```
For each
   print Proveedores
   For each
      print Analistas
   endfor
endfor
```

donde el Layout contiene los siguientes print blocks:

Fig. 39: Layout Listado de Proveedores y Analistas

La tabla base del primer FOR EACH es PROVEEDORES (porque los atributos son *PrvCod* y *PrvNom*) y la del segundo FOR EACH es ANALISTAS (porque los atributos son *AnlNro* y *AnlNom*) y no existe relación entre la tabla extendida de PROVEEDORES y la tabla base del for each anidado: ANALISTAS. Tampoco se da el caso de que la tabla extendida del FOR EACH anidado contenga a la tabla base del principal.

Cuando se da este caso, en el que GeneXus no encuentra una relación de subordinación, hace el producto cartesiano entre los dos FOR EACHs, es decir, para cada registro de la tabla PROVEEDORES se imprimirán todos los registros de la tabla ANALISTAS.

Cortes de Control

Un caso particular de FOR EACHs anidados son los llamados cortes de control.
Supongamos que queremos listar los pedidos ordenados por fecha:

Fig. 40: Listado de Pedidos por Fecha

El Layout nos queda:

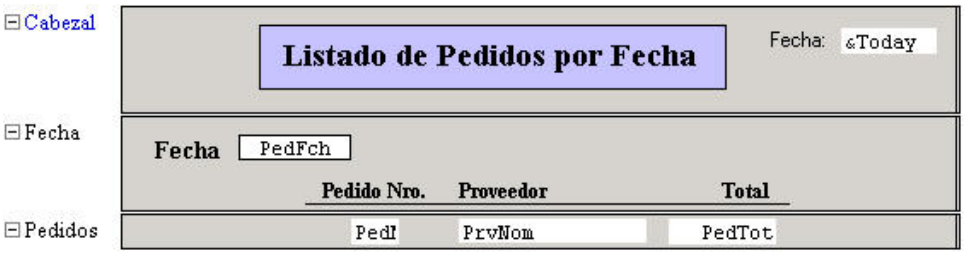


Fig. 41: Layout del "Listado de Pedidos por Fecha"

Y el Source:

```
header
   print Cabezal
end
for each order PedFch
   print Fecha
   for each
      print Pedidos
   endfor
endfor
```

La tabla base del primer FOR EACH es PEDIDOS (pues el único atributo que aparece es *PedFch*) y la del segundo FOR EACH también (pues contiene *PedNro*, *PrvNom* y *PedTot*). Este es un caso de **corte de control** sobre la tabla PEDIDOS y GeneXus lo indica en el Listado de Navegación con la palabra BREAK -en lugar de FOR EACH- para el For Each anidado. Los cortes de control pueden ser de varios niveles. Por ejemplo, si quisiéramos listar los pedidos por fecha y dentro de cada fecha por proveedor:

```
For each order PedFch
   ....
   For each order PrvCod
      ....
      For each
         ....
      endfor

   endfor

 endfor
```

Cuando se definen cortes de control es MUY IMPORTANTE indicar el ORDEN en los FOR EACH ya que es la forma de indicar a GeneXus por qué atributos se quiere realizar el corte de control.

For Eachs paralelos

También se pueden definir grupos FOR EACH paralelos, por ejemplo, si se desean listar los proveedores y luego los analistas en el mismo listado, implementamos el siguiente Source:

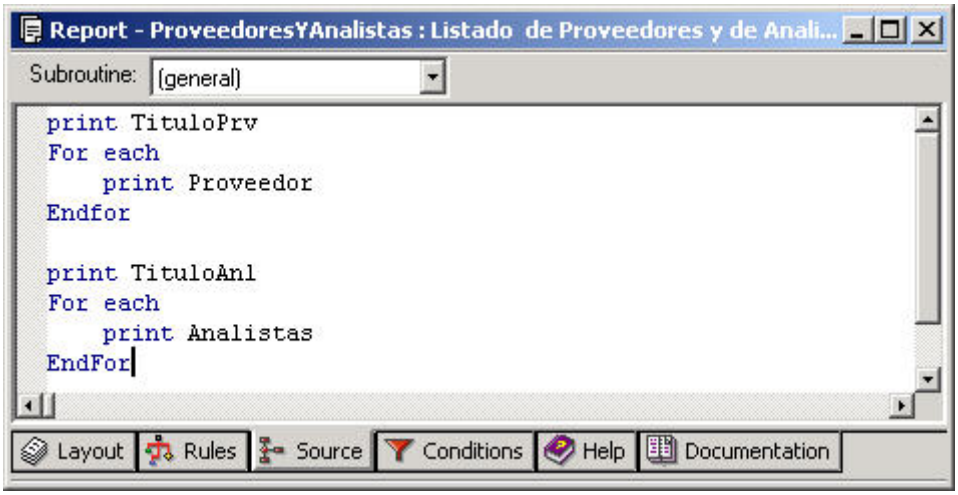


Fig. 42: Source del "Listado de Proveedores y de Analistas"

Siendo el Layout:

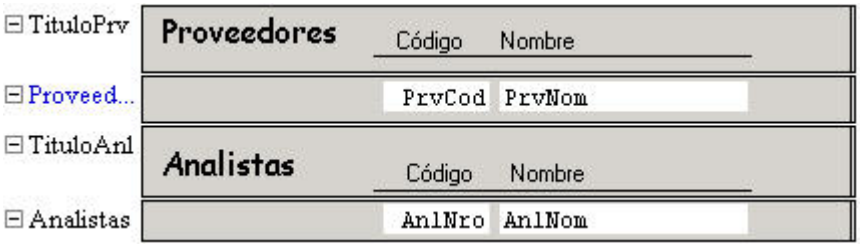


Fig. 43: Layout del "Listado de Proveedores y de Analistas"

Los listados con FOR EACHs paralelos son muy prácticos cuando se quiere hacer listados del tipo "Listar toda la Información que se tenga sobre un Proveedor", en donde el Source será del tipo:

```
For each
    ...   //Información del Proveedor
    For each
        ....   // Información sobre Pedidos
    Endfor
    For each
        ....   // Información sobre Precios
    Endfor
Endfor
```

Otros comandos

En el Source se pueden utilizar otros comandos además de los ya vistos. Nos limitaremos aquí a ver los más significativos.

Definición de Variables

Antes de describir los comandos en sí es bueno detenernos en el concepto de variable. Ya vimos que los datos se almacenan en la base de datos en atributos, pero muchas veces se necesita utilizar variables, que se diferencian de los atributos en que son locales al objeto en el que se definen y no están almacenadas en la base de datos. Una variable es reconocida por GeneXus por ser un nombre que comienza con &, por ejemplo *&Today* o *&Aux*.
Existen algunas variables predefinidas, es decir, que ya tienen un significado propio. Por ejemplo, ya mencionamos a *&Today*, que es una variable con la fecha de hoy. Otro ejemplo es la varible *&Page* que indica cuál es la página que se está imprimiendo.
Además de estas variables del sistema, el analista puede definir para cada objeto sus propias variables. Estas variables serán locales al objeto en el que se definan.
Para cada variable definida por el analista, se deben especificar sus distintas características, como nombre, descripción, tipo de datos, largo, decimales, etc.
Para ello se abre el siguiente diálogo:

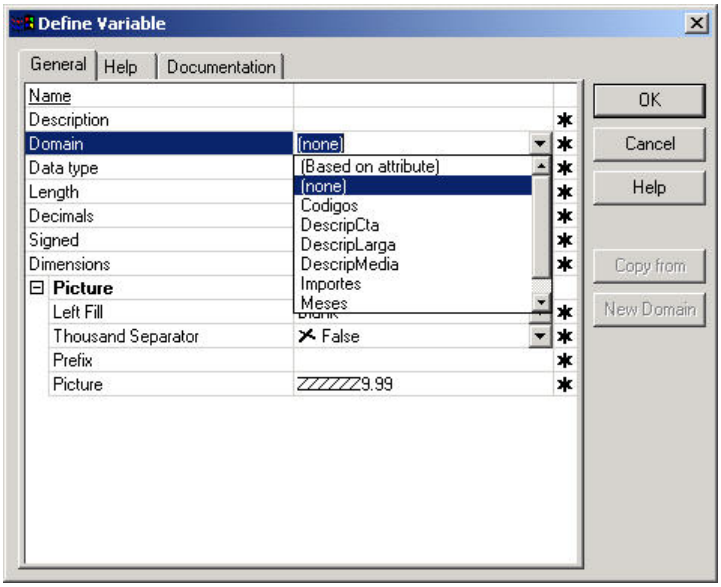


Fig. 44: Diálogo de definición de variable

Las variables se pueden definir también en función de otros atributos, usando la opción "Based on attribute" que aparece en el combo asociado al dominio (Domain). Se recomienda utilizar esta opción siempre que sea posible. De esta manera, cada vez que cambie el tipo de datos del atributo en el cuál una variable está basada, automáticamente el cambio será propagado también a la variable.

Asignación

Existen varios comandos (=, +=, -=, *=, /=) que permiten asignar valores a variables (en los procedimientos también se permite la asignación de valores a atributos, como veremos oportunamente). Por ejemplo si queremos mostrar totales en un listado:

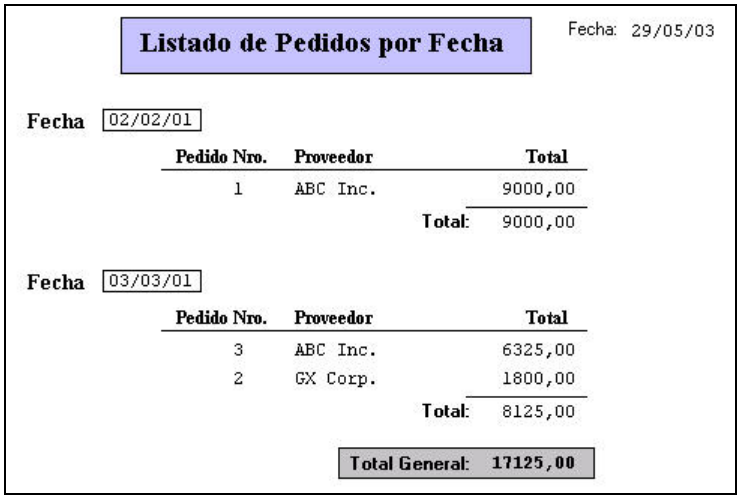


Fig. 45: Listado de Pedidos por Fecha en ejecución

El Layout es:

Fig. 46: Layout del reporte "Listado de Pedidos por Fecha"

y en el Source tendremos:

```
header
   print Cabezal
end
&TotGral = 0
for each order PedFch
   &TotFch = 0
   print Fecha
   for each
      &TotFch += PedTot
      print Pedidos
   endfor
   print TotFch
   &TotGral += &TotFch
endfor
print TotGral
```

Comandos de Control

Algunos comandos de control son los conocidos If-else-endif y Do while-enddo. El comando If-else-endif es muy utilizado para impresión condicional, es decir, imprimir una línea solo cuando se cumple alguna condición.

Por ejemplo, si queremos imprimir los pedidos por fecha, sin los totales, pero además queremos que se muestre la cantidad de pedidos que superaron los $2000 de importe total, agregaremos una variable *&Cnt* y en el Source haremos:

```
&Cnt = 0
for each order PedFch
   print Fecha
   for each
      print Pedidos
      if PedTot >= 2000
         &Cnt += 1
      endif
   endfor
endfor
print Cantidad
```

donde “Cantidad” sera el nombre de un print block que contendrá la variable *&Cnt*.
Observemos que aquí no utilizamos la cláusula else del comando if.

El comando Do while-enddo es muy usado para imprimir varias vías, por ejemplo si se quiere imprimir dos vías del pedido.

```
For each order PedNro
         &i = 0
         Do while &i < 2
     &i += 1
     // imprimir pedido
     ...
  enddo
endfor
```

Otros comandos de control son *Do case*, *For to Step*, *For in Array*, *For in Collection*.

Llamadas a otros Programas y a Subrutinas

Con el comando *Call* se puede llamar a otro programa. Este puede ser otro programa GeneXus o un programa externo. El formato del comando es:

Call(Program_name, $Parameter_1$, ..., $Parameter_n$)

Donde Program_name es el nombre del programa llamado y $Parameter_1$ a $Parameter_n$ son los parámetros que se envían. Estos parámetros pueden ser atributos, variables y constantes. Los parámetros pueden ser actualizados en el programa llamado.
Cuando desde un reporte se llama a otro reporte que también imprime, es importante pasarle como parámetro la variable predefinida *&Line* (que contiene el número de línea que se está imprimiendo) para que el reporte llamado no comience en una página nueva.
Con el comando *Do* se llama desde el código principal del reporte a una subrutina que debe estar definida mediante el comando *Sub*, al final del Source. En este caso no son necesarios los parámetros porque están disponibles para la subrutina los mismos datos (variables) que están disponibles en el lugar donde se hace el *Do*.

Así, en el Source tendremos:

```
....
DO 'Nombre-Subrutina'
....


Sub 'Nombre-Subrutina'
    ....
EndSub
```

Print if detail

Volvamos al listado de Pedidos por Proveedor, que estudiamos antes, y cuyo Layout mostrábamos en la fig.38. En el Source teníamos el siguiente comando For Each:

```
For each
   print Proveedor
   For each
      print Pedidos
   endFor
   print FinProv
endFor
```

El primer For Each tenía como tabla base PROVEEDORES pues en el print block “Proveedor” aparecían los atributos *PrvCod* y *PrvNom* y el segundo For Each tenía a PEDIDOS como tabla base, pues aparecían *PedNro*, *PedFch* y *PedTot*.
Ahora bien, el primer For Each es independiente del segundo y por lo tanto se imprimirán los datos del primer For Each ANTES de determinar si hay datos relacionados, en el segundo nivel y como consecuencia se van a imprimir TODOS los proveedores, independientemente de si tienen o no pedidos.
Si esto no es lo que se desea, es decir, si se quieren imprimir los pedidos por proveedor, pero solo de aquellos proveedores que tengan pedidos, entonces se puede utilizar el comando Print if detail:

```
For each order PrvCod
   Print if detail
   print Proveedor
   For each
      print Pedidos
   endFor
   print FinProv
endFor
```

Este comando define que el FOR EACH en donde se encuentre debe usar como tabla base la misma tabla base del siguiente FOR EACH. En el ejemplo, el primer FOR EACH pasará a estar basado en la tabla PEDIDOS y no en PROVEEDORES y por tanto solo se imprimirán los proveedores que tengan pedidos.
Se debe tener en cuenta que al estar los dos FOR EACHs basados en la misma tabla base, lo que se está definiendo implícitamente es un corte de control. Por lo tanto se DEBE definir explícitamente el ORDEN en el primer FOR EACH.

Existen otros comandos como ser EJECT (salto de página), NOSKIP, etc.

Condiciones

En esta sección de un reporte se definen las condiciones globales (filtros globales) que se quieren aplicar sobre los datos a recuperar.
Es equivalente a la cláusula WHERE del comando FOR EACH (incluso tienen la misma sintaxis), con la diferencia que el WHERE se aplica al FOR EACH en el que se encuentra y las condiciones definidas aquí se aplicarán a TODOS los FOR EACHs que correspondan. En el Listado de Navegación se indica a qué FOR EACH se están aplicando.
Muchas veces se requiere que las condiciones no sean con valores fijos, sino que el usuario elija los valores cuando se ejecuta el reporte.
Por ejemplo, si en el reporte que lista los proveedores del sistema, deseamos que se liste un rango de ellos en lugar de todos, podemos especificar las siguientes condiciones:

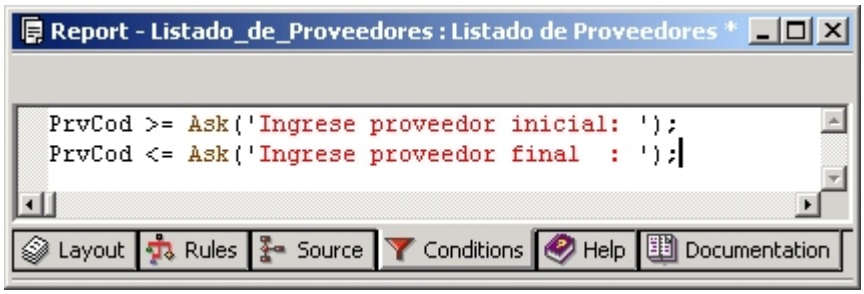


Fig. 47: Sección "Conditions" del reporte "Listado de Proveedores"

Cuando se ejecute el reporte aparecerá la siguiente pantalla:

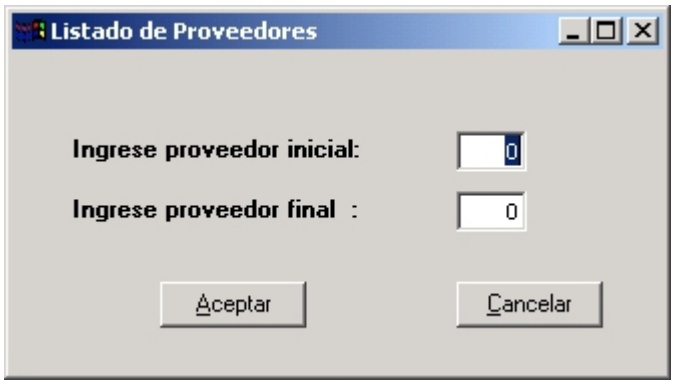


Fig. 48: Pantalla de ingreso de rango de proveedores para el listado

Donde se acepta el rango de proveedores a listar. El valor digitado por el usuario se almacena en las variables *&ask1*, *&ask2*, etc., según el orden de aparición de los ask en las condiciones. Estas variables son útiles para imprimir cuál fue el valor digitado.

Reglas

Se utilizan para definir aspectos generales del listado. Algunas de las reglas para reportes son:

Default

Permite definir valores por defecto para variables del reporte. Podemos definir, por ejemplo, valores por defecto para los 'ask':

default(*&ask1*, 1);

default(*&ask2*, 9999);

Las variables *&ask1* y *&ask2* deben definirse y el orden (1,2,...) es el de definición en las condiciones.

Parm

Es una lista de atributos o variables que recibe el reporte como parámetros. Por ejemplo, si hacemos un reporte para imprimir un pedido inmediatamente después de haberlo digitado, en la transacción "Pedidos"se declarará la regla:

call(RImpRec, ***PedNro***) On AfterComplete;

y el reporte será:

Source:

```
For each
  print CabezalDelPedido
  ....
  For each
    print LineasDelPedido
    ....
  endfor
endfor
```

Reglas:

Parm(*PedNro*);

Cuando en la regla *Parm* se recibe un atributo, automáticamente éste se toma como una condición (filtro) sobre los datos a recuperar.
En el ejemplo, al recibir como parámetro *PedNro*, el primer FOR EACH (que está basado en la tabla PEDIDOS que contiene *PedNro*) tiene como condición implícita que *PedNro* sea igual al valor recibido en el parámetro y no es necesario poner un WHERE.

Si el parámetro fuera, en cambio, una variable entonces sí sería necesario definir el WHERE:

Source:

```
For each
  WHERE PedNro = &Pedido
     print CabezalDelPedido
     ....
    For each
      print LineasDelPedido
      ....
    endfor
endfor
```

Reglas:

Parm(*&Pedido*);

Report Wizard

El Report Wizard es un asistente para la creación de reportes, de manera de hacer esta tarea más fácil y rápida.
Este asistente se abre automáticamente al crear un nuevo reporte, y el analista puede cancelarlo si prefiere realizar todas las operaciones manualmente, sin ayuda.
El asistente es útil a la hora de crear un reporte que contenga For Eachs, tanto anidados como paralelos.

Permite diseñar el Layout y el Source de una forma bien sencilla.
Se define a partir de una estructura de datos muy similar a las transacciones.

Diseñemos nuevamente el reporte de "Pedidos por proveedor".

El diseño consiste de cuatro pasos, que se enumeran a continuación.

Paso 1: Requiere que se provea de una estructura de datos. Observar que dicha estructura de datos debe incluir atributos definidos en las transacciones. La estructura de datos usará una estructura sintáctica similar a la usada en las transacciones, sin embargo, los paréntesis se usan para indicar niveles de FOR EACH (en vez de niveles de una transacción) y el asterisco ('*') se usa para indicar el orden deseado (y no para indicar la unicidad como en las transacciones). Las reglas que son usadas para definir la estructura de una transacción también se aplican al definir esta estructura.
La idea es que por cada "nivel" se podrá crear un print block en el Layout que contendrá los atributos de ese nivel y un for each en el Source que mandará a imprimir ese print block en la salida.

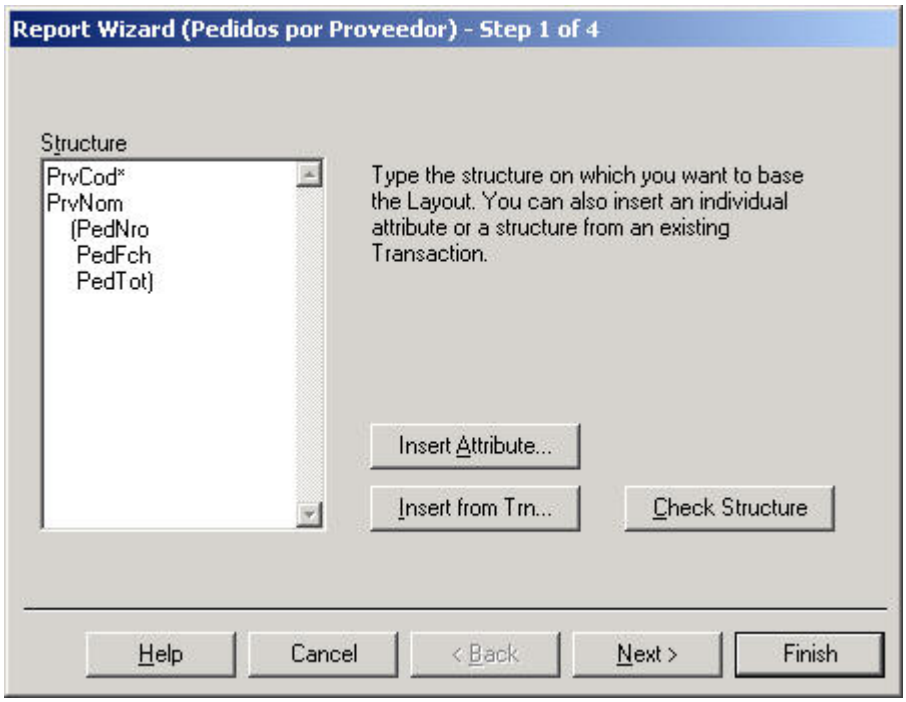


Fig. 49: Paso 1 del Report Wizard

Para diseñar nuestro reporte definimos una estructura en donde en el primer nivel tenemos los datos del proveedor y en el segundo los del pedido. Esto va a generar dos For Eachs anidados en donde el exterior va a estar ordenado por código de proveedor.
Una vez que se ha definido correctamente la estructura, se presiona el botón Next para pasar al siguiente paso.

Paso 2: En este paso se permite especificar cosas tales como si se desea crear por cada nivel de la estructura ingresada en el paso 1 solo los print blocks, solo el código en el Source (los for eachs y cabezales) o ambas cosas, a dónde se quiere enviar la salida del reporte (que el usuario decida en tiempo de ejecución, o solo a archivo, solo a impresora o solo a pantalla), etc.

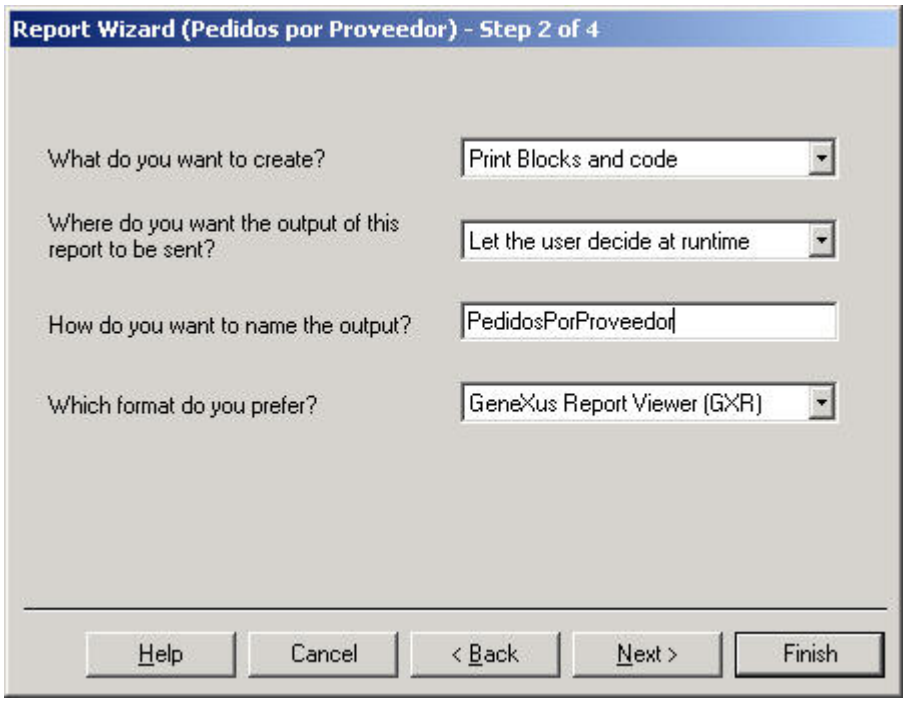


Fig. 50: Paso 2 del Report Wizard

Paso 3: Permite definir algunas características del Layout. Se permite elegir los fonts para representar los atributos o textos, también se permite definir si los cabezales de los atributos para cada uno de los niveles se despliegan horizontal o verticalmente.

El diálogo del paso 3 para la estructura de datos usada en el paso 1 se ve de la siguiente manera:

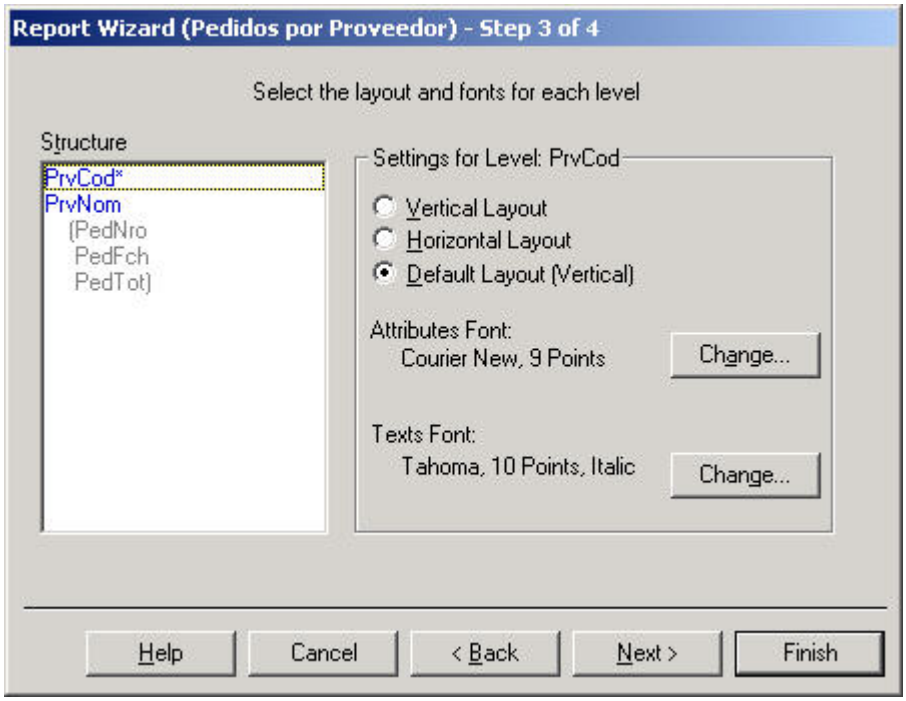


Fig. 51: Paso 3 del Report Wizard

Como se puede ver, el nivel para el cual estamos definiendo el tipo del Layout se presenta en un color diferente. Para este nivel, dejamos el tipo por defecto para ese nivel del Layout y para el segundo nivel nos aseguraremos que el check box correspondiente al tipo de Layout vertical no esté marcado. De esta manera el display del nivel interior será horizontal.

Paso 4: Este paso permite especificar condiciones para cada nivel incluido en el reporte, que serán traducidas en cláusulas where del For Each correspondiente al nivel en el Source.

Para ello se debe seleccionar el nivel y colocar las condiciones aplicables a ese nivel.

Por ejemplo, si no queremos imprimir todos los proveedores con sus pedidos, sino solo aquellos cuyo código es mayor que 100:

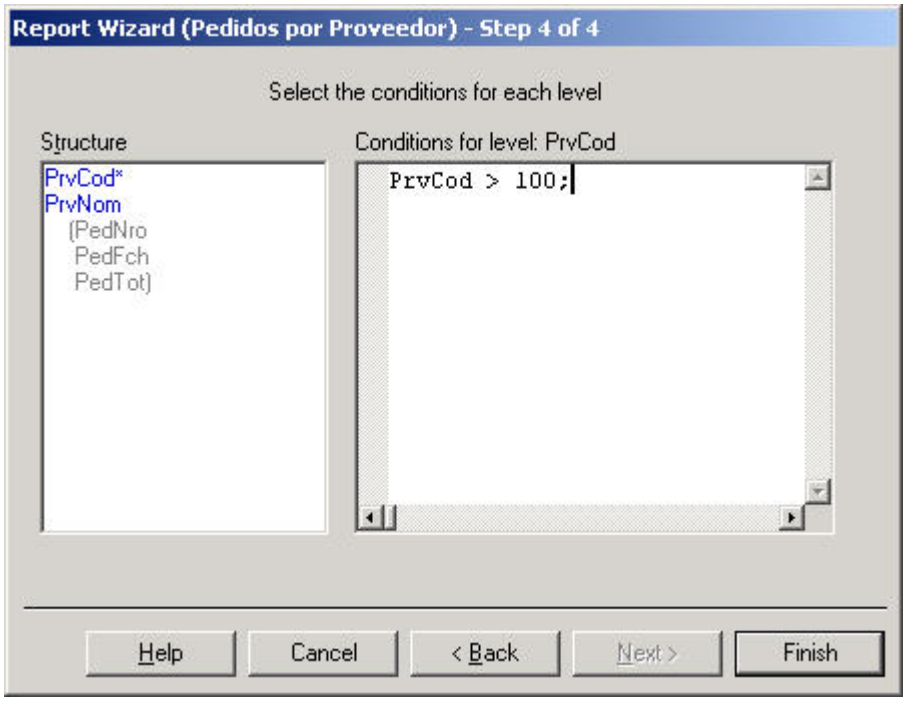


Fig. 52: Paso 4 del Report Wizard

Presionando el botón Finish se graban las definiciones para el paso actual y se sale del diálogo del Report Wizard.

Una vez que todas las opciones del Wizard fueron aceptadas, se genera el Layout y el Source del reporte, que luego pueden ser modificados como en cualquier otro reporte.

En el ejemplo, luego de dar Finish al Wizard se genera el reporte, con el siguiente Layout:

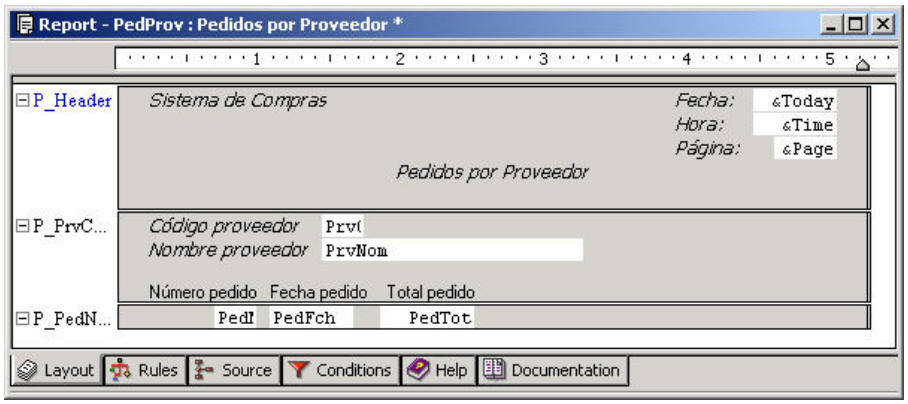


Fig. 53: Layout del reporte generado con el Report Wizard

Aquí aparecen 3 print blocks de nombres “P_Header”, “P_PrvCod” y “P_PedNro”. Para darle nombre a cada print block de la estructura del Wizard, se utiliza el prefijo “P_” seguido del primer atributo que figure en el print block. Se incluye un cabezal, que se nombra “P_Header” como vemos.

El Source de este reporte aparecerá inicializado:

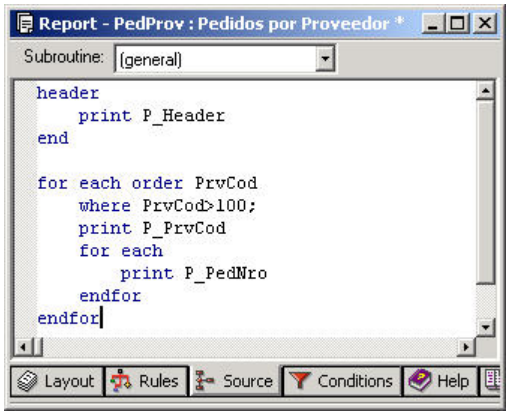


Fig. 54: Source del reporte generado con el Report Wizard

Propiedades

Como cualquier otro objeto GeneXus, los reportes también tienen propiedades que permiten definir el comportamiento general del mismo:

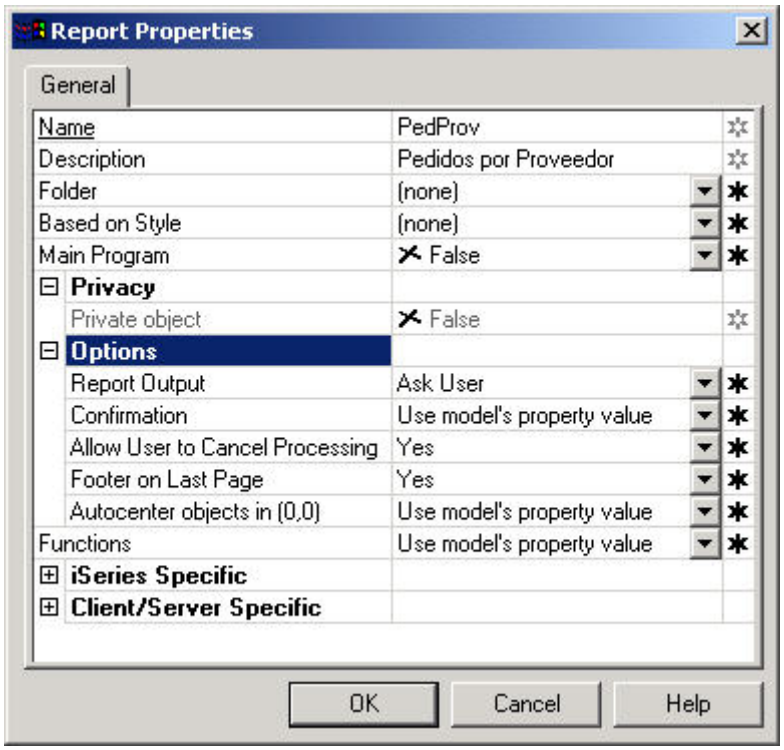


Fig. 55: Diálogo de Propiedades de un reporte

Entre las propiedades que podemos definir con el editor de propiedades de los reportes tenemos:
Report Output: permite especificar si la salida del reporte será enviada directamente a la pantalla, impresora, a un archivo o si en cambio se le debe preguntar al usuario en tiempo de ejecución.
Las opciones posibles son:

- Ask User
- Only to File
- Only to Printer
- Only to Screen

4. Reportes Dinámicos – GeneXus Query

Los reportes dinámicos permiten al usuario final extraer directamente información de la base de datos, sin necesidad de tener conocimientos técnicos, y sin necesidad de que los analistas realicen ningún desarrollo adicional.

Los reportes dinámicos se realizan con GeneXus Query, un "Add-In" de Excel que se agrega automaticamente al mismo al instalar GeneXus Query.
El usuario ingresa a Excel y a través de la barra de herramientas de GeneXus Query puede formular sus propias consultas seleccionando los atributos a consultar, sin necesidad de especificar tablas ni navegaciones.
El resultado de la consulta se devuelve en una "pivot table" o en una "query table" dentro de una planilla de Excel, permitiendo luego al usuario manipular esa información con toda la potencia de la planilla electrónica.

En momento de creación o reorganización de la base de datos con GeneXus, se genera automáticamente la "metadata", permitiéndole al usuario que formule a partir de ese mismo momento sus consultas dinámicamente desde Excel con GeneXus Query. No es necesario realizar ningún desarrollo adicional para ello.

El usuario puede además de formular sus consultas, catalogarlas organizándolas en carpetas (folders) para su uso posterior; graficarlas, publicar el resultado en Internet, definir seguridad tanto sobre atributos como sobre consultas, etc.

Formulación de consultas

Para formular una consulta, el usuario ingresa a Excel y selecciona en la barra de herramientas de GeneXus Query la opción "New Query"
Se le presenta una pantalla donde a la izquierda aparecen todos los atributos que el usuario tiene habilitados para consultar[6], para que seleccione los atributos que desea consultar (los cuales se van colocando del lado derecho a medida que los va seleccionando).

[6] Podrá definirse sus propios folders en la lista de atributos, organizando los atributos que usa en la forma más convenviente para él, etc.

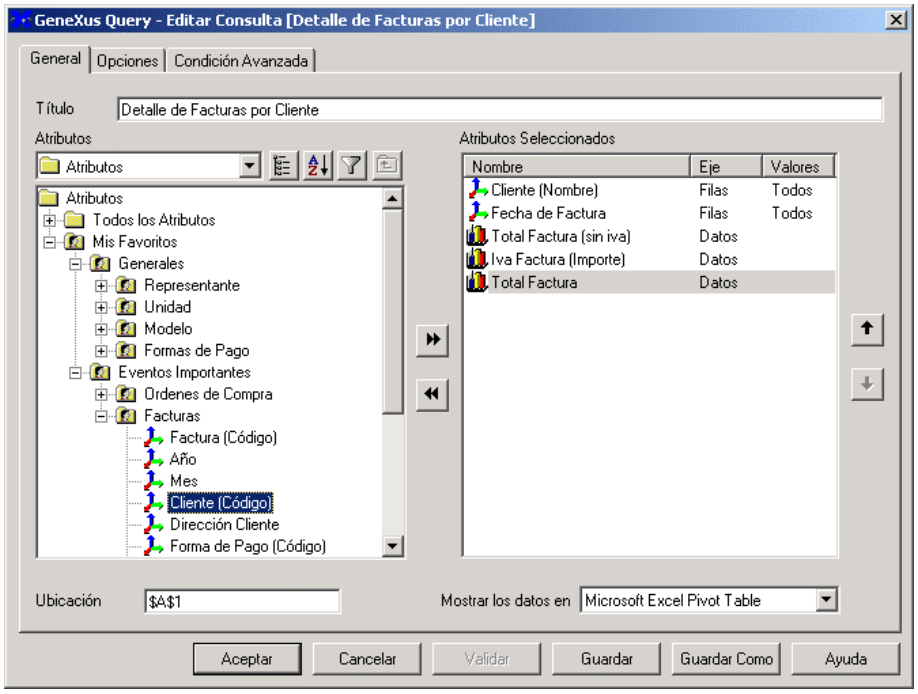


Fig. 56: Query "Detalle de Facturas por Cliente"

El usuario puede cambiar la función que se le aplicará a los atributos seleccionados (por ejemplo: podrá indicar promedios, porcentajes, contar valores de un atributo, etc.). Podrá también restringir los valores a mostrar en la consulta, especificando filtros a los atributos de la consulta (lista de valores, condiciones complejas, etc.).
Podrá definir alertas (marcar los valores que cumplan determinadas condiciones en determinado color).

Una vez especificada la consulta, se ejecuta la misma y se presenta el resultado en Excel de la siguiente forma:

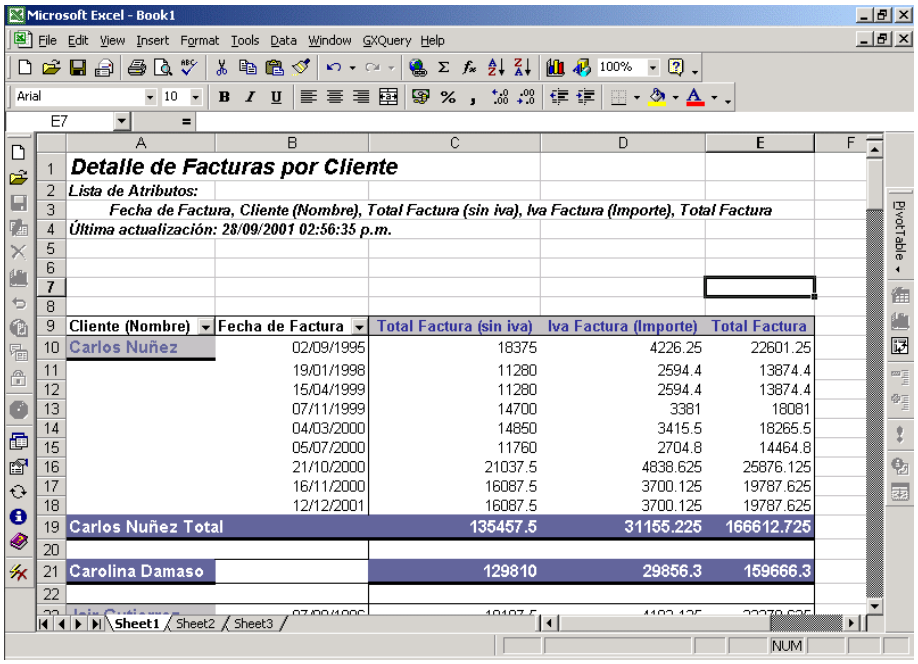


Fig. 57: Resultado de la consulta como Pívot Table en Excel

Sobre esta tabla dinámica el usuario podrá manipular los datos de la consulta, pivoteando, filtrando valores, definiendo fórmulas, graficando, etc.
Además, podrá catalogar la misma, publicarla en Internet, etc.

A diferencia de los "Query Tool" tradicionales, que requieren que el usuario seleccione campos de tablas y determine en algunos casos cómo se deben realizar los "joins", con GeneXus Query solamente se seleccionan los atributos a consultar, y la herramienta determina automáticamente a qué tablas acceder, qué navegaciones realizar, qué restricciones aplicar, etc. Además, se provee el manejo de catálogos de consultas, seguridad a diferentes niveles, impacto automático de la metadata y de las consultas catalogadas ante cambios en la base de datos, etc.

El objetivo de GeneXus Query es sustituir gran parte de los reportes de nuestras aplicaciones, que se desarrollan en forma específica y contínua, por consultas dinámicas. De esta forma se da flexibilidad y potencia a los usuarios finales y se libera a los analistas de la tarea de confeccionar esos reportes a medida.

5. Objeto Procedimiento

El procedimiento es el objeto GeneXus con el que se definen todos los procesos no interactivos de actualización de la base de datos y las subrutinas de uso general.
Este objeto tiene todas las características de los reportes (cualquier reporte se puede implementar como un procedimiento) y algunas características particulares para la actualización de la base de datos. Por esta razón se recomienda haber leído previamente el capítulo “Objeto Reporte”.
Vamos a ver entonces los comandos disponibles en los procedimientos para la actualización de la base de datos.

Modificación de datos

La modificación de datos de la base de datos se realiza de forma implícita, ya que no hay un comando específico de modificación (como podría ser REWRITE), sino que basta con asignar un valor a un atributo dentro de un FOR EACH para que automáticamente se realice la modificación.
Por ejemplo supongamos que queremos tener un proceso batch que actualiza el atributo *ArtPrcUltCmp* para adecuarlo según la inflación para todos los artículos almacenados:

Programa:

```
For each
      ArtPrcUltCmp = ArtPrcUltCmp * (1 + &Inf)
endfor
```

Reglas:

```
Parm( &Inf );
```

Se puede modificar más de un atributo dentro del mismo FOR EACH, por ejemplo, si también queremos modificar el atributo *ArtFchUltCmp*, poniéndole la fecha de hoy:

```
For each
 ArtPrcUltCmp *= (1 + &Inf)
      ArtFchUltCmp = Today( )
endfor
```

La modificación física no se realiza inmediatamente después de cada asignación, sino en el endfor.
En estos dos ejemplos se modificaron atributos de la tabla base del FOR EACH, pero también es posible actualizar atributos de las tablas pertenecientes a la extendida, utilizando el mismo mecanismo de asignación de atributos.

No todos los atributos pueden ser modificados. Por ejemplo, no se pueden modificar los pertenecientes a la clave primaria de la tabla base del FOR EACH (en este caso *ArtCod*). Tampoco se pueden modificar los atributos que pertenecen al índice (el orden) con el que se está accediendo a la tabla base.

Si se quiere actualizar un atributo que es clave candidata, se controlará la unicidad del nuevo valor. Para entender mejor este punto, volvamos a la observación que efectuamos en el capítulo de transacciones: si por algún motivo dejamos como identificador del 2do nivel de la transacción “Pedidos” el atributo *PedLinNro*, en lugar de *ArtCod*, pero no queremos permitir que se repitan los artículos en las líneas de un pedido, dijimos que una solución posible era definir un índice compuesto por *PedNro* y *ArtCod*, de tipo UNIQUE (con esto establecemos que *PedNro* y *ArtCod* forman una clave candidata).

Supongamos ahora, que el artículo de código 2 no se está fabricando más, y que existe uno parecido, el 5. Debemos cambiar los pedidos, de forma tal de sustituir las líneas donde se encontrara el artículo 2, por el artículo 5. Haríamos:

```
For each
Where ArtCod = 2
Defined by PedCnt
        ArtCod = 5
endfor
```

Pero si para un pedido dado existían líneas con los dos artículos, el 2 y el 5, cuando se quiera realizar la asignación dentro del For Each, se encontrará que ya existe un registro con esos valores de *PedNro*, *ArtCod*. Pero esos valores deben ser únicos, por lo que no se realizará la actualización para ese registro. Si se quiere realizar alguna acción cuando se encuentra duplicada una clave candidata, se utiliza la cláusula WHEN DUPLICATE del for each:

```
For each
Where ArtCod = 2
Defined by PedCnt
        ArtCod = 5
WHEN DUPLICATE
        Msg( 'Ya existe una línea con ese artículo')
endfor
```

En el Listado de Navegación se informa cuáles son las tablas que se están actualizando marcando con un símbolo correspondiente a un lápiz estas tablas. En el ejemplo de actualización del precio y fecha de la última compra de un artículo el Listado de Navegación nos muestra:

Fig. 58: Listado de Navegación del procedimiento de actualización de Artículos

Eliminación de datos

Para eliminar datos se utiliza el comando DELETE dentro del grupo FOR EACH-ENDFOR. Por ejemplo, si queremos eliminar todos los pedidos con fecha anterior a una fecha dada:

```
For each
where PedFch < ask( 'Eliminar pedidos con fecha menor a:' )
        For each
        defined by PedCnt
                // Se eliminan todas las líneas del pedido
                DELETE
        endfor
        // Después de eliminar las líneas se elimina el cabezal
        DELETE
endfor
```

Creación de datos

Para crear nuevos registros en la base de datos se usa el comando NEW-ENDNEW. Tomemos por ejemplo un procedimiento que recibe como parámetros *AnlNro* y *AnlNom* en las variables *&Nro* y *&Nom* respectivamente, e inserta estos datos en la tabla de analistas:

Programa:

```
New
   AnlNro = &Nro
   AnlNom = &Nom
 Endnew
```

Reglas:

```
Parm( &Nro, &Nom );
```

Con este procedimiento se pueden crear registros en la tabla de analistas.
El comando NEW realiza el control de duplicados, tanto para la clave primaria como para las claves candidatas y solo creará el registro si ya no se encuentra en la tabla un registro con el mismo valor de clave primaria y claves candidatas. Con la cláusula WHEN DUPLICATE se programa la acción a realizar en caso de duplicados. Por ejemplo, en el caso anterior si se quiere que si el analista ya estaba registrado se actualice el *AnlNom*:

```
New
      AnlNro = &Nro
      AnlNom = &Nom
 When Duplicate
      For each
          AnlNom = &Nom
      endfor
endnew
```

Observar que para realizar la modificación de atributos las asignaciones se deben encontrar DENTRO de un grupo FOR EACH-ENDFOR.

Controles de integridad y redundancia

En los procedimientos el único control de integridad que se realiza automáticamente es el control de duplicados.
En cambio, **no se controla automáticamente la integridad referencial**. Por ejemplo, si hacemos un procedimiento que crea automáticamente pedidos, no se controlará que el *PrvCod* que se cargue exista en la tabla de proveedores. O cuando se elimina el cabezal de un pedido, se debe tener esto en cuenta para eliminar las líneas del mismo explícitamente.
El mismo criterio se aplica para las fórmulas redundantes. Éstas deben ser mantenidas explícitamente con los comandos de asignación/update.

6. Objeto Work Panel

Con work panels (paneles de trabajo) se definen las consultas interactivas a la base de datos, en ambiente Windows. Si bien este es un objeto muy flexible que se presta para múltiples usos, es especialmente útil para aquellos usuarios que utilizan el computador para pensar o para tomar decisiones. La forma de programar los work panels está inspirada en la programación de los ambientes gráficos, especialmente la programación dirigida por eventos.

Elementos

Para cada work panel se puede definir:

Form

Al igual que las transacciones, se debe definir el form, que será la interfaz con el usuario. Pero a diferencia de las transacciones, los work panels son objetos que solo se utilizan en ambiente Windows, por lo que el form de un work panel es análogo al form GUI-Windows de una transacción. Para programar una pantalla Web para consulta de datos, se utilizará el objeto web panel, que se mencionará más adelante.

Eventos

Aquí se define la respuesta a los eventos que pueden ocurrir durante la ejecución del work panel. Por ejemplo, qué respuesta dar cuando el usuario presiona Enter o un botón, etc.

Condiciones

Define qué condiciones deben cumplir los datos para ser presentados en la pantalla. Son equivalentes a las Condiciones de los reportes y procedimientos.

Reglas/Propiedades

Definen aspectos generales del work panel, como los parámetros que recibe, etc.

Ayuda

Texto para la ayuda a los usuarios en el uso del work panel.

Documentación

Texto técnico para ser utilizado en la documentación del sistema.

Solapas de acceso a los elementos

Como para todo objeto GeneXus, se puede acceder a varios de los elementos que lo componen utilizando las solapas que aparecen bajo la ventana de edición del objeto:

Fig. 59: Solapas que figuran en la parte inferior de la ventana de edición de un work panel

En este capítulo solo se presentarán las características y usos más salientes de los work panels. Un estudio más profundo será realizado en el curso GeneXus.

Ejemplo

Para ilustrar el uso de este tipo de objetos, vamos a realizar una serie de consultas encadenadas. Estas consultas comienzan con un work panel de proveedores, donde se despliegan todos los proveedores y el usuario selecciona uno para luego realizar alguna acción con él. En este caso para cada proveedor seleccionado se podrá:

- **Visualizar los datos del proveedor**
- **Visualizar los pedidos del proveedor**

Al work panel lo llamaremos “Trabajar con Proveedores” dado que es precisamente lo que buscamos: trabajar -efectuar alguna acción- con los proveedores.

El form en ejecución se verá:

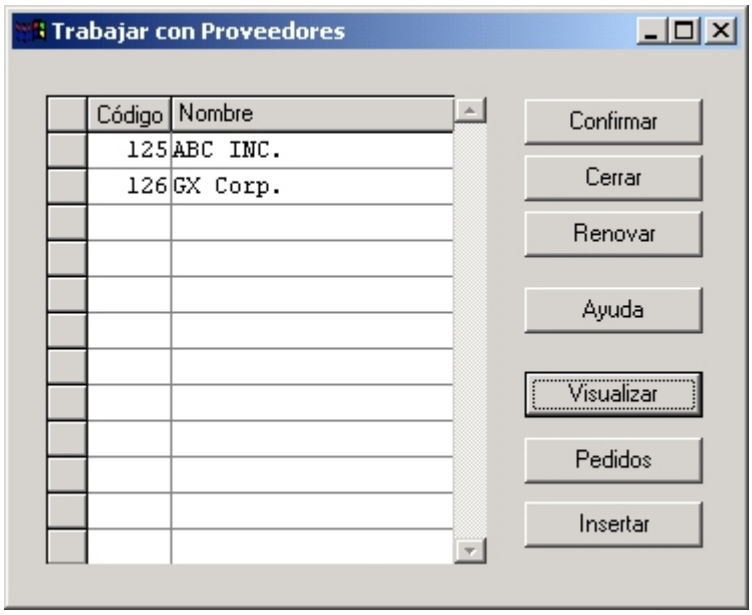


Fig. 60: Form de “Trabajar con Proveedores” en ejecución

Aquí se muestra el código y nombre de todos los proveedores del sistema, en la grilla. Luego el usuario puede posicionarse sobre una línea y hacer algo con ese proveedor.
Por ejemplo, si el usuario presiona el botón de “Visualizar” en el proveedor 125, se abre una ventana donde se muestran los datos de ese proveedor:

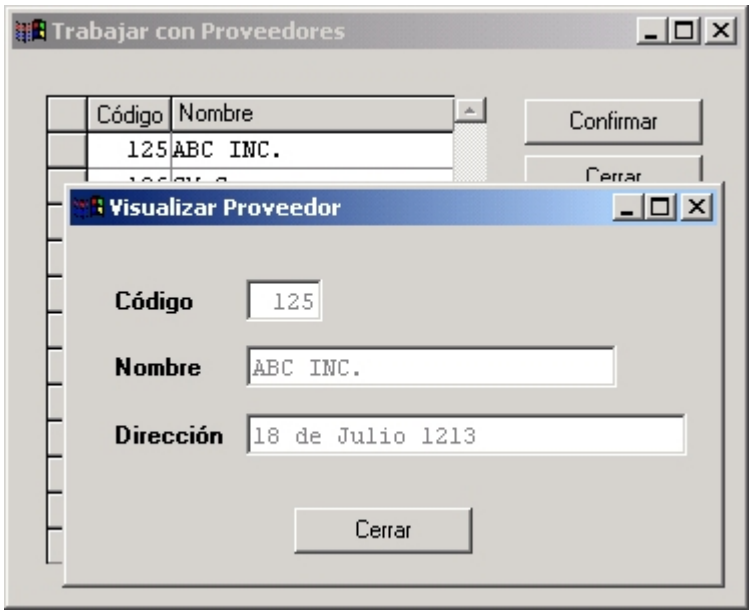


Fig. 61: Visualizar datos de un proveedor

Análogamente, si se presiona el botón “Pedidos” se abrirá la siguiente pantalla que muestra los pedidos de ese proveedor:

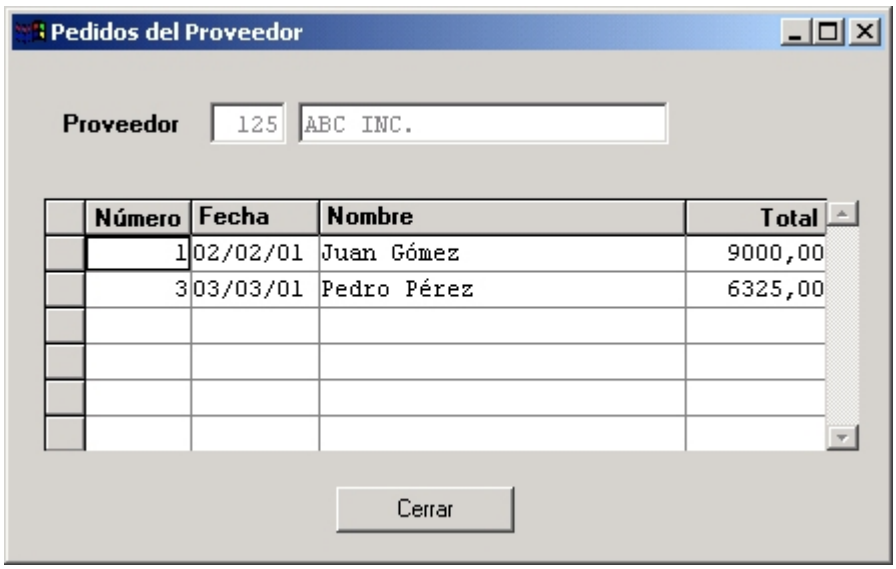


Fig. 62: Visualizar pedidos de un proveedor

A su vez este work panel podría extenderse, permitiendo seleccionar algún pedido para visualizar qué artículos lo componen y así sucesivamente.

Las acciones que vimos en el primer work panel se aplican a una línea, pero existen otras acciones que no se aplican a ninguna línea en particular, por ejemplo, si queremos insertar un nuevo proveedor. Si se elige esta acción (presionando el botón “Insertar” que figura en el form) se pasa el control a la transacción de proveedores para permitir el ingreso de un nuevo proveedor:

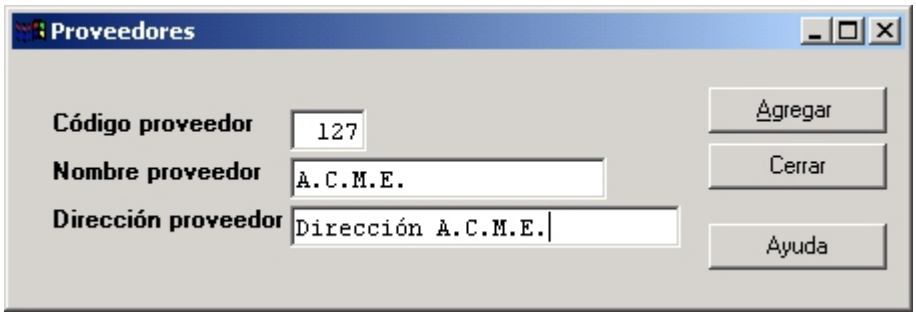


Fig. 63: Insertar un nuevo proveedor

Botón “Renovar”

Es incluido automáticamente por GeneXus en el form de todo work panel y la consecuencia de presionarlo es que se vuelven a cargar todas las líneas del grid. Como veremos más adelante existen varios casos en los que puede resultar necesario “renovar” o “refrescar” los datos cargados. Por ejemplo cuando se agrega un nuevo proveedor al sistema. Más adelante nos detendremos en esta acción.

Veamos ahora cómo se define el primer work panel del ejemplo.

Form

El form de un work panel se define de forma similar al form GUI-Windows de una transacción.

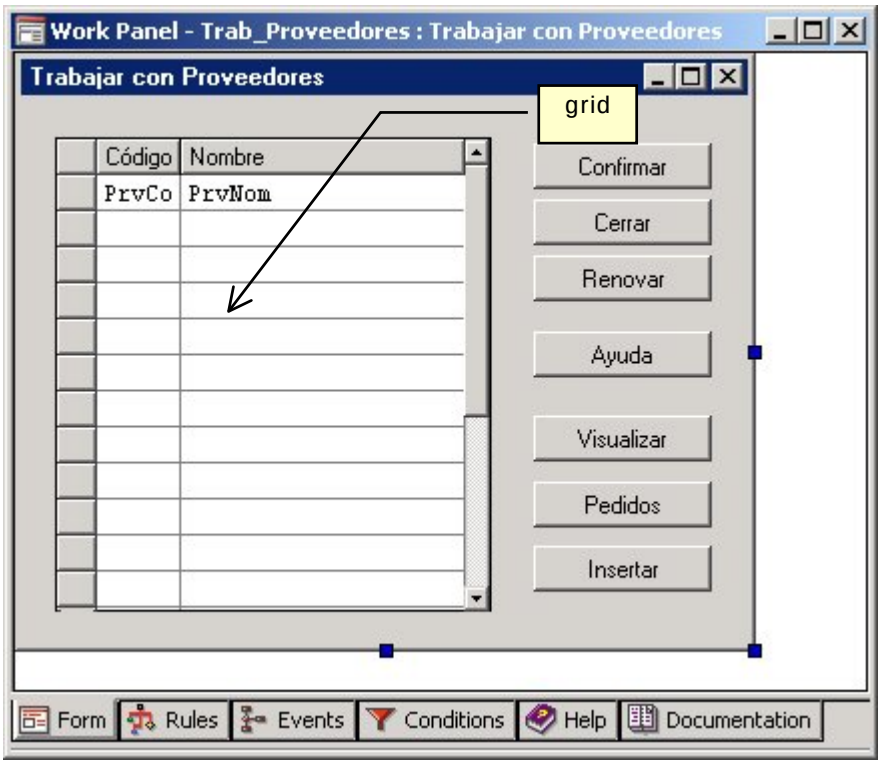


Fig. 64: Form del work panel "Trabajar con Proveedores"

Cuando se define un grid en el form (se inserta como cualquier otro control, utilizando la paleta de herramientas "Controls") se está indicando que se va a mostrar una cantidad indefinida de datos (en este caso, los proveedores). Si bien sólo se pueden ver simultáneamente las líneas presentes en la pantalla, GeneXus provee todos los mecanismos para que el usuario se pueda "mover" (llamaremos "paginar") dentro de la lista.
El work panel anterior contiene un grid cuyas columnas corresponden a los atributos *PrvCod* y *PrvNom*. Los grids pueden contener atributos, variables y elementos de arrays de una y dos dimensiones. A diferencia de las transacciones, aquí los atributos serán de salida, es decir, se utilizan para mostrar información de la base de datos, y las variables de entrada (se utilizan fundamentalmente para solicitar información al usuario).

Grid

El ícono de la paleta de herramientas que se muestra a continuación, nos va a permitir insertar un grid en el form del work panel y definir las caracterísitcas del mismo, como los atributos y variables que conformarán las columnas del grid, los títulos de tales columnas, etc.

Fig. 65: Paleta de Herramientas "Controls" para un work panel

Una vez insertado el control en el form, se abre automáticamente el siguiente diálogo, donde se especifican las columnas:

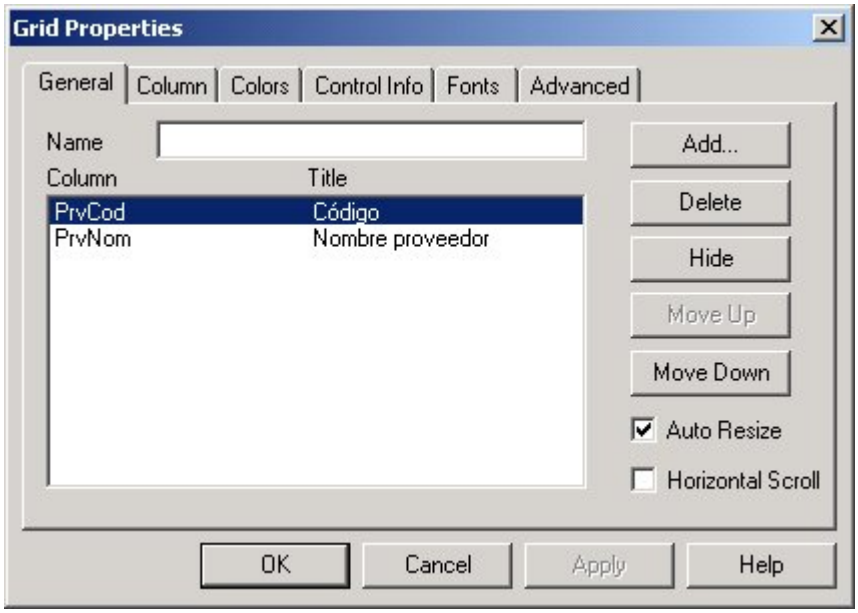


Fig. 66: Diálogo de Propiedades del control grid

En “Name” se permite dar un nombre al control, lo que será necesario si luego se le quieren asociar propiedades, eventos o métodos al mismo. Si se van a incluir varios grids dentro del form, entonces será indispensable identificarlos, por lo que se vuelve fundamental asignar valor a estar propiedad para cada uno de los grids insertados.
Para seleccionar los atributos y variables que van a ser incluidos en el grid se debe presionar el boton “Add”.

El botón “Hide” nos permite ocultar la columna seleccionada. Esto resultará necesario cuando no se quieran mostrar en pantalla determinados datos, pero sea indispensable tenerlos cargados, para poder luego utilizar sus valores cuando se seleccione una línea y se quiera efectuar alguna acción sobre la misma.

Los botones “MoveUp” y “Move Down” nos van a permitir cambiar el orden en que se van a desplegar los atributos en el grid.

Con la solapa “Column” vamos a poder cambiar los títulos de las columnas (por defecto, para un atributo toma el valor de la opción “Title Column” del atributo, ver fig.5 del capítulo 2) y en caso de tratarse de una variable no escalar, podremos seleccionar el elemento de la variable que queremos utilizar en esa columna, indicando valores de fila y columna.

Con las solapas “Colors” y “Fonts” podemos cambiar los colores y el tipo de letra de las columnas y de las filas.

Con la solapa “Control Info” se selecciona el tipo de control a utilizar para cada atributo o variable del grid, y dado el tipo, indicar la información necesaria para ese tipo. Las opciones disponibles son: Edit, Check box, Combo y Dynamic Combo (o el default del atributo o variable de la columna).

La solapa “Advanced” nos va a permitir configurar ciertas propiedades que tienen que ver con la carga del grid. Si no se configuran, entonces valen para ese grid las opciones a nivel del objeto, que son las que figuran bajo la categoría “Loading” en el diálogo de propiedades del work panel, que se ilustra más adelante, en la fig.77.

Podemos también ajustar manualmente el ancho de cada columna desmarcando la opción de “Auto Resize”. Si esta opción queda marcada, el ancho de la columna es determinado por

GeneXus en función del largo del atributo o variable y del título de la columna.

El check box “Horizontal Scroll” permite que en tiempo de ejecución aparezca una barra de scroll horizontal (además de la de scroll vertical que aparece automáticamente).

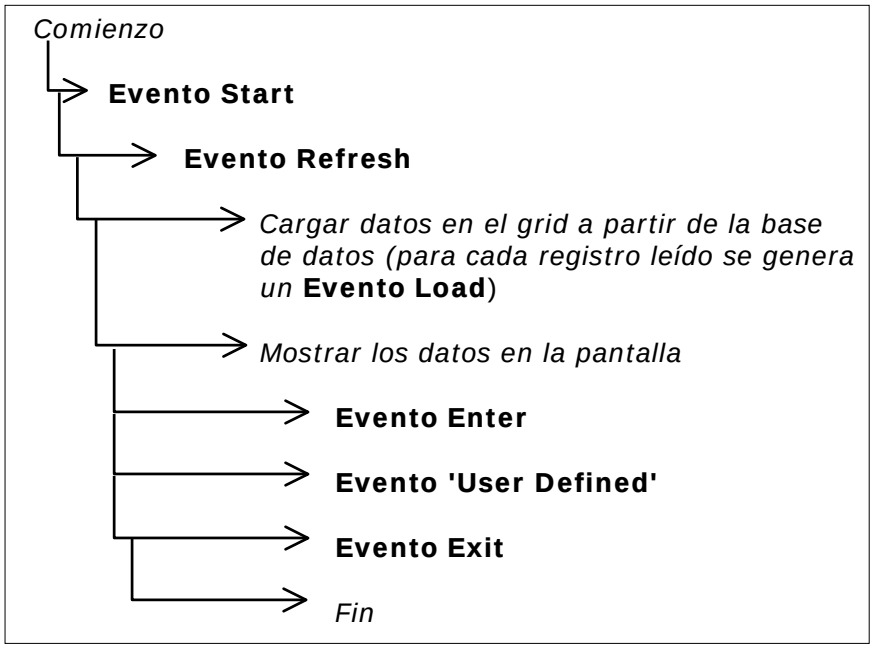
Eventos

La forma tradicional de programar la interfaz entre el usuario y el computador (incluidos los lenguajes de cuarta generación) es subordinar la pantalla al programa. Es decir, desde el programa se hacen “calls” a la pantalla. En work panels es lo opuesto: el programa está subordinado a la pantalla, ya que una pantalla realiza “calls” al programa para dar respuesta a un evento.

Esta forma de trabajar es conocida como *Event Driven* (dirigida por eventos) y es típica de los ambientes GUI (Graphic User Interface), en donde ya ha demostrado su productividad, fundamentalmente porque se necesita programar sólo la respuesta a los eventos y no la tarea repetitiva que implica el manejo de toda la interfaz.
A partir de la versión 7.5 de GeneXus se incorpora la posibilidad de definir varios grids dentro de un mismo work panel, lo que no era posible en versiones anteriores.

Hasta esa versión solo podían mostrarse en pantalla los datos de una tabla de la base de datos (y de su extendida), por lo que si queríamos listar información de varias tablas distintas, teníamos que dividir la información en distintos work panels.

Existen algunas diferencias entre work panels con un solo grid y work panels con múltiples grids, fundamentalmente con respecto a los eventos y a la determinación de las tablas que deben navegarse para recuperar los datos. En este libro trataremos el primer caso, es decir, aquel en el que el work panel cuenta a lo sumo con un grid, y dejamos el estudio de múltiples grids para el curso GeneXus. La discusión que sigue toma en cuenta, pues, el caso de un solo grid.
La estructura de eventos es, en forma resumida, la siguiente:



Para programar cada uno de los eventos se utiliza el mismo lenguaje que vimos en reportes y procedimientos, aunque no están permitidos los comandos relacionados con la impresión (Header, Footer, etc.) ni los de actualización de la base de datos (New, Delete, etc.). También existen comandos específicos para work panels, como lo son el comando Refresh y el comando Load.

Evento Start

El evento Start ocurre cuando se abre el work panel.

Ejemplo:

```
Event Start
        &Fecha = &Today
Endevent
```

En este caso, se utiliza el evento Start para mover la fecha actual a la variable *&Fecha*, que por ejemplo, se podrá mostrar en la pantalla. Esta técnica es muy útil para inicializar valores por defecto, sobre todo para aquellos que son complejos (por ejemplo, *&Valor* = udf('PX', ...)).

Evento Enter

Este evento se dispara cuando el usuario presiona ENTER (o el botón "Confirmar"). Supongamos que en el work panel "Trabajar con proveedores", deseamos agregar la posibilidad de eliminar un conjunto de proveedores. Podemos implementar esto definiendo una nueva columna en el grid, compuesta por una variable, *&op*, para que el usuario ingrese allí el tipo de operación que quiere efectuar sobre cada línea. Supongamos que un valor "4" en la variable *&op* significa que se quiere eliminar ese proveedor. Programaremos el evento "Enter" de manera tal de recorrer el grid y para cada línea que tenga el código de operación "4" llamar a un procedimiento que borre ese proveedor. El código asociado será:

```
Event Enter
    For each line
        If &op = '4'
            Call(PDltPrv, PrvCod)
        else
            If &op <> ' '
                Msg(concat(&op, ' no es una opción válida') )
            Endif
        endif
    endfor
endevent
```

El evento Enter es un poco más extenso y vale la pena detenernos en él. En primer lugar se utiliza el comando FOR EACH LINE que realiza un FOR EACH, pero no sobre los datos de la base de datos, sino sobre los datos cargados en el grid.
Para cada una de las líneas del grid, se pregunta por el valor de *&op*. Si es '4' se llama al procedimiento DltPrv, y en otro caso, se da un mensaje indicando que la opción no es válida. Observar que se podrían haber definido más opciones y preguntar por ellas en este evento.

Eventos definidos por el usuario (User Defined Events)

Fig. 67: Evento 'Insertar' en work panel "Trabajar con Proveedores"

Este es un evento definido por el usuario, al que se le debe asociar un nombre, en este caso, 'Insertar'. En este evento se llama a la transacción de proveedores. Luego de llamar a la transacción, se ejecuta el comando **Refresh**, que indica que se debe cargar nuevamente el grid, ya que se ha adicionado un nuevo proveedor y queremos que éste aparezca en la pantalla. También se podría haber usado el comando con el parámetro keep (**Refresh Keep)** que hace que una vez que el comando refresh se ejecute, el cursor quede posicionado en la misma línea del grid en la que estaba antes de la ejecución.

Evento Refresh

Los datos presentados en pantalla, son cargados en el grid desde la base de datos cuando se abre el work panel. La interacción con la base de datos se da en el momento de la carga, y luego se trabajará sobre los datos cargados.

En un ambiente multi-usuario, los datos de una pantalla pueden haber quedado desactualizados si desde otra terminal fueron modificados. Cuando el usuario desea actualizar los datos, debe dar clic sobre el botón Refresh ("Renovar" en español), o presionar la tecla F5. El efecto es que se vuelve a cargar el grid, accediendo nuevamente a la base de datos.

El evento Refresh (evento del sistema) ocurre en el instante de tiempo que antecede a la carga del grid. Inmediatamente después de producido, se accede a la base de datos y se carga el grid. Por lo tanto, como consecuencia del disparo del evento Refresh, los datos son cargados y mostrados en pantalla.

En algunos casos, desde otro evento también se necesita cargar nuevamente el grid. Por ejemplo, cuando en otro evento se llama a una transacción que actualiza los datos mostrados en el grid (es el caso del evento 'Insertar' de la figura anterior). Para ello se provee el comando Refresh, cuyo efecto es disparar el evento de igual nombre.

Tenemos, por lo tanto, que el grid se carga (es decir, ocurre el evento Refresh) cuando:

- Se carga la pantalla - por lo tanto después del evento Start hay un evento Refresh.
- El usuario hizo clic sobre el botón Refresh (o presionó la tecla F5).
- Se ejecutó el comando Refresh en otro evento.
- El usuario modificó el valor de alguna de las variables de la parte fija del form, que son utilizadas como condiciones sobre los datos a cargar (esto lo veremos unas páginas más adelante)

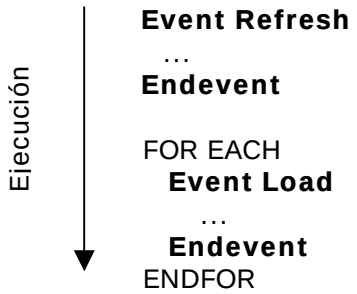
Carga de datos en el work panel

Como vimos en el diagrama de eventos, después del evento Refresh, se carga el grid con los datos obtenidos de la base de datos.
Para determinar de qué tablas se deben obtener los datos, GeneXus utiliza el mismo mecanismo descripto para los grupos FOR EACH en el capítulo de reportes. ¡Pero en este caso no hay ningún FOR EACH!
Sin embargo, generalmente se considera que cada work panel con un grid tiene un FOR EACH implícito[7], que contiene los siguientes atributos:

- Todos los atributos que se encuentran en el form (incluso los que se encuentran en la parte fija del mismo).
- Todos los atributos que se encuentren en los Eventos, con la excepción de aquellos que se encuentren dentro de un FOR EACH explícito.
- Todos los atributos del grid que estén ocultos, es decir, aquellos para los que se presionó el botón "Hide" en la solapa "General" del diálogo de propiedades del grid.
- Todos los atributos del grid que se hayan insertado en la pantalla "Grid Order" a la que se accede mediante la opción "Order" del menú pop up que se despliega al hacer botón derecho sobre el grid, como veremos luego

Con estos atributos, GeneXus determina cuál es la tabla extendida correspondiente, de la misma manera en que lo hacía con los atributos que extraía de un For Each.

Luego recorre la tabla base y accede a las tablas de la extendida, para recuperar los valores de los atributos que aparecen en el grid, tanto visibles como ocultos, de la misma manera en que lo hacía con un print block dentro de un For Each en un reporte.
Para cada uno de los registros encontrados, se genera el evento del sistema Load. Así, la relación entre el evento Refresh, el FOR EACH implícito y el evento Load es la siguiente:



Los datos que se leen en este FOR EACH implícito se cargan en el grid, que es el que se presenta en pantalla. Esta carga se realiza "bajo pedido", es decir que al comienzo sólo se carga la primera página y a medida que el usuario requiere más información, se van cargando las siguientes páginas. Existe una propiedad para indicar que se quiere cargar el grid de una sola vez si fuera necesario (la propiedad "Load Records").
Igual que en los reportes y procedimientos, en el Listado de Navegación del work panel se indica qué tablas se están utilizando, índices, etc.

Puede darse el caso de que habiendo un grid en el form, no exista el FOR EACH implícito. Esto ocurre cuando no hay atributos ni en el form, ni en los eventos fuera de For Each, ni a nivel del grid, en las opciones "Hide" y "Order". Es decir, solo aparecen variables en esos lugares.

En este caso se genera sólo UN evento Load (en lugar de uno por cada línea a cargar en el

[7] Como veremos un poco más adelante, cuando mencionemos los work panels sin tabla base, no siempre existe un for each implícito.

grid) y es en éste donde se deben cargar los datos en el grid (utilizando el **comando Load**). En este caso diremos que se trata de un work panel sin tabla base.

Por ejemplo, supongamos que queremos implementar un work panel donde se muestren todos los proveedores y por cada uno el total de los pedidos efectuados al mismo. Una forma de hacerlo es utilizando variables para cargar los datos en el grid:

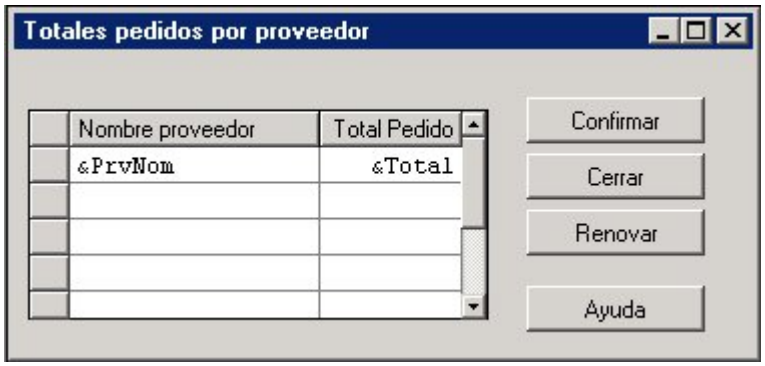


Fig. 68: Form de work panel "Totales por proveedor"

Y programamos el evento Load:

```
Event Load
   For each
      &PrvNom = PrvNom
      &Total = 0
      for each
         &total += PedTot
      endfor
      Load
   endfor
endevent
```

Ocurrirá por tanto el evento Refresh, e inmediatamente después el evento Load, una sola vez. La carga de cada línea en el grid se realiza con el comando Load y la iteración se realiza con el For Each programado por el analista.

Dependiendo de la plataforma de implementación, se mantiene o no la facilidad de carga "bajo pedido" para este caso en el que no hay tabla base. En .NET y Java se mantiene. Sin embargo en Visual Basic y Visual FoxPro, antes de mostrar los datos, se carga toda la información del grid.

Condiciones

En esta sección se definen las restricciones sobre los datos a recuperar de la misma forma que en los reportes.

Lo interesante de las condiciones en los work panels, es que en las mismas se pueden utilizar variables que se encuentran en la pantalla, de tal manera que el usuario, cambiando los valores de estas variables cambia los datos que se muestran en el grid sin tener que salir de la pantalla.

Por ejemplo, si queremos visualizar sólo un rango de proveedores, agregamos en el form las variables, *&PrvIni* y *&PrvFin*:

Fig. 69: Form de work panel para trabajar con los proveedores del rango seleccionado por el usuario

Y definiendo las condiciones:

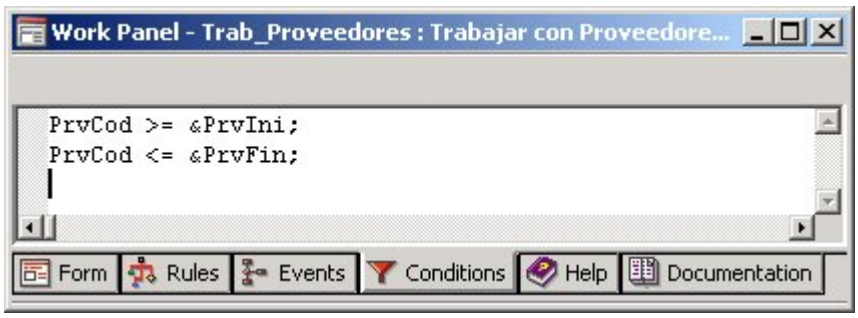


Fig. 70: Condiciones para filtrar los proveedores que se muestran en el grid

se obtiene el efecto deseado.

Observemos que cada vez que el usuario modifique una de estas dos variables que aparecen en las condiciones, deberá dispararse el evento Refresh para volver a cargar el grid. Esto es realizado automáticamente.

Todas las consideraciones sobre optimización de condiciones vistas en el capítulo de reportes son igualmente válidas aquí.

Orden de los datos en el grid

Cuando en un reporte queríamos ordenar un For Each por determinado conjunto de atributos, utilizábamos la cláusula Order. En el caso de un work panel en el que tenemos un For Each implícito, ¿dónde podemos decir que queremos recuperar los datos en el grid con determinado orden? Por ejemplo, si queremos ver los proveedores por orden alfabético, ¿cómo especificamos este orden?

Esto se logra haciendo botón derecho sobre el grid y seleccionando la opción “Order” que aparece en el menú que se abre:

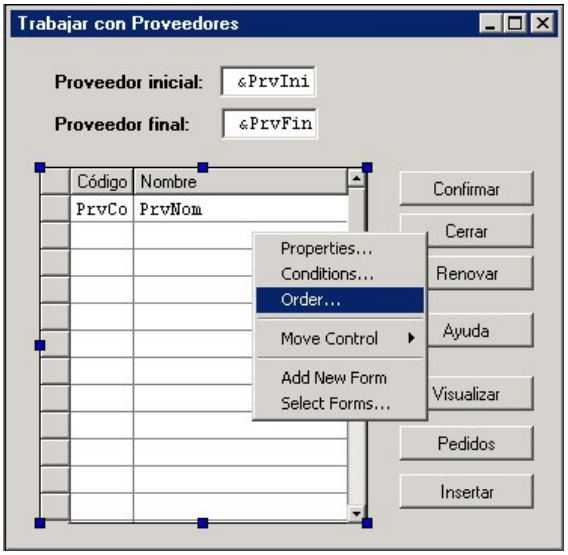


Fig. 71: Orden de los datos en el grid

Se abrirá una pantalla que permite insertar los atributos por los que se quiere ordenar el grid (en nuestro caso elegiremos *PrvNom*)

Es equivalente a la cláusula ORDER del FOR EACH y se aplica todo lo dicho sobre el tema en el capítulo sobre el objeto reporte.

Reglas

Valen la mayoría de las reglas de los reportes, más algunas particulares:

Noaccept

A diferencia de las transacciones, en los work panels ningún atributo es aceptado (siempre se muestra su valor pero no se permite modificarlo). Para variables se sigue el siguiente criterio:

- todas las variables presentes en la pantalla son aceptadas, a no ser que tengan la regla NOACCEPT.

Por ejemplo, en un work panel en el que deseemos mostrar la fecha en el form, teniendo la fecha cargada en la variable *&Fecha*, escribimos la siguiente regla, para que el valor del campo no se permita modificar:

noaccept(*&Fecha*);

Search

Cuando el grid tiene muchas líneas cargadas, a menudo se quiere brindar la posibilidad de que el usuario pueda “posicionarse” en alguna línea determinada del mismo en forma directa, sin tener que avanzar página por página haciendo scroll. Por ejemplo, si en nuestro work panel de proveedores queremos que exista la posibilidad de buscar un proveedor según su nombre, se debe definir la regla:

search(*PrvNom* >= *&Nom*);

y en la pantalla se debe agregar la variable *&Nom*:

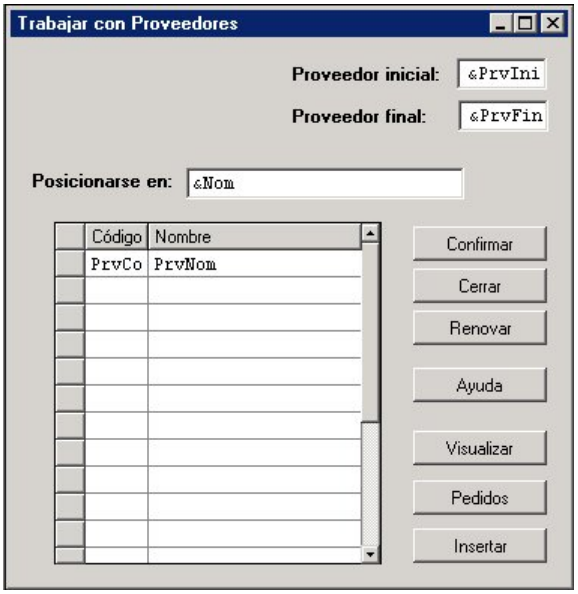


Fig. 72: Form de "Trabajar con Proveedores" con posibilidad de posicionarse en una línea dada

De esta manera, cada vez que el usuario digite algo en *&Nom*, luego de dar Enter el cursor quedará posicionado en el primer proveedor con *PrvNom* mayor o igual al digitado. En los casos de nombres, es muy usual utilizar el operador *like*:

search(*PrvNom* LIKE *&Nom*);

en donde el cursor se posiciona en el primer proveedor cuyo *PrvNom* contenga a *&Nom* (en el ejemplo si en *&Nom* se digitó 'X' se posicionará en 'GX Corp.').

Si bien esta regla es muy útil para que el usuario avance rápidamente en un grid, se debe tener en cuenta que no hay ningún tipo de optimización sobre la misma. Si se quieren utilizar los criterios de optimización se deben usar las Condiciones.

Bitmaps

Los bitmaps son utilizados usualmente para mejorar la presentación de las aplicaciones. Se pueden incorporar en los Forms de las transacciones y work panels, así como en el Layout de reportes y procedimientos. GeneXus provee dos métodos diferentes para agregar un bitmap:

Fixed Bitmaps

Se puede agregar un control bitmap seleccionando el ítem correspondiente de la paleta de herramientas "Controls". Al insertar un control bitmap se abre un diálogo para configurar sus propiedades. La más importante es la ubicación, *FileName*, del bitmap.

Agreguemos un logo en el work panel que permite visualizar los datos de un proveedor. El nuevo form se verá de la siguiente forma:

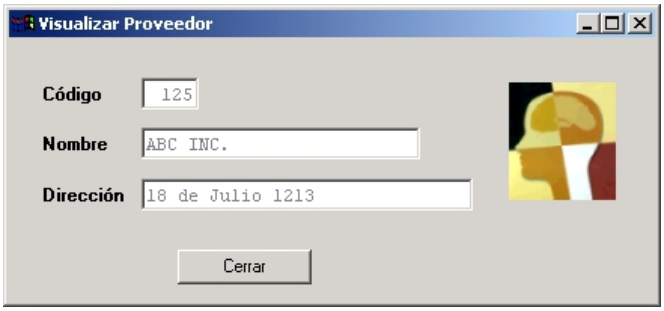


Fig. 73: Work panel "Visualizar Proveedor" con bitmap fijo

En este caso siempre aparecerá el mismo bitmap en el Form, independientemente del proveedor.

Dynamic Bitmaps

Este método tiene como diferencia que el bitmap lo asignamos a una variable y usando la función **Loadbitmap** podemos cargar el bitmap que deseemos. Por ejemplo, se puede tener un bitmap que despliegue la foto de cada proveedor.

Definición de una varible de tipo bitmap:

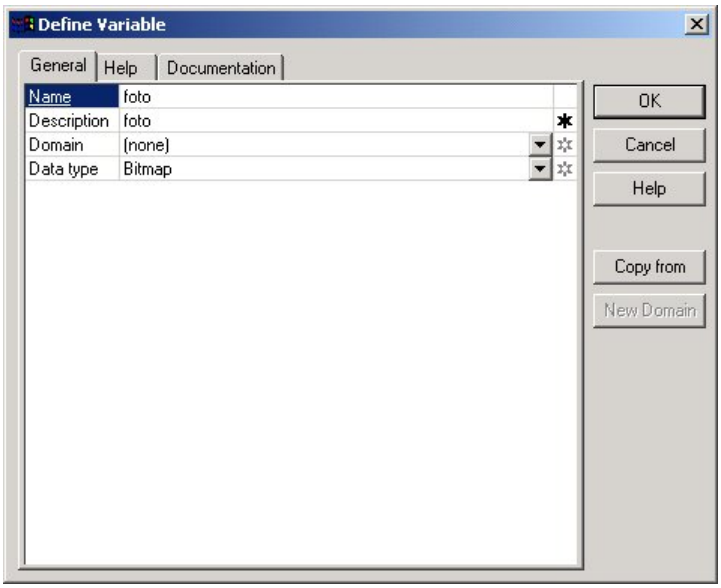


Fig. 74: Definición de variable de tipo Bitmap

En el evento load hay que cargar el bitmap para el proveedor pasado por parámetro.

Para poder hacer esto, debemos agregar un atributo, *PrvFoto*, en la transacción de proveedores, que contendrá la ubicación del archivo (bmp, gif, jpg, etc.) con la foto a cargar.

Fig. 75: Evento Load para cargar bitmap dinámico

En tiempo de ejecución se verá de la siguiente forma:

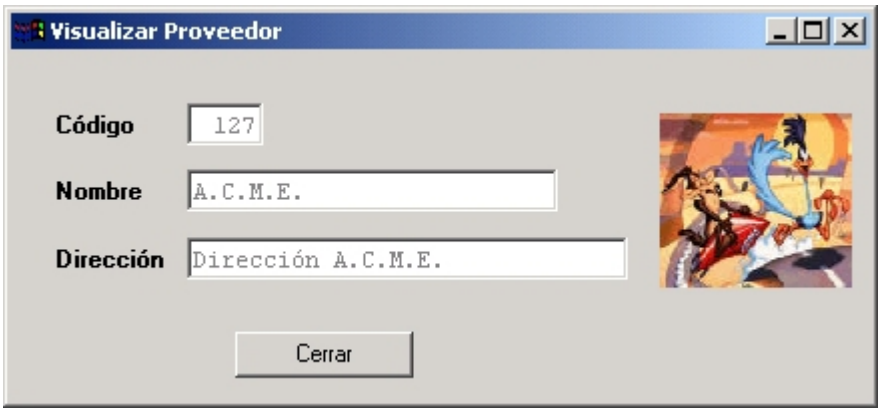


Fig. 76: Visualizar proveedor en ejecución con bitmap dinámico

Propiedades

Como para cualquier otro objeto, se pueden configurar las propiedades de un work panel, accediendo a la siguiente pantalla:

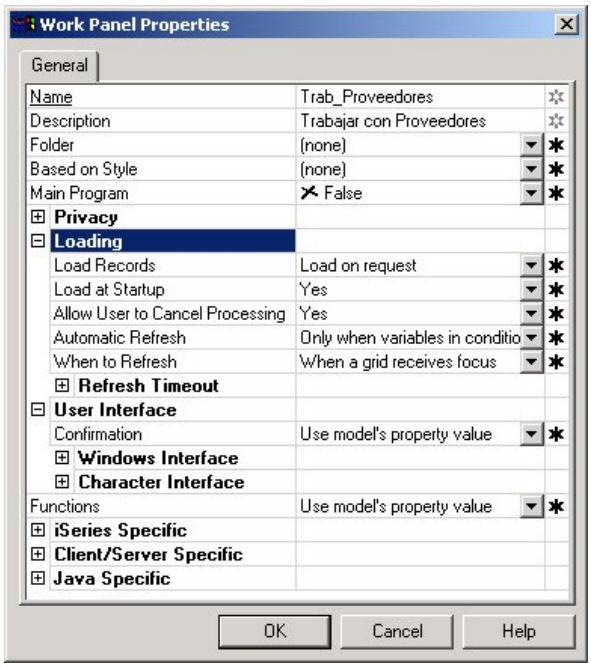


Fig. 77: Diálogo de Propiedades del work panel "Trabajar con Proveedores"

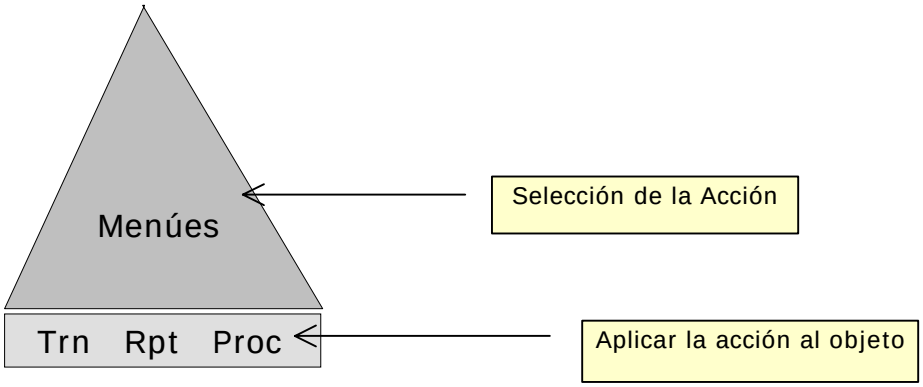
Las propiedades aparecen agrupadas por categorías. Por ejemplo, bajo la categoría “Loading” aparecen las propiedades que tienen que ver con la carga de los datos del work panel. Entre ellas aparece la propiedad “Load Records” que permite especificar si se quieren cargar los datos en el grid a medida que se va haciendo scroll (paginando), o si se quieren cargar todos al comienzo. Si la cantidad de datos a cargar es muy grande, entonces el valor de esta propiedad producirá una diferencia significativa en el tiempo de respuesta.
Si bien cada grid tiene la posibilidad de definir estas mismas propiedades de carga, si un work panel tiene múltiples grids, y en todos queremos especificar las mismas propiedades, la forma más simple es hacerlo a nivel del objeto, en lugar de tener que hacerlo grid por grid.

Existen otras propiedades que tienen que ver con la interfaz y son las que se presentan bajo el grupo “User Interface”.

Diálogos Objeto-Acción

Con el uso de work panels, se tiene la oportunidad de definir para la aplicación una arquitectura orientada a diálogos Objeto-Acción. Pero previo a introducir este concepto veamos cuál era la forma tradicional.

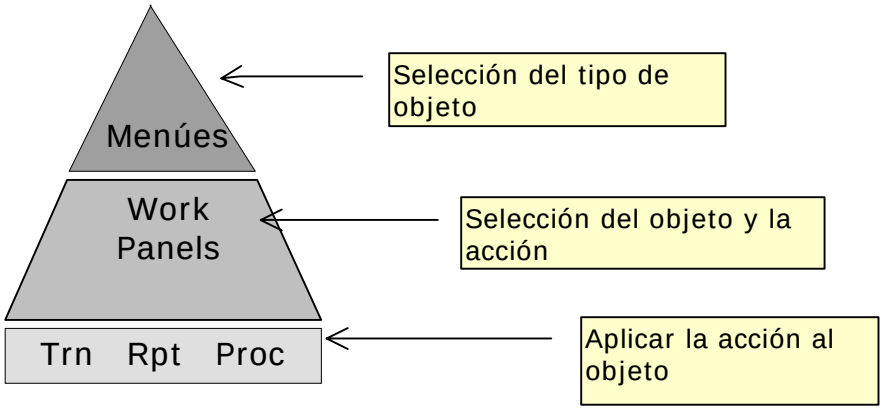
Si observamos una aplicación desarrollada por los métodos tradicionales, veremos que cuanto más grande es, más grande será el árbol de menúes que se tiene para acceder a los programas que efectivamente trabajan con los datos. La arquitectura de este tipo de sistemas normalmente es:



Esta situación se puede apreciar claramente en los sistemas con muchas consultas. En general, todas estas consultas dependen de un menú, pero no se encadena una consulta con las otras. Al no estar encadenadas, se da la paradoja de que cuanto más consultas se implementan en el sistema, menos son usadas por el usuario, pues le resulta difícil acordarse de las diferencias entre ellas.

Diremos que este tipo de aplicación tiene un diálogo Acción-Objeto, porque primero se elige la acción a realizar (por ejemplo modificar un proveedor, o listar los pedidos) y luego se elige el objeto al cuál aplicar la acción (qué proveedor o los pedidos de qué fecha).

Con work panels se pueden implementar sistemas en donde el diálogo puede ser Objeto-Acción: primero seleccionamos el objeto con el cuál vamos a trabajar y luego las acciones que aplicamos al mismo. En este caso la arquitectura será:



De esta manera, una vez que el usuario selecciona el tipo de objeto con el que va a trabajar, ya puede utilizar el work panel que le permite seleccionar el objeto en sí, y luego, las acciones a aplicar sobre el mismo.

Esta arquitectura permite desarrollar aplicaciones complejas, pero que se mantienen fáciles de usar.

Browser de GeneXus

Este navegador permite, entre otras opciones, desplegar todos los objetos del modelo que se invocan a través de los comandos call y udp, así como los prompts y themes.

Dado un programa, podemos ver su árbol de llamadas (a qué programas llama) y desde qué programas es llamado, eligiendo la opción “Call tree” o “Callers tree” del diálogo del browser.

En nuestro ejemplo vamos a ver el árbol de llamadas del work panel de Trabajar con Proveedores. Para ello elegimos la opción Tools/Browsers del menú de GeneXus y seleccionamos mediante la opción Pattern el objeto del que deseamos ver la información:

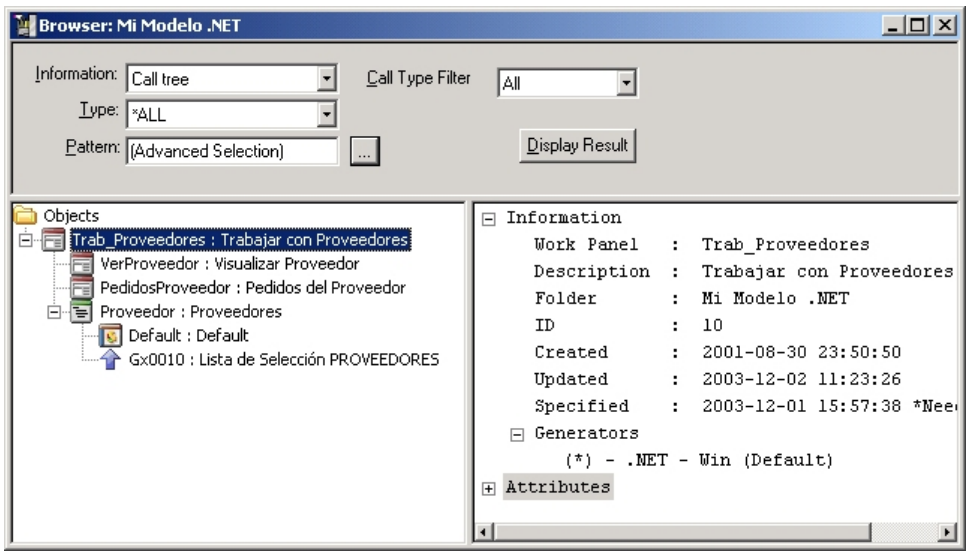


Fig. 78: Browser “Call tree”

Desde este browser podemos editar cualquier objeto que aparezca en las listas.

7. Objetos Web

Los Objetos Web se utilizan para desarrollar aplicaciones para Internet. Generan código HTML -el código que los navegadores interpretan- pudiendo interactuar con la base de datos, permitiéndonos de esta forma la generación de páginas web dinámicas además de la generación de páginas estáticas.

Los Objetos Web los podemos clasificar en: Transacciones con form Web (HTML) y en Web Panels. El ***desarrollo es inmediato***, ya que cualquier usuario GeneXus va a trabajar con los Objetos Web de la misma forma con la que lo hace con el resto de los objetos GeneXus, optimizando sus tiempos de desarrollo.

No entraremos en detalle en este tema en el presente libro, pero sí les daremos una breve descripción a continuación.

Transacciones

Las Transacciones Web no son un nuevo tipo de objeto GeneXus, sino un form más para las transacciones que permiten su ejecución en navegadores.

Para diseñar el form Web (HTML) de una transacción, se debe abrir la transacción y seleccionar la opción Object/Web Form del menú GeneXus o la solapa etiquetada como “Web Form”.

Las Transacciones Web facilitan el diseño de aplicaciones Web porque permiten resolver el ingreso de datos realizando automáticamente todos los controles de integridad referencial necesarios.
Para ejecucutarlas, sólo se requiere un navegador instalado en el cliente.

Web Panels

Se puede decir que los objetos web panel y work panel son similares ya que ambos permiten definir consultas interactivas a la base de datos. Son objetos muy flexibles que se prestan para múltiples usos, cuya programación está dirigida por eventos. De todos modos, hay algunas diferencias entre ellos, causadas principalmente por el esquema de trabajo en Internet.
Una particularidad fundamental que diferencia a estos dos objetos es que en los web panels, en tiempo de ejecución, el resultado de la consulta se formatea en HTML para presentarse en un navegador.

Anexo A. Modelos de Datos Relacionales

El propósito de este anexo es explicar una serie de conceptos sobre bases de datos relacionales relevantes para el uso de GeneXus.

Tablas

En una base de datos relacional la única forma de almacenar información es en tablas.
Una tabla es una matriz (tiene filas y columnas) con un nombre asociado. A las columnas les llamamos **atributos**, y a las filas **registros** o **tuplas**. Por ejemplo:

PROVEEDORES	***PrvCod***	***PrvNom***	***PrvDir***
	125	ABC Inc.	18 de Julio 1213
	126	GX Corp.	Br. Artigas 980

Una tabla se diferencia de un archivo convencional porque debe cumplir las siguientes reglas:

- Todos los atributos representan la misma información para cada una de las filas (o registros). Es decir en el atributo *PrvNom* se almacenan los nombres de los proveedores, y no puede ocurrir que para algunos proveedores en ese atributo se almacene la dirección del mismo. En la práctica esta regla implica que no existen tablas con diferentes tipos de registros.
- No existen dos registros con exactamente la misma información.
- El orden de los registros no contiene información. Es decir se pueden reordenar los registros sin perder información.

Clave primaria y Claves candidatas

Dado que en una tabla no pueden haber dos registros con la misma información, siempre se puede encontrar el atributo o conjunto de atributos cuyos valores no se duplican. Se llama a ese atributo o conjunto de atributos **clave primaria** (*PK*: *Primary Key*) de la tabla.

En realidad pueden existir varios de esos conjuntos; a cada uno de ellos lo llamamos **clave candidata**.
Por ejemplo, si sabemos que además de no haber dos proveedores con el mismo código, tampoco puede haber dos proveedores con el mismo nombre, entonces en la tabla PROVEEDORES tanto *PrvCod* como *PrvNom* serán claves candidatas.

En una base de datos relacional para toda tabla debe definirse una de las claves candidatas como clave primaria.

También es importante respetar la siguiente regla: no deben existir dos tablas con la misma clave primaria.

Por ejemplo consideremos el siguiente diseño:

PROVEEDORES	*PrvCod*	*PrvNom*

PROV_DIR	*PrvCod*	*PrvDir*

Tanto la tabla PROVEEDORES como PROV_DIR tienen la misma clave primaria[8]: el atributo *PrvCod*. Esto era relativamente común cuando el criterio de diseño era separar las actualizaciones. Por ejemplo dado que *PrvDir* cambia más que *PrvNom*, entonces era mejor definir un archivo diferente en donde el registro que se modificaba era menor, y por lo tanto la performance mejoraba.

Sin embargo en una Base de Datos Relacional el criterio de diseño es diferente (ver sección de Normalización) y es preferible una sola tabla con los tres atributos:

PROVEEDORES	*PrvCod*	*PrvNom*	*PrvDir*

Integridad Referencial

Son las reglas de consistencia entre los datos de las distintas tablas.
Supongamos que nuestro modelo de datos corresponde a la representación del *Sistema de Compras para una cadena de Farmacias*, introducido en el capítulo 1.
En este sistema queremos representar pedidos de artículos a proveedores, efectuados por los analistas de compras. Para ello necesitamos una tabla PEDIDOS, que contendrá información de los pedidos efectuados por los analistas de compras a los proveedores. Esta tabla podría contener los siguientes atributos: *PedNro*, *PedFch*, *PrvCod*, *AnlNro* y *PedTot* para representar el número de pedido, su fecha, el proveedor al cuál se le realiza el pedido, el analista de compras que efectúa el pedido y el importe total por concepto del pedido, respectivamente:

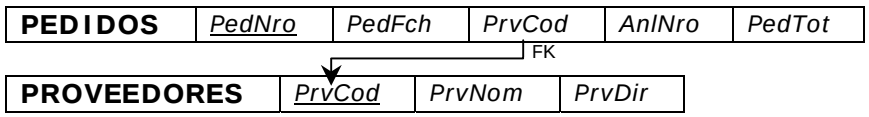
PEDIDOS	*PedNro*	*PedFch*	*PrvCod*	*AnlNro*	*PedTot*

Si observamos las dos tablas: PROVEEDORES definida antes y PEDIDOS, vemos que tienen un atributo en común: *PrvCod*. Queremos indicar con ello, que la información de ambas tablas está relacionada, ya que se trata del mismo *PrvCod* en ambas tablas.

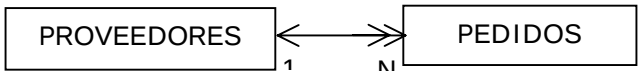
Utilizamos, pues, el concepto de **clave foránea** (*FK: Foreing Key*). Decimos que *PrvCod* es una clave foránea (FK) en PEDIDOS a la tabla PROVEEDORES (que tiene a *PrvCod* como clave primaria), con lo que queremos significar que el valor que se ingrese para un registro cualquiera de PEDIDOS en el atributo *PrvCod* (PEDIDOS.*PrvCod*) no puede ser cualquiera: tiene que cumplir que exista un registro en la tabla PROVEEDORES para el que:

PROVEEDORES.*PrvCod* = PEDIDOS.*PrvCod*

o dicho de otra manera: tiene que existir el proveedor.



La relación entre PROVEEDORES y PEDIDOS también puede representarse mediante el siguiente diagrama, que está inspirado en los conocidos Diagramas de Bachman:



En general, en los modelos de datos relacionales deben establecerse explícitamente las claves

[8] Para representar la clave primaria de una tabla, dibujamos una línea debajo de los atributos que la conforman.

foráneas de cada tabla. En GeneXus se hace implícitamente: la relación entre estas dos tablas se determina analizando los atributos que tienen en común. Aquí tenemos que el atributo común es *PrvCod*, que es la clave primaria de la tabla PROVEEDORES y se encuentra en la tabla PEDIDOS y, por lo tanto, ambas tablas están relacionadas y la relación es 1-N.

Con relación 1-N se indica que un Proveedor puede tener varios Pedidos asociados y que un Pedido sólo puede tener un Proveedor. También es usual decir que la tabla PROVEEDORES está **superordinada** a la tabla PEDIDOS, y PEDIDOS está **subordinada** a PROVEEDORES.

Esta relación implica que los datos de ambas tablas no son independientes y cada vez que se realicen modificaciones en una de las tablas, se deben tener en cuenta los datos de la otra tabla. A estos controles se les llama de **integridad referencial** y son:

- Cuando se elimina un registro en la tabla superordinada (PROVEEDORES), no deben existir registros asociados (que lo "referencien") en la tabla subordinada (PEDIDOS).
- Cuando se crean/modifican registros en la tabla subordinada (PEDIDOS), debe existir el registro correspondiente ("referenciado") en la tabla superordinada (PROVEEDORES).

Los Diagramas de Bachman tienen la virtud de explicitar las relaciones de integridad referencial del modelo de datos.

Índices

Son vías de acceso eficientes a las tablas. En GeneXus existen cuatro tipos de índices: *Primarios*, *Foráneos*, *de Usuario* y *Temporales*. A su vez los de *Usuario* se dividen en *"Unique"* o *"Duplicate"*.

El **índice primario** de una tabla es el que se define para la clave primaria y se utiliza para hacer eficiente el control de unicidad de los registros.

Por otro lado, la existencia del índice primario hace posible que se realice eficientemente el segundo control de integridad referencial enunciado más arriba. En el ejemplo visto, la existencia de un índice por *PrvCod* en la tabla PROVEEDORES, hace eficiente el chequeo de que cuando se ingresa/modifica un registro en la tabla PEDIDOS, exista uno en la tabla PROVEEDORES tal que el valor del atributo *PrvCod* del primer registro coincida con el valor del atributo de igual nombre en el segundo registro.

Con GeneXus todos los índices primarios son **definidos automáticamente** a partir de los *identificadores de las transacciones*.

Los **índices foráneos** son utilizados para hacer eficientes los controles de integridad inter-tablas. Se definen para las claves foráneas, por lo que son los que hacen posible que se realice eficientemente el primer control de integridad referencial mencionado anteriormente.

En el ejemplo, la existencia del índice foráneo por *PrvCod* en la tabla PEDIDOS, posibilita que eficientemente se chequee y, en base al chequeo se prohiba, eliminar un registro de la tabla PROVEEDORES si existe uno en la tabla PEDIDOS, para el cual PEDIDOS.*PrvCod* = PROVEEDORES.*PrvCod*. De esta manera se evita dejar referencias colgadas.

Con GeneXus todos los índices foráneos son **definidos automáticamente** a partir de las claves foráneas, que como mencionamos son identificadas por GeneXus basándose en el nombre de los atributos.

De no tener creados estos dos tipos de índices, todo el sistema de control de la integridad referencial, fundamental para asegurar la consistencia de los datos, sería terriblemente ineficiente. Esto es lo que justifica que GeneXus cree y mantenga en forma automática estos

índices.

Los **índices de usuario**, en cambio, no son creados automáticamente por GeneXus. Son creados a pedido del analista y se definen, fundamentalmente, para recuperar eficientemente los datos con determinado orden. Por ejemplo en la tabla PROVEEDORES se puede definir un índice por *PrvNom*, que es muy útil para los listados de proveedores ordenados por nombre.

La forma de definir *claves candidatas* para una tabla (distintas de la clave primaria) en GeneXus es a través de la creación de índices de usuario. Cuando se define una *clave candidata* en GeneXus, éste controla la unicidad de los valores de los atributos que la componen, así como lo hace para la clave primaria. Para poder realizar este control eficientemente, GeneXus deberá crear un índice, por lo que directamente, la forma de definir una *clave candidata*, será crear un *índice de usuario*, y especificar que los valores deben ser únicos. Por lo tanto, los *índices de usuario* permiten especificar que los valores serán "Unique" o "Duplicate".

Por último, los **índices temporales** son creados en algunas plataformas de implementación, cuando quiere recorrerse una tabla por un determinado orden de forma eficiente, pero no quiere crearse un índice de usuario para ello. En este caso, se crea un índice en forma temporal, se utiliza y luego se elimina. Los índices temporales también **son creados automáticamente** por GeneXus.

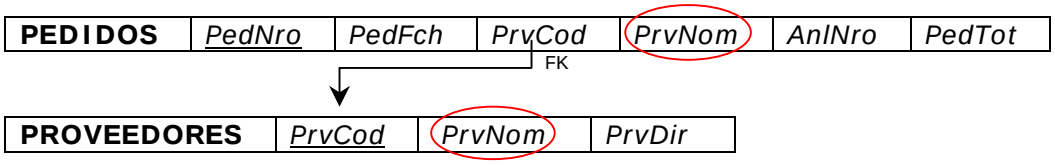
En una base de datos relacional los índices se utilizan sólo por problemas de performance, pero siempre es posible acceder a los datos de una tabla por cualquiera de los atributos de la misma.

Normalización

El proceso de normalización determina en qué tablas debe residir cada uno de los atributos de la base de datos.

El criterio que se sigue es básicamente determinar una estructura tal de la base de datos, que la posibilidad de inconsistencia en los datos sea mínima.

Por ejemplo, si tenemos que almacenar información sobre proveedores y pedidos, se podría tener la siguiente estructura:

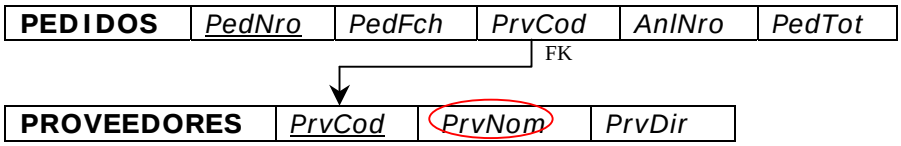


en donde vemos que ambas tablas tienen dos atributos en común (*PrvCod* y *PrvNom*). En el caso de *PrvCod* es necesario que se encuentre en ambas tablas dado que es el identificador de los proveedores y por lo tanto debe estar tanto en la tabla PROVEEDORES como clave primaria, como en la tabla PEDIDOS como clave foránea, para indicar a qué proveedor corresponde el pedido.

La situación es diferente para *PrvNom*. No es necesario que esté en la tabla PEDIDOS, porque si conocemos el código de proveedor, *PrvCod*, podemos determinar cuál es su nombre, *PrvNom*. Basta con buscar en la tabla PROVEEDORES por la clave primaria. Si de cualquier manera se almacena *PrvNom* en la tabla PEDIDOS tenemos que esta estructura tiene más posibilidades de inconsistencia que si no estuviera allí.
Por ejemplo si un proveedor cambia su *PrvNom*, el programador debe encargarse de tener un

programa que lea todos los pedidos de ese proveedor y le asigne el nuevo *PrvNom*.

Entonces la estructura correcta (diremos “normalizada”) será:



El proceso de normalización, que se acaba de introducir de una manera muy informal, se basa en el concepto de dependencia funcional, cuya definición es:

> Se dice que un atributo depende funcionalmente de otro si para cada valor del segundo sólo existe UN valor del

Por ejemplo *PrvNom* depende funcionalmente de *PrvCod* porque para cada valor de *PrvCod* sólo existe UN valor de *PrvNom*.

La definición es algo más amplia, ya que se puede definir que un conjunto de atributos dependa funcionalmente de otro conjunto de atributos.

GeneXus diseña la base de datos llevándola a **tercera forma normal** (admitiendo algunas excepciones).

Para un estudio más detallado de normalización, dirigirse a cualquier libro que trate sobre el modelo de datos relacional.

Tabla extendida

Como vimos en la sección anterior, el criterio de diseño de la base de datos se basa en minimizar la posibilidad de inconsistencia en los datos.

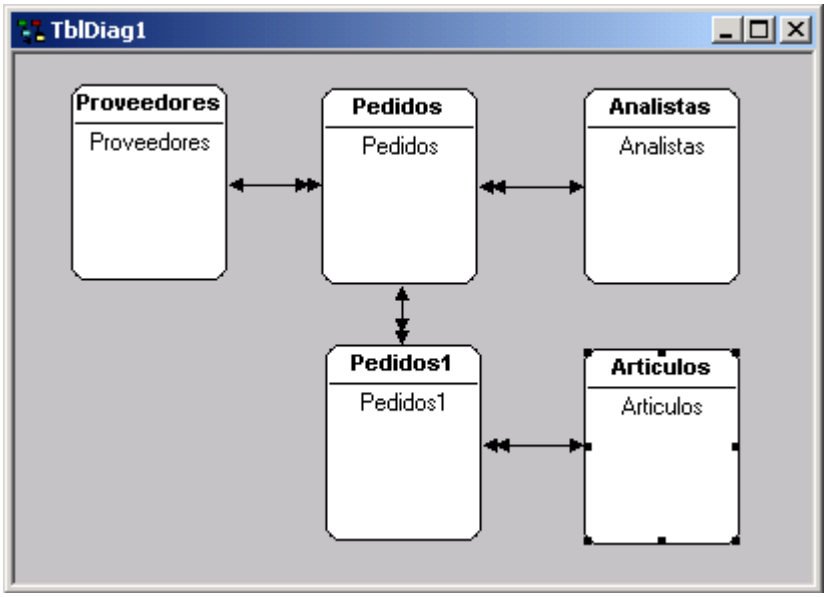
Una base de datos diseñada de esta manera tiene una serie de ventajas importantes (tal es así que actualmente la normalización de datos es un estándar de diseño), pero se deben tener en cuenta también algunos inconvenientes.

El inconveniente más notorio es que los datos se encuentran dispersos en muchas tablas, y por lo tanto cuando se quieren hacer consultas más o menos complejas a la base de datos se debe consultar una cantidad importante de tablas. Así, por ejemplo, para listar los pedidos por proveedor es necesario consultar las tablas PROVEEDORES y PEDIDOS.

Para simplificar esta tarea GeneXus utiliza el concepto de **tabla extendida**, cuya definición es:

> Dada una tabla de la base de datos, que llamaremos *tabla base*, se denomina ***tabla extendida*** de la misma al conjunto conformado por los atributos que:
> - pertenecen a la tabla.
> - pertenecen a toda tabla Y, tal que la relación entre la tabla extendida determinada hasta el

Si hacemos un diagrama de Bachman del modelo de datos correspondiente al sistema de compras para la cadena de farmacias, introcido en el capítulo 1, obtenemos:



donde además de las tablas de proveedores y pedidos, tenemos tres tablas más: ARTICULOS, para almacenar la información de los artículos, ANALISTA que almacena la información de los analistas de compras, y PEDIDOS1, que registra las líneas de pedido, es decir, los artículos que componen cada pedido, junto al precio por artículo y la cantidad pedida del mismo.

La tabla extendida de PEDIDOS comprende a todos los atributos de dicha tabla, más todos los atributos de PROVEEDORES y todos los de ANALISTAS, pero no los atributos de las líneas del pedido, PEDIDOS1, o de ARTICULOS (porque si bien están relacionados, su relación no es N-1).

En el siguiente diagrama se muestra, para cada una de las tablas del modelo, cuál es su tabla extendida:

Tabla base	**Tabla extendida**
PROVEEDORES	PROVEEDORES
ANALISTAS	ANALISTAS
PEDIDOS	PEDIDOS, PROVEEDORES, ANALISTAS
PEDIDOS1	PEDIDOS1, PEDIDOS, PROVEEDORES, ANALISTAS, ARTICULOS
ARTICULOS	ARTICULOS

El concepto de tabla extendida es muy utilizado en GeneXus, donde generalmente se trabaja con tablas extendidas y no con tablas físicas.

Por ejemplo, el comando FOR EACH que estudiaremos determina una tabla extendida y no una tabla física. Lo mismo ocurre en las transacciones, donde se pueden actualizar directamente mediante las reglas cualesquier atributos que pertenezcan a la tabla extendida correspondiente al nivel de la transacción en el que se esté trabajando.

Anexo B. Funciones, reglas y comandos

Algunas funciones, reglas y comandos disponibles se listan en las siguientes tablas (no se abarca aquí la totalidad de ellos):

FUNCIÓN	DESCRIPCIÓN
MANEJO DE FECHAS	
Day	Número de día de una fecha
Month	Número de mes de una fecha
Year	Año de una fecha
Today	Fecha de hoy
DoW	Número del día de la semana de una fecha (1..7)
CdoW	Nombre del día de la semana de una fecha
CMonth	Nombre del mes de una fecha
CtoD	Pasar de tipo carácter a tipo fecha
DtoC	Pasar de tipo fecha a carácter
YMDtoD	Armar fecha a partir de año, mes y día
Addmth	Sumar un número dado de meses a una fecha
Addyr	Sumar un número dado de años a una fecha
Age	Diferencia en años entre dos fechas
Eom	Fecha fin de mes de una fecha
MANEJO DE STRINGS	
Str	Pasar un numérico a string
Substr	Substring de un string
Concat	Concatenar dos strings
Space	Definir un strings con “n” blancos
Len	Largo de un string
Trim	Quita los caracteres en blanco de un string
Ltrim	Quita los caracteres en blanco al principio de un string
Rtrim	Quita los caracteres en blanco al final de un string
Upper	Convierte un string a mayúsculas
Lower	Convierte un string a minúsculas
Int	Parte entera de un numérico
Round	Redondear un valor numérico
Val	Pasar un string a numérico

OTRAS	
Old	Valor del atributo antes de ser modificado
Previous	Valor del atributo en la última inserción
Time	Hora
Systime	Hora del sistema
Sysdate	Fecha del sistema
Userid	Identificación del usuario
Usercls	Clase de usuario
Wrkst	Workstation
Ask	Para pedir valores por pantalla
Udf	Función definida por el usuario
Udp	Procedimiento definido por el usuario
Null(<Att|Var>)	Si un atributo tiene valor nulo o no
Nullvalue(<Att|Var>)	Devuelve el valor nulo de un atributo
LoadBitmap	Carga un bitmap
RGB (<R>,<G>,<B>)	Retorna el número que representa al color RGB determinado por los 3 parámetros
Color(<GXColor>)	Retorna el número que representa al color pasado como parámetro
Random()	Permite generar números aleatorios
RSeed	Indica la semilla para la función Random()

MOMENTOS DE DISPARO DE LAS REGLAS EN LAS TRANSACCIONES	
AfterInsert	Luego que se insertó el registro
AfterUpdate	Luego que se actualizó el registro
AfterDelete	Luego que se dio de baja el registro
AfterValidate	Luego que se confirmó el registro (todavía no se hizo la modificación del registro)
AfterComplete	Luego que termina un ciclo de la transacción (después del COMMIT)
After(<Att>)	Función que devuelve true después de pasar por el atributo especificado.
AfterLevel	Luego que se iteró el nivel en el que está el atributo mencionado en la cláusula Level que le sigue. En caso de no estar sucedido de cláusula Level, se trata del primer nivel.
Level <Att>	Nivel de una transacción
Modified()	Devuelve true si fue modificado algún atributo de un nivel
Insert← función booleana	Si la transacción está en modalidad insert
Update←función booleana	Si la transacción está en modalidad update
Delete← función booleana	Si la transacción está en modalidad delete

REGLAS	DESCRIPCIÓN	T	W	R	P
Default	Dar un valor por defecto a un atributo o variable				
Equal	Asigna un valor en modalidad insert y funciona como filtro en update y delete.				
Add	Sumar un valor a un atributo teniendo en cuenta la modalidad de la transacción				
Subtract	Restar un valor a un atributo teniendo en cuenta la modalidad de la transacción				
Serial	Numerar serialmente atributos				
Msg	Desplegar un mensaje				
Error	Dar un error				
Noaccept	No aceptar atributo o variable en la pantalla				
Call	Llamar a un programa				
Submit	Someter una llamada a un programa				
Parm	Para recibir los parámetros				
Nocheck	No realizar el control de integridad referencial				
Allownulls	Permitir ingresar valores nulos				
Refcall	Hacer un call cuando falla el control de integridad referencial				
Refmsg	Mensaje que se despliega cuando falla el control de integridad referencial				
Prompt	Asociar al prompt un objeto definido por nosotros				
Noprompt	Inhibe la lista de selección				
Accept	Aceptar atributo o variable en la pantalla				
Default_mode	Modalidad por defecto al ingresar en una transacción				
Color	Dar colores a los atributos, variables y fondo				
Search	Condición de búsqueda sobre el subfile				
Noread	Inhibe la lectura de atributos				
Printer	Para seleccionar el printer file en iSeries o el “printing form” en otros generadores				

T = Transacciones
W = Work Panels
R = Reportes
P = Procedimientos

Las casillas que aparecen marcadas indican que la función aparece disponible para ese objeto.

COMANDOS	DESCRIPCIÓN	T	W	F	P
Header	Comienzo del cabezal				
Footer	Comienzo del pie de página				
For each	Para recorrer los registros de una tabla				
For each line [IN]	Para recorrer las líneas de un subfile				
Exit	Permite salir de un while				
Do while	Repetición de una rutina mientras se cumpla una condición				
For to Step	Iterar cierta cantidad de veces a partir de un cierto valor hasta determinado valor con un incremento dado				
Do Case	Ejecuta algún bloque según qué condición se cumpla				
If	Ejecuta un bloque de comandos en caso que la condición evalúe verdadera				
Commit	Commit				
Rollback	Rollback				
Print if detail	Imprime si existen registros en el subfile.				
Return	Finaliza la ejecución y vuelve al programa llamador				
&<a> = <Exp>	Asignación de una variable				
Lineno	Número de línea donde los datos serán impresos				
Prncmd	Para pasar comandos especiales a la impresora				
Pl	Largo de página				
Mt	Margen superior				
Cp	Provoca un salto de página si quedan menos de “n” líneas por imprimir				
Noskip	No pasar a la siguiente línea de impresión				
Eject	Salto de página				
Call	Llamar a un programa				
Submit	Someter un programa				
Msg	Desplegar un mensaje				
Do	Llamar a una subrutina				
Sub - EndSub	Bloque de subrutina				
ATT = <Exp>	Actualización de atributo				
New - EndNew	Insertar registro				
Delete	Borrar un registro				

Indice de figuras

Fig. 1: Solapas que figuran en la parte inferior de la ventana de edición de una transacción.....13
Fig. 2: Estructura transacción “Proveedores” 14
Fig. 3: Form GUI de la transacción “Proveedores” 14
Fig. 4: Form Web de la transacción “Proveedores” 14
Fig. 5: Diálogo “Define Attribute”.....15
Fig. 6: Estructura de “Pedidos” en GeneXus 23
Fig. 7: Form GUI - transacción “Proveedores”.....27
Fig. 8: Form GUI por defecto de la transacción “Pedidos” 28
Fig. 9: “Lista de Selección PROVEEDORES” 29
Fig. 10: Form transacción “Artículos” 30
Fig. 11: Form personalizado de transacción “Pedidos”.....32
Fig. 12: Paleta de herramientas “Controls”.....33
Fig. 13: Paleta de herramientas “Form Editor” 33
Fig. 14: Propiedades control atributo/variable 34
Fig. 15: Propiedades control texto 35
Fig. 16: Propiedades control rectángulo.....36
Fig. 17: Diálogo de selección de propiedades de controles.....37
Fig. 18: Diagrama de Bachman de tablas PEDIDOS y PEDIDOS1.....39
Fig. 19: Browser: Sistema de Compras.....39
Fig. 20: Fórmulas en la estructura de transacción “Pedidos” 41
Fig. 21: Diálogo de propiedades de una transacción 46
Fig. 22: Solapas que figuran en la parte inferior de la ventana de edición de un reporte 47
Fig. 23: Listado de Proveedores en ejecución 48
Fig. 24: Layout del “Listado de Proveedores”.....48
Fig. 25: Propiedades del control Print Block.....49
Fig. 26: Paleta de herramienta “Controls” en reportes 49
Fig. 27: Source del “Listado de Proveedores” 50
Fig. 28: “Listado de Pedidos” en ejecución 51
Fig. 29: Layout “Listado de Pedidos” 51
Fig. 30: Diagrama de Bachman del modelo 52
Fig. 31: Listado de Navegación del reporte “Listado de Pedidos” 52
Fig. 32: Fragmento del listado de navegación reporte “Listado de Pedidos”.....53
Fig. 33: Listado de navegación del reporte “Listado de Pedidos” ordenado por fecha 54
Fig. 34: Fragmento listado de navegación de for each optimizado 56
Fig. 35: Fragmento listado de navegación de for each NO optimizado 56
Fig. 36: Diagrama de Bachman 58
Fig. 37: Listado de Pedidos por Proveedor.....59
Fig. 38: Layout de “Listado de Pedidos por Proveedor”.....59
Fig. 39: Layout Listado de Proveedores y Analistas 61
Fig. 40: Listado de Pedidos por Fecha 61
Fig. 41: Layout del “Listado de Pedidos por Fecha”.....61
Fig. 42: Source del “Listado de Proveedores y de Analistas” 63
Fig. 43: Layout del “Listado de Proveedores y de Analistas”.....63
Fig. 44: Diálogo de definición de variable.....64
Fig. 45: Listado de Pedidos por Fecha en ejecución 65
Fig. 46: Layout del reporte “Listado de Pedidos por Fecha”.....65
Fig. 47: Sección “Conditions” del reporte “Listado de Proveedores”.....68
Fig. 48: Pantalla de ingreso de rango de proveedores para el listado 68
Fig. 49: Paso 1 del Report Wizard.....70
Fig. 50: Paso 2 del Report Wizard.....71
Fig. 51: Paso 3 del Report Wizard.....72
Fig. 52: Paso 4 del Report Wizard.....73

Fig. 53: Layout del reporte generado con el Report Wizard....73
Fig. 54: Source del reporte generado con el Report Wizard 74
Fig. 55: Diálogo de Propiedades de un reporte....74
Fig. 56: Query “Detalle de Facturas por Cliente” 76
Fig. 57: Resultado de la consulta como Pívot Table en Excel....76
Fig. 58: Listado de Navegación del procedimiento de actualización de Artículos 79
Fig. 59: Solapas que figuran en la parte inferior de la ventana de edición de un work panel 81
Fig. 60: Form de “Trabajar con Proveedores” en ejecución 82
Fig. 61: Visualizar datos de un proveedor 82
Fig. 62: Visualizar pedidos de un proveedor 83
Fig. 63: Insertar un nuevo proveedor....83
Fig. 64: Form del work panel “Trabajar con Proveedores”....84
Fig. 65: Paleta de Herramientas “Controls” para un work panel....84
Fig. 66: Diálogo de Propiedades del control grid 85
Fig. 67: Evento ‘Insertar’ en work panel “Trabajar con Proveedores”....88
Fig. 68: Form de work panel “Totales por proveedor” 90
Fig. 69: Form de work panel para trabajar con los proveedores del rango seleccionado por el usuario 91
Fig. 70: Condiciones para filtrar los proveedores que se muestran en el grid....91
Fig. 71: Orden de los datos en el grid....92
Fig. 72: Form de “Trabajar con Proveedores” con posibilidad de posicionarse en una línea dada 93
Fig. 73: Work panel “Visualizar Proveedor” con bitmap fijo....94
Fig. 74: Definición de variable de tipo Bitmap....94
Fig. 75: Evento Load para cargar bitmap dinámico....95
Fig. 76: Visualizar proveedor en ejecución con bitmap dinámico....95
Fig. 77: Diálogo de Propiedades del work panel “Trabajar con Proveedores” 95
Fig. 78: Browser “Call tree” 98

NOTAS