



# **PowerBuilder**

## **Manual del programador**

Waldo Gómez Alvarez

Viña del Mar, Julio de 2000

Para Yasna, Fernando, Catalina y Esteban

# **Introducción a PowerBuilder**

# Introducción a PowerBuilder

PowerBuilder es un medio ambiente gráfico de desarrollo de aplicaciones cliente/servidor. Usando PowerBuilder, se puede desarrollar fácilmente poderosas aplicaciones gráficas orientadas al objeto, que accedan bases de datos locales o en el servidor, haciendo pleno uso de una atractiva interfaz de usuario (GUI). Powerbuilder proporciona las herramientas necesarias para crear todo tipo de aplicaciones .

## ¿ Qué es una aplicación GUI ?

Todas las aplicaciones GUI (graphical user interface) lucen de un modo similar. Esto significa que la apariencia, características y funcionamiento de los diferentes objetos que forman una aplicación, por ej: menús, ventanas, cajas de diálogo y botones están estandarizadas. De esta manera, la aplicación se provee de una consistencia visual que la hace atractiva y fácil de usar.

## ¿ Qué es una aplicación PowerBuilder ?

La interfáz de usuario de una aplicación PowerBuilder consiste de menús y ventanas con las cuales interactúa el usuario. Las aplicaciones PowerBuilder pueden incluir todos los controles windows standard, tales como: buttons, checkboxes, dropdown listboxes y edit boxes, así como controles especiales PowerBuilder que permiten que las aplicaciones sean fáciles de desarrollar y usar.

## Las aplicaciones PowerBuilder son manejadas por eventos (event-driven)

En una aplicación, el usuario controla qué pasa, a través de las acciones que toma. Por ejemplo, cuando un usuario le da click a un button, selecciona un ítem de menú, o ingresa datos en un edit box, se gatillan uno o más eventos. El usuario digita scripts que especifican el proceso que debe ejecutarse cuando los eventos se gatillan.

Por ejemplo, uno de los eventos de los botones se llama 'Clicked'. El usuario escribe un script para el evento Clicked, el cual especifica que sucederá cuando le de un click a ese botón. De modo similar, las edit boxes o cajas de edición, tienen un evento llamado 'Modified', el cual se gatilla cada vez que el usuario cambia un valor en la caja.

## Lenguaje PowerScript

Los scripts (códigos) se escriben usando PowerScript, el lenguaje de PowerBuilder. Los scripts consisten de comandos y sentencias PowerBuilder que ejecutan algún tipo de procesamiento en respuesta a un evento.

Por ejemplo, el script para el evento Clicked de un button podría recuperar y mostrar información de la base de datos; el script para el evento Modified de un edit box, podría evaluar los datos que se le ingresan y ejecutar procesamiento basado en estos datos.

La ejecución de un script de un evento, podría también causar que se gatillen otros eventos. Por ejemplo, el script para el evento Clicked en un button podría abrir una ventana, lo cual gatillaría a su vez el evento Open de esa ventana.

## Funciones PowerScript

PowerScript proporciona gran cantidad de funciones pre-construidas que se pueden usar para construir las variadas componentes de una aplicación. Por ejemplo, existe una función para abrir una ventana, una función para cerrar una ventana, una función para habilitar un button, una función para recuperar o traer (retrieve) datos, una función para actualizar la base de datos, etc.

Además, el usuario puede construir sus propias funciones para definir sus propios procesamientos. También puede acceder funciones externas presentes en archivos .DLL construidas con otros lenguajes.

## Programación orientada al objeto con PowerBuilder

Los bloques básicos de construcción de una aplicación PowerBuilder son los **objetos** que el usuario va creando. Cada objeto contiene las características particulares y comportamientos ( propiedades, eventos y funciones ) que son apropiadas a él. Aprovechando las técnicas de la programación orientada a objetos, tales como encapsulación, herencia y polimorfismo, se obtiene el máximo de cada objeto que se crea, haciendo el trabajo más reusable, extensible y poderoso.

La programación orientada al objeto está basada en tres principios fundamentales:

- Herencia
- Encapsulación
- Polimorfismo

En PowerBuilder se construyen ventanas, menús y objetos de usuario para definir objetos ancestros, los cuales tienen atributos, eventos, scripts, estructuras y funciones **encapsuladas**. A partir de ellos, se pueden heredar objetos para crear objetos descendientes. Un objeto puede **heredar** todas las características dadas a un objeto origen (código, otros objetos contenidos dentro de éste, etc), pero con la libertad de deshacerse de estos o añadir nuevos sin alterar el objeto original.

Los eventos que pueden gatillar scripts en el ancestro, pueden gatillar scripts en los descendientes. Esto se llama **polimorfismo**. Así como en cualquier objeto, los scripts determinan el proceso que se ejecuta cuando un evento ocurre; en los descendientes, el evento puede gatillar el mismo script que se gatilló en el ancestro o un script que sobrescribe o extiende el script del ancestro.

La programación orientada al objeto, permite trabajar en forma modular, extensible, flexible y con total o parcial reusabilidad de código.

## Atributos, eventos y métodos

### Atributos

Un objeto está constituido por ‘Atributos’. Estos lo caracterizan. Existen atributos que pueden modificarse y otros no. También existen atributos que son modificables tanto durante el diseño de la aplicación como durante la ejecución, y otros que sólo pueden ser modificados durante el diseño.

### Eventos

Cada acción que se puede realizar o que le puede ocurrir a un objeto, es un evento. Por ejemplo, hacer click sobre un objeto botón, presionar una tecla al escribir sobre un SingleLineEdit. Cada una de estas acciones es independiente una de otra, pero no necesariamente excluyentes. Por ejemplo, al cerrar una ventana, dando click al botón en el extremo superior derecho de la ventana, se realizan los eventos ‘close’ de la ventana y el evento ‘destroy’ de la misma. El primero se realiza al sacar la ventana de la pantalla y el segundo cuando se saca la ventana de la memoria.

### Métodos

Los ‘métodos’, son funciones destinadas a manipular elementos que son definidos en conjunto con el objeto. Esto implica que un objeto sólo puede manipular elementos que estén contenidos en él. Por ejemplo, la función *buscar* aplicada a un listBox, sólo puede buscar ítems en esa lista.

Este principio se denomina **encapsulamiento**, y permite una autonomía de cada objeto con su entorno.

## Desarrollo multiplataforma

PowerBuilder soporta desarrollo y distribución multiplataforma. Por ejemplo, se puede desarrollar una aplicación usando PowerBuilder bajo Windows, y distribuir la misma aplicación sin cambios en un Macintosh, o viceversa. Se puede tener un equipo de desarrolladores multiplataforma, algunos usando Windows y otros usando Macintosh, desarrollando la misma aplicación al mismo tiempo, pudiendo compartir libremente objetos PowerBuilder usados en la aplicación, debido a que los objetos son los mismos a través de las diferentes plataformas que soportan PowerBuilder.

## Conectividad con bases de datos

PowerBuilder proporciona fácil acceso para información corporativa almacenada en una amplia variedad de bases de datos. Usando PowerBuilder, se puede acceder a las bases de datos de las siguientes formas:

- **Usando la interfaz ODBC de Powersoft**

La interfaz ODBC Powersoft permite acceder a las bases de datos usando el standard Windows ODBC (Open Database Connectivity) para la conectividad de bases de datos. Cuando se usa la interfaz ODBC, se debe definir una fuente de datos, la cual consiste de los datos que se desea acceder y su DBMS (Data Base Management System), en el fondo la base de datos, sistema operativo y, si está presente el software de red que acceda a la DBMS. Las fuentes de datos almacenan la información necesaria para que la aplicación se conecte y accese exitosamente a la base de datos.

Las fuentes de datos ODBC pueden estar residentes localmente en el computador del usuario, o bien en un servidor de red. Por ejemplo, se puede acceder a una base de datos Sybase SQL Anywhere creada en un servidor remoto, instalando el driver ODBC para SQL Anywhere y definiendo la fuente de datos ODBC.

- **Usando una de las interfaces para bases de datos Powersoft que proveen una conexión directa a la base de datos.**

Una interfaz de base de datos Powersoft es una conexión nativa (directa) a una base de datos. PowerBuilder no va a través de ODBC para acceder a una base de datos cuando usa una interfaz de base de datos Powersoft. Por ejemplo, si se tiene instalado el software apropiado de SQL Server, se puede acceder a una base de datos SQL Server a través de la interfaz SQL Server de Powersoft.

Cada interfaz de base de datos Powersoft tiene su propia interfaz DLL que comunica con la base de datos específica. Cuando se usa una interfaz de base de datos Powersoft, la interfaz DLL se conecta a la base de datos a través de la API (application programming interface) proporcionada por el fabricante de la DBMS.



## ¿ Qué hace único a PowerBuilder ?

PowerBuilder fija el standard para las herramientas de desarrollo de aplicaciones en ambiente Cliente / Servidor.

El elemento PowerBuilder que marca la diferencia con otras herramientas similares, es el **DataWindow**. Esta es una ventana de datos que se usa para la recuperación, manipulación y muestra de información.

Sus ventajas principales son:

- Su construcción requiere poco o nada de conocimiento de SQL. Para construir un DataWindow Object, se especifica gráficamente la información que se desea recuperar desde la base de datos, seleccionando ítems en un ‘pintor’ de DataWindow. La sentencia SQL se genera automáticamente.
- Los DataWindows reducen el número de recursos de sistema requeridos para representar datos en forma de tablas o reportes.
- Los DataWindows poseen una amplia gama de características de reporte, incluyendo campos calculados (computed columns), gráficos, reportes anidados y compuestos.

# **Objetos PowerBuilder**

---

# Objetos PowerBuilder

## Objeto aplicación



El objeto aplicación es el punto de entrada a una aplicación. Es un objeto discreto que se graba en una librería PowerBuilder, tal como una ventana, menú, función, o un objeto datawindow object.

El objeto aplicación define el comportamiento a nivel de aplicación, tal como cuáles librerías contienen los objetos que usa la aplicación, qué fonts se usan por defecto para texto, y qué procesamiento debiera ocurrir cuando la aplicación comienza o termina.

Cuando un usuario corre la aplicación, se gatilla el evento *Open* del objeto aplicación. El script escrito para ese evento inicia la actividad en la aplicación. Cuando el usuario sale de la aplicación, se gatilla el evento *Close* del objeto aplicación. El script que se escribe generalmente para este evento, hace tareas de limpieza, tal como cerrar la conexión a la base de datos o escribir preferencias a un archivo. Si se producen errores graves durante la ejecución, se gatilla el evento *SystemError* del objeto aplicación.

## Ventana



Las ventanas son la interfáz entre el usuario y una aplicación PowerBuilder. Se usan para desplegar información, requerir información del usuario, y responder a acciones del mouse o el teclado.

Una ventana consiste de:

- **Propiedades**, que definen la apariencia de la ventana y su comportamiento. Por ejemplo, una ventana tiene una barra de título o está minimizada.
- **Eventos**, que son gatillados en respuesta a acciones del usuario.
- **Controles** dispuestos en la ventana.

## Datawindow



Un objeto DataWindow es un objeto que se usa para recuperar (retrieve), presentar, y manipular datos desde una base de datos relacional u otra fuente de datos, tal como una planilla Excel o un archivo dBASE.

Existen varios estilos de presentación para los DataWindows. Por ejemplo, se puede elegir mostrar datos en un formato fila/columna o en un formato freeform.

Se puede especificar formatos y otros atributos de dato para que la presentación sea lo más significativa para el usuario. Por ejemplo, si una columna puede tomar sólo un número pequeño de valores, se puede optar por darle la apariencia de radio buttons en el DataWindow para que el usuario elija solo de los valores presentados.

Se puede formatear el display o muestra del dato. Por ejemplo, mostrar teléfonos, números, salarios, y fechas en formatos apropiados al tipo de dato.

Si una columna puede tomar números sólo en un rango específico, se puede especificar una regla de validación simple para el dato, de tal forma de no tener que validar el dato entrado.

Existen muchas otras maneras en que se puede mejorar la presentación y manipulación de los datos en un objeto DataWindow. Por ejemplo, incluyendo campos calculados, pictures, y gráficos los cuales se vinculan directamente a los datos recuperados por el objeto DataWindow.

## Menús



Los menús son listas de comandos u opciones (ítems de menú) que un usuario puede seleccionar en la ventana activa. Los ítems de menú tienen scripts escritos por el usuario. La mayoría de las ventanas en una aplicación PowerBuilder tienen menús asociados. Cada elección en un menú se define como un objeto menú en PowerBuilder.

Los menús que se definen en PowerBuilder, trabajan exactamente como los menús standard en el medio ambiente operativo. Por ejemplo, se puede seleccionar los ítems de menú con el mouse o con el teclado, y se definen teclas rápidas de acceso a los ítems. Los ítems de menú se muestran en una barra de menú, en un dropdown menú o en un menú en cascada. Un dropdown menú es un menú bajo un ítem en la barra de menú. Un menú cascada es un menú que aparece al lado de un ítem de un dropdown menú.

El diseño de un buen menú para una aplicación, es un trabajo que reviste la mayor importancia. Debe facilitar el acceso a las distintas tareas que realiza la operación en una forma ordenada y clara. La definición de la barra de menú y los dropdown menú, debe ser

tal, que el usuario identifique rápida e intuitivamente los ítems que debe seleccionar para cada tarea.

## Funciones globales

PowerBuilder permite definir dos tipos de funciones:

- **Funciones a nivel de objeto**, se definen para un tipo particular de ventana, menú, u otro tipo de objeto, y son encapsuladas dentro del objeto en el que se definen.
- **Funciones globales**, no están encapsulados dentro de ningún objeto; son almacenados como objetos independientes.

Al contrario de las funciones a nivel de objeto, las funciones globales no actúan en una instancia particular de un objeto. En lugar de eso, se crean para ejecutar procesamiento de propósito general, tales como cálculos matemáticos o manejo de string.

## Queries

Una query es una sentencia SQL que se graba con un nombre, de tal forma que puede usarse repetidamente y como fuente de datos para los DataWindow object. Los queries mejoran la productividad, ya que se crean una vez y pueden reusarse tan a menudo como sea necesario.

## Estructuras

Una estructura es una colección de una o más variables relacionadas, del mismo o diferente tipo de dato, agrupadas bajo un solo nombre. En algunos lenguajes, tales como Pascal y COBOL, las estructuras se llaman registros.

Las estructuras permiten referirse a entidades relacionadas como una unidad, en lugar de individualmente. Por ejemplo, si se define: dirección, nivel de acceso y un picture (bitmap) del empleado como una estructura llamada user\_struct, se puede referir a esta colección de variables como user\_struct.

Hay dos tipos de estructuras:

- **Estructuras a nivel de objeto**, las cuales están asociadas con un tipo particular de objeto, tal como una ventana o menú. Estas estructuras pueden siempre ser usadas en script dentro del objeto. Sin embargo, también pueden ser accedidas desde script fuera del objeto que las contiene.
- **Estructuras globales**, las cuales no están asociadas con un objeto particular. Se pueden referenciar directamente desde cualquier lugar de la aplicación.

## Objetos de usuario

Las aplicaciones a menudo tienen características en común. Por ejemplo, se podría usar a menudo un botón ‘Close’ que ejecute un cierto conjunto de operaciones, y luego cerrar una ventana. O un listbox que liste todos los departamentos. O se puede desear que todos los DataWindow controls ejecuten un mismo chequeo de error. O un visor predefinido de archivos que pueda ponerse en las ventanas cada vez que se necesite.

Si el programador encuentra que está usando la misma característica de aplicación repetidamente, debiera definir un objeto de usuario. El objeto de usuario se define una vez, y se usa tantas veces como se requiera. No se necesita redefinir cada vez el objeto.

Hay dos tipos de objetos de usuario:

- **Visual**

Un objeto de usuario visual es un control o conjunto de controles visual y reusable que tiene un cierto comportamiento. Por ejemplo, un objeto de usuario consistente de varios botones que funciona como una unidad. Los botones podrían tener script asociado que ejecuta procesamiento standard. Una vez que se define el objeto, se puede usar cuando se necesite.

- **Class**

En algunas situaciones, se necesita reusar módulos de proceso que no tienen componentes visuales. Por ejemplo, calcular comisiones o ejecutar análisis estadístico. Para hacer esto, se define un objeto de usuario de tipo Clase. Generalmente, se usan objetos de usuario Clase, para definir reglas del negocio (business rules) y otros procesamientos que actúan como una unidad. Para usar un objeto de usuario Clase, se crea la instancia del objeto en un script y luego se llaman las funciones que se hayan definido para la clase.

## Librerías



Las librerías son archivos (.PBL), en los cuales se graban los objetos que forman la aplicación, tales como ventanas y menús. Cuando se corre la aplicación, PowerBuilder recupera los objetos de las librerías. Las aplicaciones pueden usar tantas librerías como se requiera. Cuando se crea una aplicación, se debe especificar en qué librería estará.

## Proyectos



Para permitir que los usuarios ejecuten una aplicación de la misma forma que ejecutan otras aplicaciones Windows, se crea un objeto proyecto. El objeto proyecto puede empaquetar una aplicación en una de las siguientes formas:

- **Como un archivo standalone**, que contiene todos los objetos en la aplicación.
- **Como un archivo ejecutable** y una o más librerías dinámicas PowerBuilder que contienen objetos que se linkean en tiempo de ejecución.

Se debe además, proveer algunos recursos adicionales que la aplicación usa, tales como bitmaps e íconos. Hay dos formas de proporcionar estos recursos: incluirlos en el ejecutable y/o librerías dinámicas, o distribuirlos separadamente.

# **El medio ambiente PowerBuilder**



## El medio ambiente PowerBuilder

Cuando se inicia PowerBuilder, se abre una ventana que contiene una barra de menú y la PowerBar.

El usuario puede abrir painters (pintores) y ejecutar tareas dando click en la PowerBar.

### Painters

El programador construye componentes de la aplicación usando los painters, los cuales proveen una variedad de herramientas para construcción de objetos.

Por ejemplo, una ventana se construye en el Window painter. Allí se define las propiedades de la ventana y se agregan a ella controles, tales como buttons y edit boxes.

PowerBuilder tiene un painter por cada tipo de objeto que puede construirse. Así, existen:

- Application painter
- Window painter
- Menu painter
- DataWindow painter
- Structure painter
- Table painter
- Database painter
- Pipeline painter
- Query painter
- Function painter
- Project painter
- Library painter
- User Object painter
- Debugger

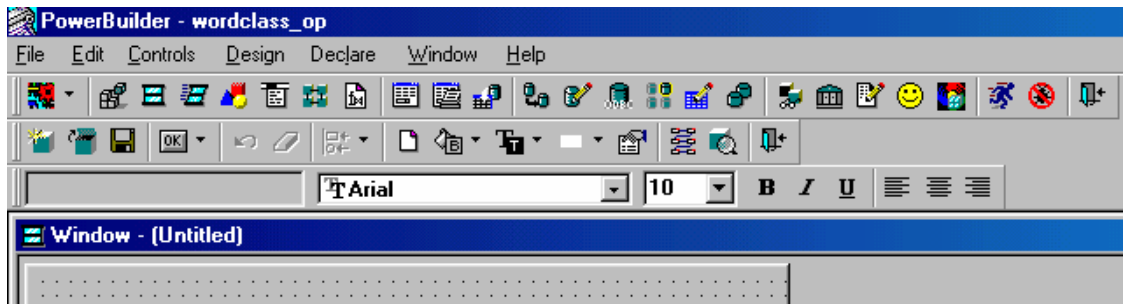
## PowerBar



La PowerBar se muestra cuando comienza una sesión de PowerBuilder. La PowerBar es el punto de control principal para construir aplicaciones PowerBuilder. Desde la PowerBar, se puede abrir un painter, hacer debug o correr la aplicación actual, ver la ayuda PowerBuilder, o configurar PowerBuilder para adaptarlo a nuestras necesidades.

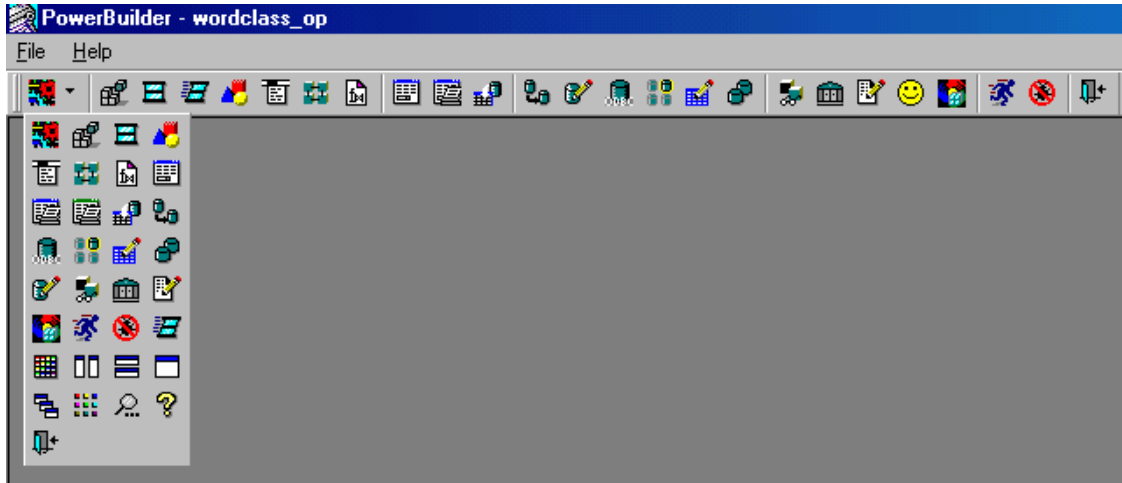
PowerBuilder proporciona dos barras de herramientas (toolbars ) más que se pueden usar como punto de partida para realizar tareas.

## PainterBar



Cuando se abre un pintor, PowerBuilder muestra una nueva ventana que tiene un área de trabajo (workspace) en la cual se diseña el objeto que se construye. Además, PowerBuilder muestra una PainterBar. La PainterBar tiene botones para manipular las componentes en el actual pintor.

## PowerPanel



PowerBuilder también proporciona un PowerPanel, el cual, como la PowerBar, provee botones para abrir pintores y ejecutar otras actividades. El PowerPanel no es configurable, pero contiene todos los pintores y herramientas que están globalmente disponibles a través de PowerBuilder.

Usualmente se usará la PowerBar para abrir pintores, pero la PowerPanel es cómoda si se desea abrir un painter que no está disponible en el momento en la PowerBar.

Ambas toolbars muestran PowerTips – una ayuda de navegación – y pueden configurarse para adaptarlas a las propias necesidades.

## **Painters and Tools**

---

## Painters and Tools

A continuación, se da una breve explicación de cada uno de los painters y tools disponibles.



### Application painter

Use el Application painter para especificar información acerca de la aplicación, tal como, nombre y librerías de la aplicación.

También, se usa para crear el objeto aplicación, ver su estructura y modificar sus propiedades.

Use 'Properties', para setear:

Fonts para header, column, label de textos.

Variables de la aplicación

Library search paths donde los objetos de la aplicación se grabarán

Comentarios del objeto aplicación.

### Project painter

Use el Project painter para crear el ejecutable de la aplicación, especificando los componentes que la constituyen.

En el project painter, se define:

El nombre del archivo ejecutable

Librerías dinámicas (archivos .PBD) a generar

Archivos de recursos ( .PBR): bitmaps, íconos

## Window painter



Use el Window painter para:

- Definir una ventana
- Asociarla con un menú
- Poner controles en la ventana
- Modificar las propiedades de la ventana

Use el cuadro de diálogo 'Properties', para setear:

- Tipo
- Tamaño y localización
- Icono
- Scrollbar para desplazamiento
- Toolbar

## Run Window



Permite ejecutar una ventana. No es una previsualización de ésta; realmente se ejecuta el código contenido en cada objeto de la ventana. Si la ventana tiene código que accesa a una base de datos, la conexión debiera estar disponible en algún lugar de la ventana. Por ejemplo, se puede agregar en el evento open de la ventana el script necesario para la conexión, extraído del objeto aplicación que es donde usualmente se codifica.

## User Object painter



Este painter se utiliza para construir objetos de usuario que serán usados repetidamente en la aplicación.

## Menu painter

Cada selección de un menú se define como un 'MenuItem' en PowerBuilder. Los Menuitems se muestran en un menú bar, en un dropdown o cascading menú. Un dropdown menú es un menú bajo un ítem en un menú bar. Un cascading menú es un menú al lado de un ítem en un dropdown menú.

Use los tabs del Menu painter para definir:

Nombres y estilos de Menu item

Texto para MicroHelp

Teclas aceleradoras de menú

Opciones de Toolbar

Para agregar MenuItems al menú bar:

- De click al espacio vacío a la derecha del último MenuItem definido en el menu bar.
- Ingrese el texto que mostrará el MenuItem
- Para agregar otro MenuItem al menu bar, de click a la derecha del MenuItem que acaba de definir.

## Structure painter

Use el Structure painter para definir estructuras (grupos de variables) para uso en la aplicación.

Para crear una nueva estructura:

- De click al botón de structure painter en el PowerBar
- De click en New
- Ingrese la información de la nueva estructura, tales como variables y tipo de datos
- De click en el botón Close en el PowerBar, ingrese un nombre para la estructura y de OK.

## Function painter



Use el Function painter para construir funciones que ejecuten procesamiento específico a la aplicación.

Para crear una nueva función:

- De click al botón function painter en el PowerBar
- De click en New
- Ingrese la información de la función: nombre, tipo de dato que retorna y argumentos que se le envían.

Las funciones creadas en este painter, son globales, es decir, se pueden acceder desde cualquier parte de la aplicación.

## DataWindow painter



Los Datawindows son objetos para obtener (retrieve), presentar y manipular datos desde una base de datos relacional u otra fuente de datos, tal como Excel o dBASE.

Se pueden elegir varios estilos de presentación. Por ejemplo, mostrar los datos en un formato fila/columna o en un formato freeform (libre).

En el DataWindow painter se define una fuente de datos y su estilo de presentación. Además, también se define:

Formato de display  
Reglas de validación  
Ordenamiento y criterios de filtro  
Gráficos para el DataWindow object.

Use 'Properties', para setear:

Unidades de medida  
Colores por defecto  
Especificación de impresión



Para crear un DataWindow:

- De click al botón DataWindow en el PowerBar o PowerPanel  
La caja de diálogo de DataWindow lista los DataWindow objects en la librería actual.
- De click al botón New.
- Seleccione un estilo de presentación para el DataWindow
- Seleccione una fuente de datos para el DataWindow Object.
- (Opcional) Mejore la presentación del DataWindow en el DataWindow painter. Por ejemplo, el formato de las columnas, añadir objetos tales como texto, bitmaps, campos calculados, sumas, filtros ordenamientos, grupos y gráficos.

## Report painter

Use el Report painter para construir reportes que presentan información de la base de datos. Un reporte PowerBuilder tiene todas las características de un DataWindow object , salvo que no sirve para realizar actualizaciones a la base de datos.

En el Report painter se define una fuente de datos y un estilo de presentación. Se puede definir también, formatos de display, reglas de validación, criterios de ordenamiento y filtro, gráficos para el DataWindow object.

Mediante el cuadro de diálogo ‘Properties’, se setean:

Unidades de medida  
Colores por defecto  
Especificaciones de impresión

## Query painter

Use el Query painter para definir gráficamente sentencias SQL para reusar en los DataWindow objects. Se pueden definir argumentos de retrieve, uniones, sort, y criterios de agrupación.

## Data Pipeline painter

Use el Pipeline painter para copiar datos desde una base de datos a otra. En el Data Pipeline painter se define una fuente de datos, una base de datos fuente, una base de datos destino, argumentos para retrieve, uniones, orden, selección y criterios de grupo.

## Database Administration painter

Use el Database Administration painter para controlar acceso a las bases de datos, crear y modificar usuarios y grupos, crear, importar, pegar y ejecutar statements SQL (sentencias SQL).

Se puede configurar el Database Administration painter y setear:

Fonts

Indentación

Color para palabras claves en la sintaxis.

## Configure ODBC

Permite definir un profile (perfil o definición) de conexión. Aquí se especifica un nombre de **fuentes de datos** con el que luego se podrá acceder una base de datos. Entre otros parámetros, se indica: tipo de base de datos, servidor (si lo hay), nombre de la base de datos, user id, password, si la base de datos es local o está en una red, etc.

## DB Profile

Permite seleccionar un profile previamente definido, con el cual la aplicación se conectará a la base de datos correspondiente.

## Table painter

Use el Table painter para crear y alterar tablas de bases de datos, definiendo columnas, headers, formatos de display, criterios de validación y estilos de edición.

Use el cuadro de diálogo 'Properties' para:

Editar comentarios de la tabla

Editar fonts y datos de la tabla, headings, y labels

Crear y editar índices de la tabla

## Database painter

Use el Database painter para:

- Mantenición de bases de datos
- Crear índices y foreign keys (clave foránea)
- Alterar foreign keys
- Drop (eliminar) índices, foreign keys, vistas y tablas.

Use 'Properties', para:

- Editar comentarios de una tabla
- Definir fonts para los datos, heading y labels
- Crear, alterar y eliminar claves primarias

Usando column properties se puede:

- Editar comentarios de una columna
- Definir headers

## Library painter

Use el library painter para crear y mantener las librerías de objetos PowerBuilder que forman la aplicación. Los objetos son cosas tales como: aplicación, ventanas, menús y otros, los cuales se graban en las librerías.

Use el diálogo 'Properties' para modificar la descripción de un objeto librería. Para ver este cuadro de diálogo, seleccione una librería y de click al botón derecho del mouse.

## Edit

Use el editor de archivos para editar archivos tales como:

- Archivos fuente (\*.sr\*)
- Archivos de recursos (\*.pbr)
- Archivos de inicialización (\*.ini)
- Archivos de texto (\*.txt)

## Run

Permite ejecutar la aplicación. Si se ha estado modificando alguno de los elementos que forman la aplicación, se le indicará si desea grabar los cambios.

## Debug

Permite hacer un debug o seguimiento de la aplicación para los objetos y eventos que se especifiquen. La finalidad de depurar, es encontrar errores o verificar que los procesos se ejecutan del modo esperado.

## Exit

Termina la ejecución de la aplicación en el ambiente PowerBuilder.

## PowerScript painter

Use el PowerScript painter para escribir script o código en la aplicación.

Para crear o modificar un script:

- Abra el objeto al cual se le escribirá código
- Abra el PowerScript painter del objeto
- Escriba código PowerScript
- Compile ( Ctrl-L) el script y retorne al pintor previo

# **Lenguaje PowerScript**

## Lenguaje PowerScript

Todo lenguaje de programación contiene sentencias propias que le permiten manipular la información, permite separar cursos de acción dependiendo de ciertos valores o circunstancias, o bien sencillamente presentarla al usuario. Todo esto debe ser realizado en forma autónoma por el programa, y a veces sin participación externa.

La siguiente es una lista de los principales tópicos del lenguaje PowerScript, utilizados en la construcción de una aplicación PowerBuilder.

### Operadores para expresiones

PowerBuilder soporta los siguientes tipos de operadores en expresiones:

#### Aritméticos para tipos de datos numéricos

Operador	Significado	Ejemplo
+	Suma	Total = SubTotal + Impuesto
-	Resta	Precio = Precio - Descuento
*	Multiplicación	Total = Cantidad * Precio
/	División	Factor = Descuento/Precio
^	Exponenciación	Rank = Rating ^ 2.4

#### Relacionales para todo tipo de datos

PowerBuilder usa operadores relacionales en expresiones relacionales para evaluar dos o más operandos. El resultado es siempre TRUE o FALSE.

La siguiente tabla lista los operadores relacionales

Operador	Significado	Ejemplo
=	Igual	If Precio = 100 Then Razon = .05
>	Mayor que	If Precio > 100 Then Razon = .05
<	Menor que	If Precio < 100 Then Razon = .05
<>	Distinto que	If Precio <> 100 Then Razon = .05
>=	Mayor o igual	If Precio >= 100 Then Razon = .05
<=	Menor o igual	If Precio <= 100 Then Razon = .05

## Lógicos para todo tipo de datos

PowerBuilder usa operadores lógicos para formar expresiones booleanas. El resultado de evaluar una expresión booleana es siempre TRUE o FALSE.

Estos son los operadores lógicos:

Operador	Significado	Ejemplo
NOT	Negación lógica	If NOT Precio = 100 Then Razon = .05
AND	Y lógico	If Impuesto > 3 AND Ship < 5 Then R = .05
OR	O lógico	If Impuesto > 3 OR Ship < 5 Then RT = .07

## Evaluación de valores nulos (NULL)

Cuando se forma una expresión booleana que contiene un valor nulo, y se evalúa teniendo AND / OR, piense en NULL como algo no definido (no es TRUE ni FALSE).

### Ejemplo

En este ejemplo, la expresión involucra la variable f, la cual se setea a NULL.

boolean d, e = TRUE, f

SetNull(f)

d = e and f // d es NULL

d = e or f // d es TRUE

## Concatenación para datos de tipo string

El operador de concatenación une el contenido de dos variables del mismo tipo para formar otra mayor. Se pueden concatenar variables tipo string y blobs.

Para concatenar valores, use el signo más (+) como operador.

### Ejemplos

string Test

Test = "over" + "stock" // Test contiene "overstock"

string Lname, Fname, FullName

FullName = Lname + ' ' + Fname

// FullName contiene el apellido y el nombre separados por una coma

// y un espacio.

Este ejemplo muestra cómo un blob actúa como un acumulador cuando se lee datos de un archivo.

```
integer i, fnum, loops
```

```
blob tot_b, b
```

```
...
```

```
FOR i = 1 to loops
```

```
    bytes_read = FileRead(fnum, b)
```

```
    tot_b = tot_b + b
```

```
NEXT
```



## Tipos de datos

La siguiente tabla lista todos los tipos de datos standard PowerScript.

Tipo de dato	Descripción
Blob	Binary large object. Usado para almacenar una cantidad de datos sin límite. Por ejemplo, imágenes o un texto largo, como un documento de Word.
Boolean	Contiene TRUE o FALSE.
Char o Character	Un carácter ASCII
Date	Fecha, incluye año completo (2000), mes (01 a 12), y el día (01 a 31)
DateTime	Fecha y hora en un solo tipo de dato. Usado sólo para lectura y escritura de valores DateTime hacia o desde una base de datos. Para convertir valores DateTime a tipos de datos que puedan usarse en PowerBuilder, usar: <ul style="list-style-type: none"> <li>• Date(datetime) para convertir un valor DateTime a un valor de fecha PowerBuilder, después de leerlo desde una base de datos</li> <li>• Time(datetime) para convertir un valor DateTime a un valor de hora PowerBuilder, después de leerlo desde una base de datos.</li> <li>• DateTime(date,time) para convertir una fecha y (opcional) una hora a un valor DateTime antes de escribirlo a una columna DateTime en una base de datos.</li> </ul> <p>PowerBuilder soporta microsegundos en la interfáz de base de datos para cualquier DBMS que los soporte.</p>
Decimal o Dec	Números decimales con signo hasta 18 dígitos. El punto decimal se puede poner en cualquier parte dentro de los 18 dígitos. Por ejemplo 123.456, 0.000001 o 12345678901234.5697.
Double	Números de punto flotante con signo con 15 dígitos de precisión y un rango desde 2.2E-308 hasta 1.7E+308
Integer o Int	Enteros de 16 bits con signo, desde -32768 hasta +32767.
Long	Enteros de 32 bits con signo, desde -2,147,483,648 hasta + 2,147,483,647.
Real	Números de punto flotante con signo con 6 dígitos de precisión y un rango desde 1.17E-38 hasta 3.4E+38.
String	Caracteres ASCII con longitud variable ( 0 a 60.000).
Time	La hora en formato 24 horas, incluyendo la hora (00 a 23), minutos (00 a 59), segundos (00 a 59), y fracción de segundo (hasta 6 dígitos), con un rango desde 00:00:00 a 23:59:59.99999. PowerBuilder soporta microsegundos en la interfáz de base de datos para cualquier DBMS que los soporte.
UnsignedInteger, UnsignedInt o UInt	Enteros de 16 bits sin signo, desde 0 a 65,535.
UnsignedLong, Ulong	Enteros de 32 bits sin signo, desde 0 a 4,294,967,295.

PowerBuilder también soporta el tipo de dato 'Any', el cual puede aceptar cualquier tipo de dato, incluyendo los tipos standard de datos, objetos y estructuras.

## Acerca de las variables PowerScript y constantes

Se pueden definir variables y constantes y usarlas en los script. Las variables PowerScript tienen un alcance, un tipo de dato, y un valor inicial.

### Variables

Una variable puede ser un tipo de dato, un objeto, o una estructura. A la mayoría de las variables se les asigna un valor cuando se las declara. De todas formas, siempre se puede cambiar su valor dentro del script.

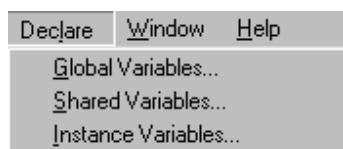
### Constantes

A cualquier declaración de un tipo de dato standard se le puede agregar una asignación de valor inicial para que sea una constante en lugar de una variable. Para hacerla constante, incluya la palabra clave `CONSTANT` en la declaración. Una constante debe tener asignado un valor en la declaración, y no se le puede asignar un valor en el código que le sigue.

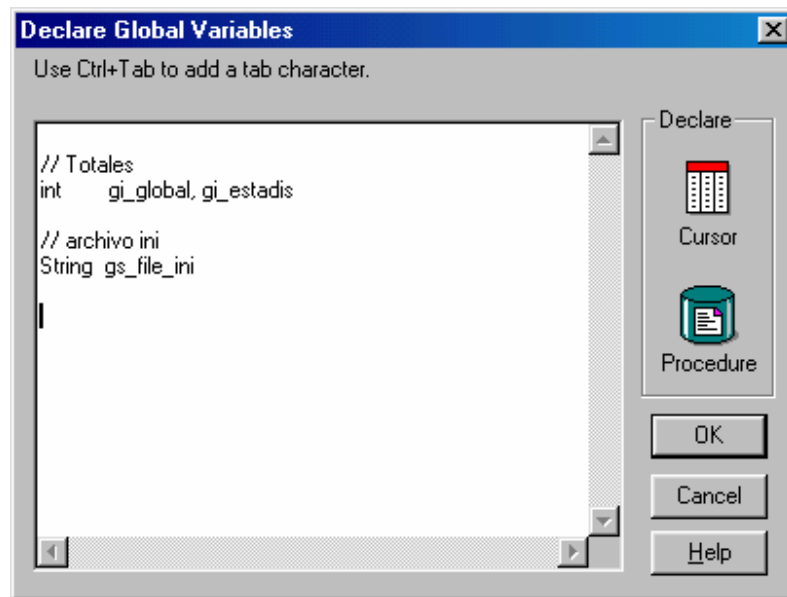
### Ámbito

El ámbito de una variable determina donde la variable se reconoce. Existen cuatro ámbitos: Global, Instance, Shared y Local.

Los tres primeros son definidos fuera de cualquier módulo ejecutable y se declaran por ejemplo, en el área de trabajo de las ventanas y objetos de usuario, en el menú 'Declare'.



Todos estos tienen la misma ventana de ingreso de variables, solo cambiando el ámbito (rango) de alcance de la variable dentro del programa.



Los ámbitos son:

- **Global**

Una variable que es accesible desde cualquier lugar de la aplicación. Es independiente de cualquier definición de objeto.

- **Instance (globales al objeto)**

Una variable que pertenece a un objeto y está asociada con una instancia de ese objeto (es como una propiedad del objeto). Un acceso directo a una variable de instance sólo puede realizarse desde el objeto al cual pertenece. Para accederla desde otro lugar, debe anteponerse el objeto en el cual se creó, con notación de punto. Ejemplo: w\_estadistica.ic\_valor\_promedio.

Las variables de instances pueden pertenecer al objeto aplicación, una ventana, un objeto de usuario, o un menú.

- **Shared**

Una variable que pertenece a un objeto y existe a través de todas las instancias del objeto. Tienen la particularidad de *conservar* o *recordar* el valor almacenado en ésta al momento de cerrar y abrir el objeto en que fueron definidas. Las variables shared son siempre privadas. Son accesibles sólo en script del objeto y controles asociados al objeto.

Las variables de instancias pueden pertenecer al objeto aplicación, una ventana, un objeto de usuario, o un menú.

- **Local (al módulo)**

Una variable temporal que es accesible sólo en el script en la cual se define. No son compartidas por otros módulos del mismo objeto inclusive. Cuando el script finaliza, la variable cesa de existir.

Una variable se define anteponiéndole el tipo de dato que será permitido para esa variable.

Ej. Se desea declarar una variable de tipo entera de nombre 'nCont', y una variable tipo string de nombre cDescrip:

```
Integer nCont  
String cDescrip
```

Se pueden declarar varias variables del mismo tipo, separando cada una de éstas con una coma.

Ej.

```
Integer nNumCuota , nTotal = 0
```

En el caso de la variable nTotal, se le asignó un valor inicial de 0. Esto puede hacerse, teniendo el cuidado de asignar un valor de acuerdo al tipo que se define.

Al declarar una variable como una constante, el valor asignado no se podrá modificar durante el script. Para declarar una variable como una constante, se antepone la palabra **CONSTANT**.

## Sintaxis para llamada de funciones

La siguiente sintaxis trabaja para declaración de funciones de sistema, objetos y externas. Debido a que el nombre de una función puede repetirse, PowerBuilder usa la que encuentra primero.

Sintaxis

*functionname* ( { *argumentlist* } )

Parámetro	Descripción
<i>functionname</i>	El nombre de la función externa, global o de objeto.
<i>argumentlist</i>	Los valores que se desea pasar a <i>functionname</i> . Cada valor en la lista que se pase debe tener el tipo de dato que corresponda al tipo declarado en la definición de la función.

Si la función retorna un valor, se puede llamar la función a la derecha de una sentencia de asignación, como un argumento para otra función, o como un operando en una expresión.

Una función es externa si está contenida en un archivo .DLL o .EXE de librería. Antes de usar una función externa, ésta debe ser declarada.

## Nombres identificadores en PowerScript

Se usan los identificadores para nombres de variables, labels, funciones, windows, controls, menus, y cualquier cosa a que se haga referencia en los script.

Reglas para los identificadores

- Deben comenzar con una letra.
  - Pueden tener hasta 40 caracteres, pero no espacios.
  - Son case insensitive (PART, Part, y part son idénticos).
  - Pueden incluir cualquier combinación de letras, números, y estos caracteres especiales.
- Dash
  - \_ Underscore
  - \$ Dollar sign
  - # Number sign
  - % Percent sign

## Pronombres en script

PowerBuilder incluye cuatro pronombres que permiten hacer una referencia general a un objeto o control. Al usar un pronombre, la referencia funciona correctamente incluso si el nombre del objeto o control cambia.

Estos pronombres pueden usarse en funciones y script donde se usaría el nombre del objeto. Por ejemplo, se usa un pronombre para:

- Causar un evento en un objeto o control
- Manipular o cambiar un objeto o control
- Obtener o cambiar el seteo de una propiedad

Cada pronombre tiene un significado y uso específico como se describe en la tabla:

Pronombre	En un script para un...	Se refiere a...
Parent	Control en una ventana	La ventana que contiene el control.
	Control en un objeto de usuario	Objeto de usuario que contiene el control.
	Menú	Item de menú en el nivel anterior al actual menú.
This	Ventana, objeto de usuario, menú, objeto aplicación, o control	Objeto o control
ParentWindow	Menú	Ventana a la cual está asociado el menú en tiempo de ejecución.
Super	Objeto o control descendiente	Parent
	Ventana u objeto de usuario descendiente	El ancestro inmediato de la ventana u objeto de usuario descendiente
	Un control en una ventana u objeto de usuario	El ancestro inmediato de la ventana u objeto de usuario que contiene el control

## Comentarios en Scripts

Los comentarios se usan para documentar los scripts y para prevenir que algunas sentencias se ejecuten.

Hay dos formas para designar comentarios en PowerBuilder:

- // El doble slash  
Designa toda o parte de una línea como un comentario
- /\* .. \*/ El slash y asterisco  
Designa toda o parte de una línea como un comentario, o varias líneas como un solo comentario.

## Nota

---

En el PowerScript painter y el Function painter, se pueden usar los botones de comentarios para seleccionar una o más líneas y hacerlas comentario.

---

## Palabras reservadas PowerScript

Las palabras que PowerBuilder usa internamente se llaman palabras reservadas y no pueden ser usadas como identificadores.

Alias	Event	Not	Shared
And	Execute	Of	Static
Autoinstantiate	Exit	On	Step
Call	External	Open	Subroutine
Case	False	Or	Super
Choose	Fetch	Parent	System
Close	First	Post	Systemread
Commit	For	Prepare	Systemwrite
Connect	Forward	Prior	Then
Constant	From	Private	This
Continue	Function	Privateread	To
Create	Global	Privatewrite	Trigger
Cursor	Goto	Procedure	True
Declare	Halt	Protected	Type
Delete	If	Protectedread	Until
Describe	Immediate	Protectedwrite	Update
Descriptor	Indirect	Prototypes	Updateblob
Destroy	Insert	Public	Using
Disconnect	Into	Readonly	Variables
Do	Intrinsic	Ref	While
Dynamic	Is	Return	With
Else	Last	Rollback	Within
Elseif	Library	Rpcfunc	_debug
End	Loop	Select	
Enumerated	Next	Selectblob	

## Sentencias PowerBuilder

### Asignación

Use el signo igual (=), para asignar un valor a una variable.

### Sintaxis

*varname = expression*

Parámetro	Descripción
<i>Varname</i>	El nombre de la variable o elemento de arreglo al cual se desea asignar un valor.
<i>Expression</i>	El valor que se asina a la variable. Puede ser una expresión, un string, o un literal.

### Ejemplos

A=C+9

Class="COBOL"

PAYMENT=INTEREST(12)+36.50

También se puede usar el signo igual como un operador relacional en la expresión. De este modo, el primer signo igual en una sentencia denota la asignación.

### Ejemplo

El valor de la expresión B=C determina si el valor de A en esta sentencia es TRUE o FALSE:

A = B = C

Para asignar valores a atributos de un objeto, use la puntuación de punto PowerBuilder en la sentencia de asignación.

### Ejemplo

cbx\_direct.Checked = True

w\_main.title = "Gestión de compras"



### Asignando valores a arreglos

Se puede asignar múltiples valores a un arreglo con una sola sentencia de asignación, tal como:

```
int    Arr[]
```

```
Arr = { 1, 2, 3, 4 }
```

También se puede copiar todo un arreglo a otro. Por ejemplo:

```
Arr1 = Arr2
```

Copia el contenido de Arr2 en el arreglo Arr1.

### Abreviaturas de asignación

PowerBuilder provee las siguientes abreviaturas que se pueden usar al asignar valores a variables.

Esta abreviatura	Usada aquí	Es equivalente a
++	i++	i = i + 1
--	i--	i = i - 1
+=	i+=3	i = i + 3
-=	i-=3	i = i - 3
*=	i*=3	i = i * 3
/=	i/=3	i = i / 3
^=	i^=3	i = i ^ 3

Si usa guiones en nombres de variables, deje un espacio después de – y -=, si no, PowerBuilder considera el signo menos parte del nombre de la variable.

### Nota

Las abreviaturas se permiten sólo en sentencias de asignación. No pueden usarse con otros operadores en una sentencia.

### Ejemplos

Estas son sentencias válidas de asignación:

```
int    i = 4
```

```
i ++      // i vale 5.
```

```
i --      // i vale 4 nuevamente.
```

```
i += 10      // i vale 14.
i /= 2       // i vale 7.
```

La última sentencia de asignación siguiente es inválida, debido a que ++ no se usó separadamente.

```
int i, j

i = 12
j = i ++      // INVALIDA
```

La siguiente sentencia usando ++ es válida.

```
int    i, j

i = 12
i ++   //VALIDA
j = i
```

### Usando atributos en Script

Los atributos, los cuales se asocian con cada objeto, determinan cómo luce y se comporta un objeto. Por ejemplo, los atributos de un CommandButton determinan su tamaño, localización, texto, y si es visible o no.

En los scripts se usan los atributos para cambiar la apariencia o comportamiento de un objeto, o bien para obtener información acerca del objeto.

Para cambiar los atributos de un objeto, use la sentencia de asignación junto a la notación punto (el nombre del objeto seguido inmediatamente de un punto) para identificar el objeto al cual se desea cambiar el atributo.

### Nota

---

Si no se especifica el objeto, PowerBuilder cambia el atributo del objeto en el cual se escribe el script.

---

## Sintaxis

*{objectname.}attributename = value*

Parámetro	Descripción
<i>Objectname</i> (opcional)	El nombre del objeto al cual se desea asignar un valor a un atributo. El defecto para <i>objectname</i> es el objeto actual. Note que el punto entre <i>objectname</i> y <i>attributename</i> es requerido si se entra el nombre del objeto. Nota: Si el objeto es el objeto donde se escribe el script, se puede usar la palabra reservada <i>this</i> para referirse al objeto.
<i>Attributename</i>	El nombre del atributo que se cambia.
<i>Value</i>	El nuevo valor del atributo.

## Ejemplos

Esta sentencia cambia el ancho del CommandButton *cb\_cancel*:

```
Cb_Cancel.width = 100
```

Esta sentencia habilita el CheckBox *cbx\_box*:

```
Cbx_box.enabled = True
```

Esta sentencia habilita el menuitem llamado *m\_file* en el menú *menu\_appl*:

```
menu_appl.m_file.enabled=TRUE
```

## CHOOSE CASE

La estructura de control CHOOSE CASE direcciona la ejecución del programa basado en el valor de una expresión (usualmente una variable):

Sintaxis

```
CHOOSE CASE testexpression
CASE expressionlist
    statementblock
{CASE expressionlist
    statementblock
...
CASE expressionlist
    statementblock}
{CASE ELSE
    statementblock}
END CHOOSE
```

Parámetro	Descripción
<i>testexpression</i>	La expresión en que se basará la ejecución del script
<i>expressionlist</i>	Una de las siguientes expresiones <ul style="list-style-type: none"> <li>• un valor .</li> <li>• una lista de valores separadas por comas ( Ej: 2,4,5,7).</li> <li>• Una cláusula TO (Ej: 1 To 30 ).</li> <li>• IS seguido de un operador relacional y valor a comparar ( E: IS &lt; 5).</li> <li>• Cualquier combinación de las anteriores con un OR implícito entre las expresiones (Ej: 1,3,5,7 TO 33, IS &gt; 42).</li> </ul>
<i>statementblock</i>	El bloque de sentencias que PowerBuilder debe ejecutar si la expresión de test coincide con los valores en <i>expressionlist</i> .

Se requiere al menos un CASE. La estructura de control CHOOSE CASE finaliza con END CHOOSE.

Si *testexpression* al comienzo de una sentencia CHOOSE CASE coincide con un valor en *expressionlist* para una cláusula CASE, se ejecutan las sentencias que le siguen inmediatamente. El control pasa a la primera sentencia después de la cláusula END CHOOSE.

Si existen múltiples CASE, entonces *testexpression* se compara con cada *expressionlist* hasta encontrar una coincidencia, o hasta encontrar un CASE ELSE o un END CHOOSE.

Si existe una cláusula CASE ELSE, y el valor testeado no coincide con ninguna de las expresiones, se ejecuta *statementblock* de la cláusula CASE ELSE. Si no existe cláusula CASE ELSE y no hay coincidencia, el control pasa a la primera sentencia después de END CHOOSE.

## Ejemplos

Este ejemplo procesa de diferentes formas basado en el valor de la variable Weight.

CHOOSE CASE Weight

```
CASE IS<16
    Postage=Weight*0.30
    Method="USPS"
CASE 16 to 48
    Postage=4.50
    Method="UPS"
CASE ELSE
    Postage=25.00
    Method="FedEx"
END CHOOSE
```

Este ejemplo convierte el texto en un SingleLineEdit control, a un valor real y realiza distintos procesos según su valor.

```
CHOOSE CASE Real(sle_real.Text)
CASE is < 10.99999
    sle_message.Text = "Real Case < 10.99999"
CASE 11.00 to 48.99999
    sle_message.Text = "Real Case 11 to 48.9999"
CASE is > 48.9999
    sle_message.Text = "Real Case > 48.9999"
CASE ELSE
    sle_message.Text = "No se puede evaluar!"
END CHOOSE
```

## CONTINUE

Use la sentencia CONTINUE en una estructura de control DO..LOOP o FOR..NEXT.

- **En una estructura DO...LOOP**

Cuando PowerBuilder encuentra una sentencia CONTINUE en un DO..LOOP, el control pasa a la siguiente sentencia LOOP. Las sentencias entre la sentencia CONTINUE y la sentencia LOOP son saltadas en la actual iteración DO..LOOP. En una estructura DO..LOOP anidada, una sentencia CONTINUE bypassa las sentencias en la estructura actual DO..LOOP.

### Ejemplo

La siguiente sentencia muestra un mensaje dos veces: cuando B es igual a 2 y cuando B es igual a 3. Tan pronto B es mayor que 3, se encontrará siempre con el CONTINUE, hasta que A sea mayor o igual que 100.

```
int A=1, B=1
```

```
DO WHILE A < 100
```

```
    A = A+1
```

```
    B = B+1
```

```
    if B > 3 then CONTINUE
```

```
    MessageBox("Hola", "B es " + String(B) )
```

```
LOOP
```

- **En una estructura FOR...NEXT**

Cuando PowerBuilder encuentra una sentencia CONTINUE en una estructura de control FOR..NEXT, el control pasa a la siguiente sentencia NEXT. Las sentencias entre la sentencia CONTINUE y la sentencia NEXT se saltan para la actual iteración FOR..NEXT.

### Ejemplo

Las siguientes sentencias paran de incrementar B tan pronto como Count es mayor que 15.

```
int A=0, B=0, Count
```

```
FOR Count = 1 to 100
```

```
    A = A + 1
```

```
    IF Count > 15 then CONTINUE
```

```
    B = B + 1
```

```
NEXT
```

// Después de completar: a=100 and b=15.

## CREATE

La sentencia CREATE genera una instancia de objeto para un tipo específico de objeto. Usualmente se usa para crear un nuevo objeto transaction.

La sentencia CREATE retorna una instancia de objeto que debe ser almacenada en una variable del mismo tipo.

### Sintaxis

CREATE *object\_type*

Parametro	Descripción
<i>Object_type</i>	El objeto tipo de dato que se desea crear.

### Ejemplo

Para crear una nueva transacción y almacenar el objeto en una variable 'dbtrans':

```
transaction dbtrans
```

```
dbtrans = CREATE transaction
```

## DESTROY

DESTROY elimina una instancia de objeto que fue creada con la sentencia CREATE. Después de una sentencia DESTROY, los atributos de la instancia creada ya no pueden ser referenciados.

### Sintaxis

DESTROY *objectvariable*

Parametro	Descripción
<i>Objectvariable</i>	Una variable cuyo tipo de dato es un objeto PowerBuilder.

## Ejemplo

El siguiente enunciado destruye el objeto transacción dbTrans que fué creado con una sentencia CREATE.

DESTROY DBTrans

## DO...LOOP

Use la sentencia de iteración de propósito general DO..LOOP para ejecutar un bloque de sentencias mientras (while) o hasta (until) que una condición sea verdadera ( TRUE).

Tip Los DO..LOOPS se pueden anidar.

## Sintaxis

DO WHILE tiene cuatro diferentes formatos:

- Haga sólo **mientras** la condición sea verdadera:

```
DO WHILE condition
    statementblock
LOOP
```

- Haga sólo **hasta** que la condición sea verdadera:

```
DO UNTIL condition
    statementblock
LOOP
```

- Haga una vez y repita **mientras** la condición sea verdadera:

```
DO
    statementblock
LOOP WHILE condition
```

- Haga una vez y repita **hasta** que la condición sea verdadera:

```
DO
    statementblock
LOOP UNTIL condition
```

Parámetro	Descripción
-----------	-------------



<i>Condition</i>	La condición que se desea testear.
<i>Statementblock</i>	El bloque de sentencias que se desea repetir.

DO marca el comienzo del bloque a repetir; LOOP marca el final.

DO WHILE y DO UNTIL testean la condición antes de ejecutar el bloque de sentencias.

Use LOOP WHILE y LOOP UNTIL, cuando se desee ejecutar siempre el bloque al menos una vez. Este formato comienza a testear la condición después que el bloque se ha ejecutado una primera vez.

### Nota

Una sentencia EXIT en cualquier lugar dentro de una estructura de control DO..LOOP, causa que el control pase a la primera sentencia después del LOOP. En una estructura DO..LOOP anidada, una sentencia EXIT pasa el control fuera de la actual estructura DO..LOOP.

### Ejemplos

La siguiente sentencia DO UNTIL ejecuta un bloque de funciones BEEP hasta que A es mayor que 15.

```
Int    A = 1, B = 1

DO UNTIL A > 15
    BEEP(A)
    A = (A + 1) * B
LOOP
```

La siguiente sentencia DO WHILE ejecuta un bloque de funciones BEEP sólo mientras A es menor o igual a 15.

```
Int    A = 1, B = 1

DO WHILE A <= 15
    BEEP(A)
    A = (A + 1) * B
LOOP
```

La siguiente sentencia LOOP UNTIL ejecuta un función Beep y entonces continúa ejecutándola hasta que A es mayor 15.

```
Int    A = 1, B = 1

DO
    BEEP(A)
    A = (A + 1) * B
LOOP UNTIL A > 15
```

La siguiente sentencia LOOP WHILE ejecuta una función beep y luego continúa ejecutándola mientras A es menor o igual a 15.

```
Int    A = 1, B = 1

DO
    BEEP(A)
    A = (A + 1) * B
LOOP WHILE A <= 15
```

La siguiente sentencia Do While pasa el control a la siguiente sentencia cuando count = 11.

```
Int    count = 0

DO WHILE A < 11
    If count > 10 then EXIT
    count = count + 1
LOOP
A = B
```

La función beep en el siguiente ejemplo se ejecuta en las primeras 3 iteraciones del loop.

```
Int    A = 1, B = 1

DO WHILE A < 11
    B = B + 1
    If B > 3 then CONTINUE
    Beep(2)
LOOP
```

## EXIT

La sentencia EXIT se usa en estructuras de control DO..LOOP y FOR..NEXT.

- En **DO...LOOP**

Una sentencia EXIT dentro de una estructura de control DO..LOOP provoca que el control pase a la sentencia que sigue a LOOP. En DO..LOOP anidados, la sentencia EXIT pasa el control fuera del actual DO..LOOP.

Ejemplo

```
Int    Nbr[10]
```

```
Int    count = 0
```

```
DO WHILE count < 16
    IF Nbr[count] > 10 then EXIT
    count = count + 1
LOOP
```

```
// El control sale del LOOP cuando se encuentra un elemento mayor
// que 10 en el arreglo o cuando count es mayor que 16.
```

- En **FOR...NEXT**

Una sentencia EXIT dentro de una estructura de control FOR..NEXT causa que el control se pase a la sentencia que sigue al NEXT.

Ejemplo

```
Int    Nbr[10]
```

```
Int    count = 0
```

```
For count = 1 to 16
    IF Nbr[count]>10 then EXIT
    count=count+1
NEXT
```

```
// El control pasa aquí cuando se encuentra un elemento mayor que 10
// en el arreglo o cuando count es mayor que 16.
```

## FOR...NEXT

Use la sentencia FOR..NEXT para ejecutar una o más sentencias (bloque) un número específico de veces.

TIP: Use FOR..NEXT para inicializar un arreglo con un valor único.

### Sintaxis

```
FOR varname = start TO end { STEP increment }
    statementblock
NEXT
```

Parámetro	Descripción
<i>varname</i>	El nombre de la variable contador de iteraciones.
<i>start</i>	El valor de inicio del contador de iteraciones.
<i>end</i>	El valor final del contador de iteraciones.
<i>increment</i> (opcional)	El valor de incremento de la variable <i>varname</i> . Por defecto es 1.
<i>statementblock</i>	El bloque de sentencias que se desea repetir.

Nota. Las sentencias FOR..NEXT se pueden anidar.

### Ejemplos

Este For..Next suma 10 a A mientras n varía de 5 hasta 25.

```
FOR n = 5 TO 25
    A = A + 10
NEXT
```

Este For..Next suma 10 a A e incrementa n en 5 hasta llegar a 25.

```
FOR n = 5 TO 25 STEP 5
    A = A + 10
NEXT
```

Este For..Next puebla un arreglo

```
INT   Matrix[100,50,200]
FOR x = 1 to 100
    FOR y = 1 to 50
        FOR z = 1 to 200
            Matrix[x,y,z]
        NEXT
    NEXT
NEXT
```

## GOTO

Use GOTO para transferir el control de una parte a otra en un script PowerBuilder.

Sintaxis

*GOTO label*

Parámetro	Descripción
<i>label</i>	La etiqueta asociada con la sentencia a la cual se desea transferir el control. Una etiqueta es un identificador seguido de dos puntos (:). No use los dos puntos con la etiqueta en la sentencia GOTO.

## Ejemplo

Esta sentencia transfiere el control a la sentencia asociada con la etiqueta OK.

GOTO OK

...

OK:

## IF...THEN

Use la estructura de control IF..THEN para ejecutar una acción específica si una condición es TRUE. Hay dos versiones de la sentencia IF..THEN: la sentencia unitaria en línea y la sentencia múltiple.

- Sentencia unitaria en línea IF...THEN

Sintaxis

IF *condition* THEN *action1* { ELSE *action2* }

Parámetro	Descripción
<i>condition</i>	La condición que se desea testear.
<i>action1</i>	La acción que se ejecutará si la condición es TRUE. La acción debe ser una sola sentencia.
<i>action2</i> (opcional)	La acción que se ejecutará si la condición es FALSE. La acción debe ser una sola sentencia. La acción por defecto es: ejecutar la siguiente sentencia del script.

Ejemplo

IF X + 1 = Y THEN X =0 ELSE X = -1

IF X+1=Y+2 THEN halt

- Sentencia Múlti-Línea IF...THEN.

Sintaxis

```
IF condition1 THEN
    action1
{ELSEIF condition2 THEN
    action2
    ... }
{ELSE
    action3 }
END IF
```

Parámetro	Descripción
<i>condition1</i>	La condición a testear.
<i>action1</i>	La acción que se ejecutará si la condición es TRUE. La acción puede ser una o varias sentencias separadas por punto y coma, o puestas en líneas separadas. Se requiere al menos una acción.
<i>condition2</i> (opcional)	La condición a testear si la condición previa es FALSE. Se pueden tener varios ELSEIF..THEN en una sentencia IF..THEN.
<i>action2</i> (opcional)	La acción que se ejecutará si la segunda condición es TRUE. La acción puede ser una o varias sentencias separadas por punto y coma, o puestas en líneas separadas. Se requiere al menos una acción en cada ELSEIF..THEN
<i>action3</i> (opcional)	La acción que se ejecutará si no se cumple ninguna condición. La acción puede ser una o varias sentencias separadas por punto y coma, o puestas en líneas separadas. La acción por defecto es: ejecutar la siguiente sentencia del script.

### Ejemplos

Esta sentencia compara las posiciones horizontal de WINDOW6 y WINDOW7, y si WINDOW6 está a la derecha de WINDOW7, mueve WINDOW6 a la izquierda de la pantalla.

```
IF WINDOW6.X>WINDOW7.X THEN
    WINDOW6.X=0
END IF
```

La siguiente sentencia causa:

Beep Si X es igual a Y  
Muestra el Edit ListBox Parts y destaca el item 5 si X es igual a Z.  
Muestra el listBox Choose si X es blanco.  
Oculta el button1 y muestra el botón button2 si ninguna condición se cumple.

```
IF X = Y THEN
    BEEP(2)
ELSEIF X=Z THEN
    Show (Parts); SetState(Parts,5,TRUE)
ELSEIF X=" " THEN
    Show (Choose)
ELSE
    HIDE (BUTTON1)
    SHOW(BUTTON2)
END IF
```

## HALT

Use HALT sin palabras claves asociadas, para terminar inmediatamente la aplicación. Use HALT con la palabra clave CLOSE, para cerrar la aplicación antes de terminar la aplicación.

### Sintaxis

HALT {*CLOSE*}

Cuando PowerBuilder encuentra Halt en un script sin la palabra clave *CLOSE*, termina inmediatamente la aplicación.

Cuando PowerBuilder encuentra Halt en un script con la palabra clave *CLOSE*, ejecuta inmediatamente el script del evento Close del objeto aplicación y entonces termina la aplicación. Si en este evento no hay script, PowerBuilder termina inmediatamente la aplicación.

### Ejemplos

En el siguiente script, se termina la aplicación si el usuario entra un password en sle\_password que no coincide con el valor almacenado en el string CorrectPassword:

```
if sle_password.text <> CorrectPassword then HALT
```

La siguiente sentencia ejecuta el script para el evento Close del objeto aplicación antes de terminar la aplicación si el usuario entra un password incorrecto

```
if sle_password.text <> CorrectPassword then HALT close
```



## RETURN

Use RETURN para finalizar la ejecución de un script o función inmediatamente. Cuando se usa RETURN en un script, el script detiene su ejecución y el sistema queda a la espera. Cuando se usa RETURN en una función, el script de la función se detiene y se transfiere el control a la sentencia que sigue a la que llamó la función.

### Sintaxis

En un script:

RETURN

En una función:

RETURN *expression*

Parámetro	Descripción
<i>expression</i>	Cualquier valor o expresión que se desea retorne la función. El valor de retorno debe ser del mismo tipo de dato especificado en el tipo de dato de retorno de la función.

### Ejemplo

El segundo beep no se ejecutará:

```
beep(1)
Return
beep(3)    // Esta sentencia no se ejecutará.
```

La siguiente función retorna el resultado de dividir arg1 por arg2 si arg2  $\neq$  0. Si arg2 = 0, la función retorna -1.

```
If arg2  $\neq$  0 then
    Return arg1/ arg2
else
    Return -1
end if
```

## Funciones PowerScript

La siguiente es una lista de las funciones PowerScript más empleadas.

Se usan las funciones:

- Para obtener información de un objeto, string, o número y retornar la información al programa o la pantalla.
- Cambiar la apariencia o comportamiento de un objeto.
- Manipular o cambiar los datos en un objeto.

Todas las funciones PowerScript retornan valores, usualmente un integer o un string. En las funciones que cambian la apariencia o comportamiento de un objeto, o cambio o manipulación de datos, el valor de retorno indica si la función fue exitosa. Usualmente el retorno es 1 si hubo éxito, y -1 indica error.

Las funciones PowerScript están divididas en los siguientes grupos:

### Arrays

Use esta función	Para obtener
LowerBound	El límite inferior de la dimensión n de un arreglo específico.
UpperBound	El límite superior de la dimensión n de un arreglo específico.

### Funciones de chequeo de tipo de dato y conversión.

Use esta función	Para:
Char	Obtener un blob, integer, o string como un carácter.
Dec	Obtener el contenido de un string como un decimal.
Double	Obtener el contenido de un string como un double.
Integer	Obtener el contenido de un string como un entero.
Long	Obtener el contenido de un string como un long.
Real	Obtener el contenido de un string como un real.
Date	Obtener la porción date (fecha) de un valor datetime traído de la base de datos. Date tiene tres formatos.
Datetime	Obtener un date (fecha) y un time (hora) como un valor datetime.
IsDate	Determina si el string especificado contiene una fecha válida.
IsNull	Determina si el argumento es NULL (nulo).
Isnumber	Determina si el string especificado contiene un número.
IsTime	Determina si el string especificado contiene una hora válida.
String	Obtiene una fecha como string con un formato específico.
Time	Obtiene la porción time (hora) desde un valor DateTime recuperado desde la base de datos. Time tiene tres formatos.

## Funciones de archivo

Las funciones de archivo se usan para abrir, leer y escribir archivos de texto o blob desde PowerBuilder. Los archivos se pueden abrir en line mode (modo línea) o stream mode (modo flujo).

En modo línea, el archivo se lee línea a línea (hasta que se encuentra un CR= Carriage return, un LF= Line Feed o EOF = End of file), y se agrega un CR o LF al final de cada escritura al archivo.

En modo stream, el archivo se lee entero (hasta que se encuentra un EOF ), y no se agrega CR ni LF cuando se escribe al archivo.

**Tip** Usualmente, se usa el modo stream para leer un archivo completo a un MultiLineEdit, y luego se escribe una vez modificado su contenido.

Cuando se abre un archivo, PowerBuilder le asigna un número entero único y setea la posición del puntero del archivo a la posición que se especifique ( al comienzo o final del archivo). El valor asignado, se usa para identificar el archivo cuando se lee, escribe o cierra. La posición del puntero, define donde comenzará la próxima lectura o escritura, y PowerBuilder avanza el puntero automáticamente después de cada lectura o escritura.

Use esta función	para
FileClose	Cerrar un archivo.
FileDelete	Eliminar un archivo.
FileExists	Determina si un archivo existe.
FileLength	Obtiene la longitud de un archivo.
FileOpen	Abre un archivo.
FileRead	Lee un archivo.
FileSeek	Mueve el puntero de un archivo a una posición específica.
FileWrite	Escribe a un archivo.
GetFileOpenName	Muestra caja de diálogo para apertura de archivo.
GetFileSaveName	Muestra caja de diálogo para grabar archivo

## Funciones de librería

Las funciones de librería permiten crear, eliminar, importar y exportar librerías PowerBuilder. También se puede generar una lista de objetos en una librería específica.

Use esta función	Para
LibraryCreate	Crea una librería y le asocia comentarios.
LibraryDelete	Elimina una librería.
LibraryDirectory	Retorna una lista que contiene todos los objetos de un tipo específico.
LibraryExport	Exporta objetos desde una librería específica.
LibraryImport	Importa objetos a una librería específica.

## Funciones numéricas

Use esta función	Para
Abs	Obtener el valor absoluto de un número.
Ceiling	Obtener el menor valor entero que es mayor o igual a un número específico.
Cos	Obtener el coseno de un ángulo. El ángulo está en radianes.
Exp	Obtener e elevado a la potencia x
Fact	Obtener el factorial de x.
Int	Obtener el mayor valor que es menor o igual a un número específico.
Log	Obtener el logaritmo natural (base e) de un número.
logTen	Obtener el logaritmo decimal (base 10) de un número.
Max	Obtener el mayor de dos números.
Min	Obtener el menor valor de dos números.
Mod	Obtener el módulo de dos números (el resto).
Pi	Obtener pi (3.14159265358979323) veces un número.
Rand	Obtener un número aleatorio (entre 1 y un número especificado).
Randomize	Inicializar el generador de números aleatorios.
Round	Obtener un número redondeado a un cierto número de lugares decimales.
Sign	Obtener un número (-1,0,1) que indica el signo de un número.
Sin	Obtener el seno de un ángulo. El ángulo está en radianes.
Sqrt	Obtener la raíz cuadrada de un número.
Tan	Obtener la tangente de un ángulo. El ángulo está en radianes.
Truncate	Obtener un número truncado a un número específico de lugares decimales.

## Funciones de impresión

Las siguientes funciones de impresión se usan para formatear e imprimir listas y reportes. Se llama a la función PrintOpen para comenzar un print job y a PrintClose para finalizar un job. Entre estas dos funciones, se llaman una o más de otras funciones de impresión para especificar lo que se desea imprimir y cómo se desea que luzca la lista o el reporte.

Use esta función	Para
Print	Envía datos a la impresora. Datawindow, objeto visual, texto.
PrintBitMap	Imprime una imagen bitmap en una locación específica en el área de impresión.
PrintCancel	Cancela la impresión.
PrintClose	Cierra el print job y envía la página a la impresora.
PrintDefineFont	Define un font para el print job. PowerBuilder soporta 8 fonts para cada print job.
PrintLine	Imprime una línea de un grosor determinado en una localización específica.
PrintOpen	Comienza el print job y se le asigna un número.
PrintOval	Imprime un ovalo (o círculo) con un grosor específico en una localización determinada.
PrintPage	Envía la página actual a la impresora y setea una nueva página en blanco.
PrintRec	Imprime un rectángulo con un grosor específico en una localización determinada.
PrintRoundRect	Imprime un round rectángulo con un grosor específico en una localización determinada.
PrintSend	Envía un string específico directamente a la impresora.
PrintSetFont	Setea la font del print job
PrintSetSpacing	Setea el factor de espaciado que será usado para determinar el espacio entre líneas.
PrintSetup	Llama la caja de edición de impresión.
PrintText	Imprime texto especificado en una localización determinada.
PrintWidth	Retorna el ancho (en 1/1000s de pulgada) del string especificado.
PrintX	Retorna la coordenada X del cursor.
PrintY	Retorna la coordenada Y del cursor.

## Funciones de String

Use esta función	Para obtener
Asc	El valor ASCII del primer carácter de un string.
Char	El caracter que corresponde a un valor ASCII.
Fill	Un string de una longitud específica llenado con ocurrencias de un string específico.
Left	Un número específico de caracteres de un string, comenzando con el primer carácter.
LeftTrim	Una copia de un string sin espacios a la izquierda.
Len	La longitud de un string.
Lower	Una copia de un string específico con todas sus letras en minúscula.
Mid	Un string conteniendo un número de caracteres copiados (comenzando en una posición específica) desde otro string.
Pos	La posición de comienzo de un string dentro de otro.
Replace	Una copia de un string en el cual un número de caracteres, comenzando en una posición, han sido reemplazados con caracteres de otro string.
Right	Un número de caracteres desde el final de un string.
RightTrim	Una copia de un string sin espacios a la derecha.
Space	Un string de una longitud determinada lleno con espacios.
Trim	Un string sin espacios a la izquierda y derecha.
Upper	Una copia de un string con todas sus letras en mayúsculas.

## Funciones de sistema y medio ambiente

Use esta función	Para obtener
Clipboard	Obtiene el contenido del portapapeles
CommandParm	Trae el string de parámetro, si fue especificado cuando comenzó la aplicación.
DoScript	Corre un script AppleScript. La extensión de systema AppleScript debe estar instalada en un Macintosh.
GetApplication	Obtiene el puntero (handle) del actual objeto aplicación para permitir obtener y setear sus atributos.
GetEnvironment	Obtiene información acerca del sistema operativo, procesador, y monitor del sistema.
Handle	Se usa para hacer llamadas a las funciones del Windows SDK.
Post	Añade un mensaje específico, word o long, al final de la cola de mensaje para la ventana especificada.
ProfileInt	Obtiene un entero desde un archivo de perfil (profile file).
ProfileString	Obtiene un string desde un archivo de perfil (profile file).
Restart	Detiene la ejecución de todos los script, cierra todas las ventanas, hace commit y desconecta de la base de datos, y restaura la aplicación.
Run	Corre o ejecuta un programa específico.
Send	Envía secuencialmente un mensaje, lowword, o long a una ventana.
SetProfileString	Escribe un valor a un archivo de perfil.
ShowHelp	Da acceso al sistema de ayuda que se crea para la aplicación.
SignalError	Provoca un evento SystemError a nivel de la aplicación.
Yield	Chequea la cola de mensajes y opcionalmente pone mensajes en ella.

## Funciones de Timing

Use esta función	Para obtener
CPU	Número de milisegundos de tiempo de CPU que han pasado desde que comenzó la actual aplicación PowerBuilder.
Idle	Gatilla un evento Idle de aplicación después de n segundos de inactividad.
Timer	Gatilla repetidamente un evento timer en una ventana a un intervalo dado.

## Funciones Misceláneas

Use esta función	Para
Beep	Causa un beep un número de veces.
DBHandle	Retorna el puntero (handle) a una DBMS.
IsValid	Determina si una ventana está abierta.
KeyDown	Determina si el usuario presionó una tecla específica.
MessageBox	Muestra una caja conteniendo un mensaje
PixelsToUnits	Convierte pixeles a unidades PowerBuilder.
RGB	Determina el long que representa un color.
SetNull	Setea cualquier variable a NULL.
SetPointer	Setea el puntero a un tipo.
TypeOf	Determina el tipo de un objeto (por ejemplo, Checkbox, Picture, o RadioButton).
UnitsToPixels	Convierte unidades PowerBuilder a pixeles.

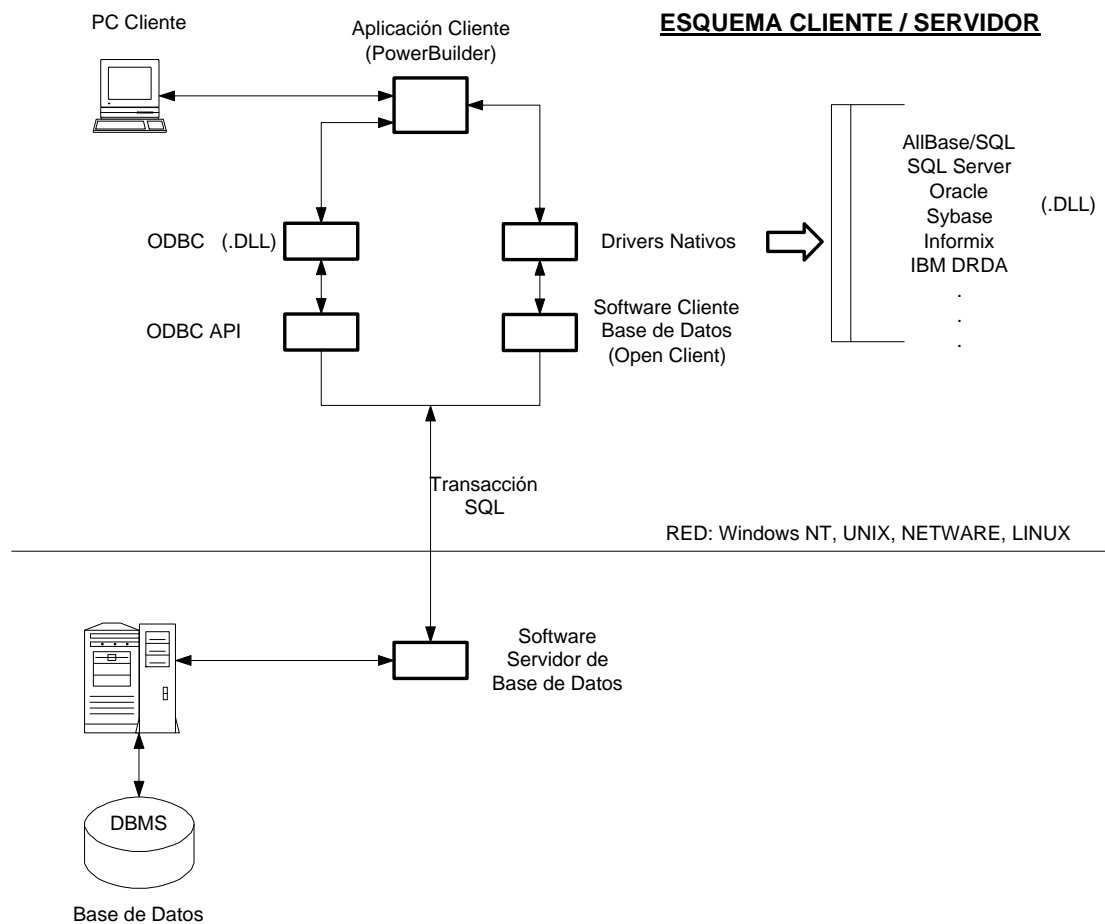
## ***Arquitectura Cliente / Servidor***



## Arquitectura Cliente / Servidor

La idea principal bajo el concepto Cliente / Servidor, es:

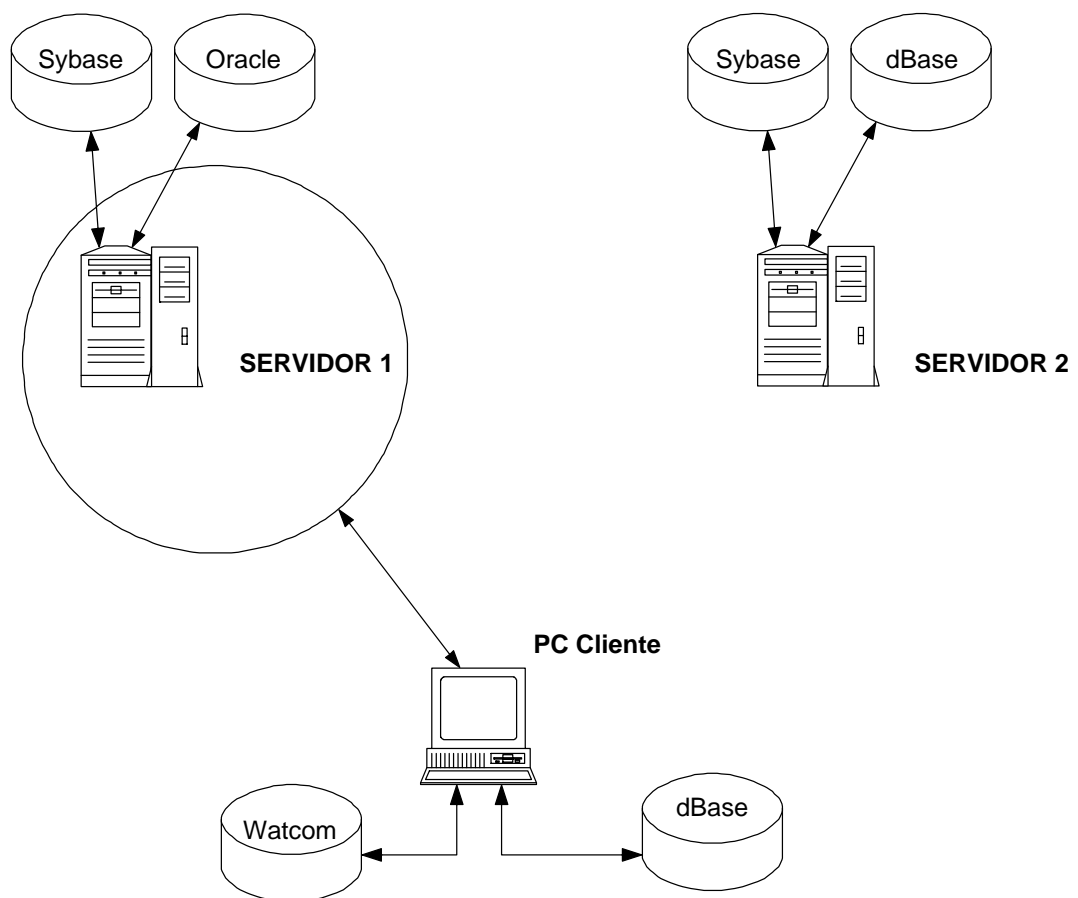
*“Una aplicación Cliente solicita (request) servicios a un Servidor de Base de Datos usando sentencias SQL. El servidor procesa la información recibida y envía al Cliente los datos que solicitó. Una vez que los datos solicitados llegan al cliente, el procesamiento de los mismos toma lugar localmente. De esta forma, los datos están disponibles a muchos usuarios a la vez.”*



## Acceso a los Datos

En un ambiente Cliente / Servidor:

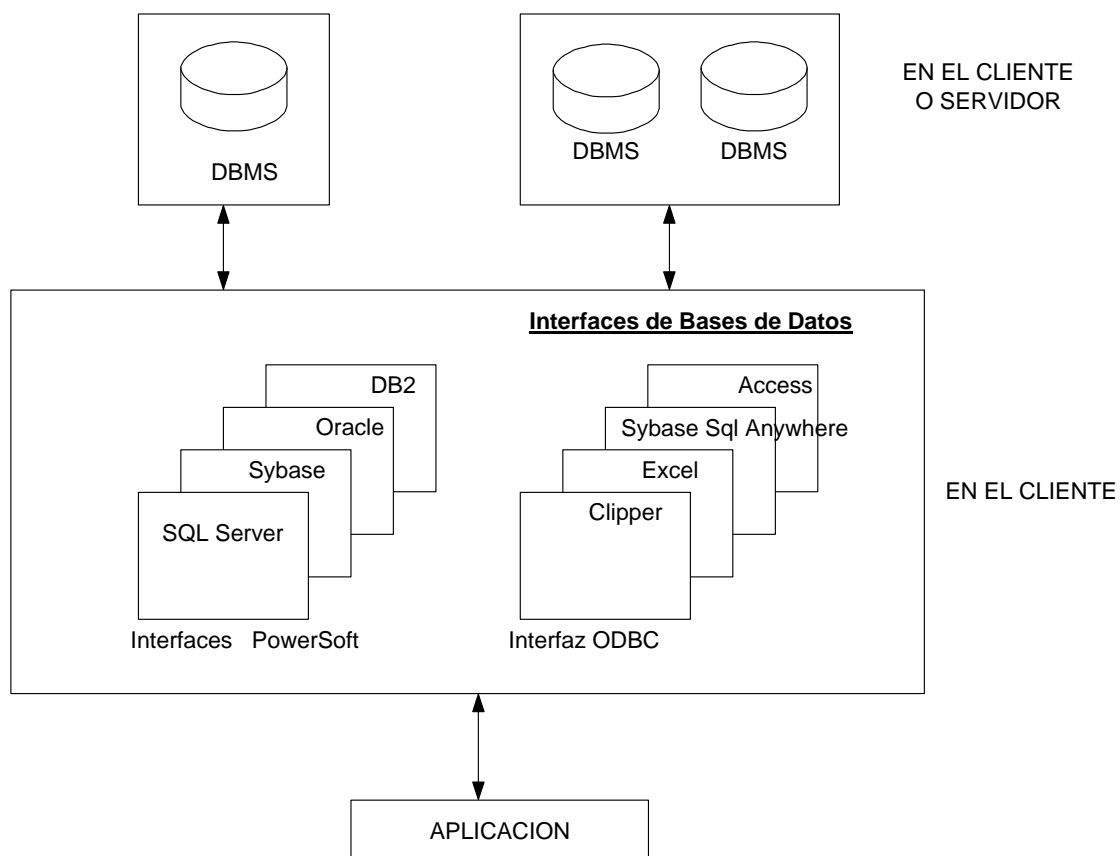
- Las tablas pueden estar almacenadas en una o más bases de datos.
- Las bases de datos pueden estar localizadas en varios lugares: en el computador cliente, en uno o más servidores, o en una mezcla de ellos.
- Las bases de datos pueden estar implementadas bajo una variedad de DBMS (databases management systems).



## Interfaces de Bases de Datos

PowerBuilder permite crear aplicaciones que se comunican con una amplia variedad de DBMSs. Esto lo hace, separando los aspectos específicos de acceso a las bases de datos, del aspecto externo o interfaz de usuario. La aplicación es independiente tanto como sea posible, del acceso a la base de datos que está usando. De esta forma, el desarrollador fija su atención en el uso lógico de una tabla, y no en el cómo esa tabla es accesada en una base de datos particular.

PowerBuilder maneja DBMSs específicos, usando una capa separada de software que se instala en el computador cliente. Esta capa consiste en varias interfaces de bases de datos, cada una de las cuales sabe cómo entenderse con un tipo particular de DBMS, y cómo tomar ventaja de las características únicas de esa DBMS. Cuando una aplicación solicita cualquier tipo de acceso a una base de datos, la solicitud es tomada por la interfaz de base de datos apropiada (dependiendo de la DBMS para esa base de datos) para efectuar la operación.



## Tipos de Interfaces de Datos

Cómo se indicó en la introducción, existen dos tipos de interfaces de bases de datos que PowerBuilder provee para usar con las aplicaciones:

- **La interfaz ODBC**

Se utiliza esta interfaz cuando se crean aplicaciones que accedan bases de datos compatibles con ODBC.

La ODBC API fue desarrollada por Microsoft para dar a las aplicaciones acceso estandarizado a diversas fuentes de datos (data sources).

Estas fuentes de datos son usualmente bases de datos, pero pueden ser también otros tipos de archivos tales como: hojas de cálculo o archivos de texto. La interfaz ODBC incluida con PowerBuilder fue desarrollada por PowerSoft para permitir que las aplicaciones se comuniquen con la ODBC API, la cuál finalmente se entiende con las bases de datos.

- **Interfaces de Base de datos PowerSoft**

Si la aplicación no va a acceder una base de datos particular vía ODBC, o bien la base de datos no es compatible con ODBC, PowerBuilder ofrece una variedad de interfaces nativas, cada una de las cuales sabe como comunicarse a una DBMS específica, tal como SQL Server, Oracle y Oracle.

Así, si se desea acceder una base de datos SQL Server, por ejemplo, se diseña la aplicación para que use la interfaz PowerSoft SQL Server.

Se puede diseñar la aplicación para que use cualquier combinación de estas interfaces de bases de datos.

## Beneficios de las interfaces de bases de datos

El principal beneficio que brinda la capa de software de interfaces de bases de datos, es aislar la aplicación de toda la complejidad interna de manejo de un medio ambiente Cliente/Servidor. Como resultado, el acceso a los datos que se diseña en la aplicación resulta:

- **Flexible**

La aplicación es independiente de la localización física de la base de datos o DBMS. De este modo, si se desea mover la base de datos (desde el cliente al servidor, desde el servidor al cliente, o desde servidor a servidor) o migrar a una DBMS diferente, la aplicación en sí misma no debe sufrir grandes cambios para que pueda seguir funcionando. Esta característica ayuda mucho durante el desarrollo de una aplicación, pues permite cambiar entre una versión producción y una versión de prueba de la base de datos.

Adaptar la aplicación a tales cambios, puede significar tan solo, cambiar el contenido de una variable que apunte a una nueva localización de la base de datos o a una nueva interfaz de base de datos.

- **Consistencia**

Se puede trabajar con todas las tablas de una aplicación de la misma forma, es decir, usando las mismas características de procesamiento de tablas, sin mirar las DBMSs involucradas. Esto significa que no se requiere diseñar rutinas de proceso específicas a una base de datos particular. Incluso, en casos en que se desea aprovechar capacidades únicas de ciertas DBMSs (tales como procedimientos almacenados, uniones externas, chequeos de integridad, etc); aún sólo deben usarse técnicas consistentes de PowerBuilder para hacerlo. Por supuesto, si se planea migrar posteriormente a una DBMS diferente, se debe chequear si soportará tales características especiales, sino, se necesitará modificar la aplicación.

## ***Elementos de una aplicación PowerBuilder***

## *Elementos de una aplicación PowerBuilder*

Cuando se construye una aplicación PowerBuilder, se crea una colección de objetos, los cuales se van almacenando en una o varias librerías.

Las librerías PowerBuilder son archivos con extensión PBL, creadas para almacenar los distintos tipos de objetos que forman una aplicación.

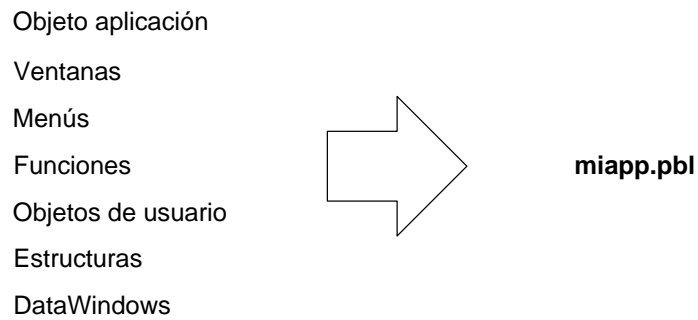
Existen ocho tipos principales de objetos PowerBuilder:

- El objeto aplicación
- Ventanas
- Menús
- Funciones
- Objetos de usuario
- Estructuras
- Datawindows object
- Queries

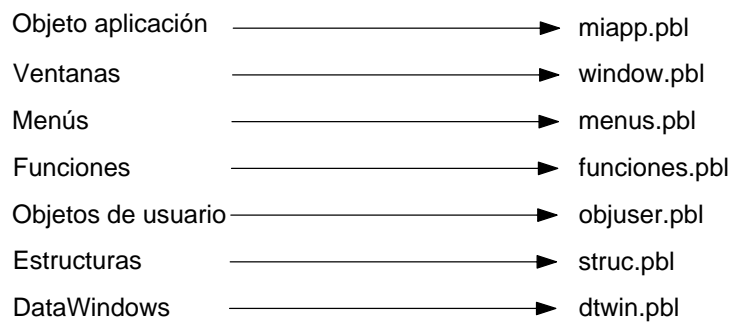
Los objetos de una aplicación pueden agruparse en una única librería si la aplicación es pequeña y tiene pocos elementos. Sin embargo, para aplicaciones avanzadas, con gran cantidad de objetos, es recomendable crear otras librerías que agrupen los elementos por tema, por tipos de objetos u otra forma que sea útil a efectos de facilitar un orden durante su diseño.

Ejemplos:

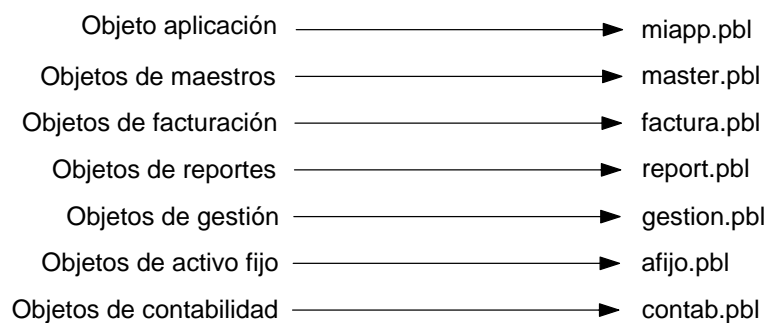
#### Todos los objetos en una librería



#### Objetos en librerías según tipo



#### Objetos en librerías según temas

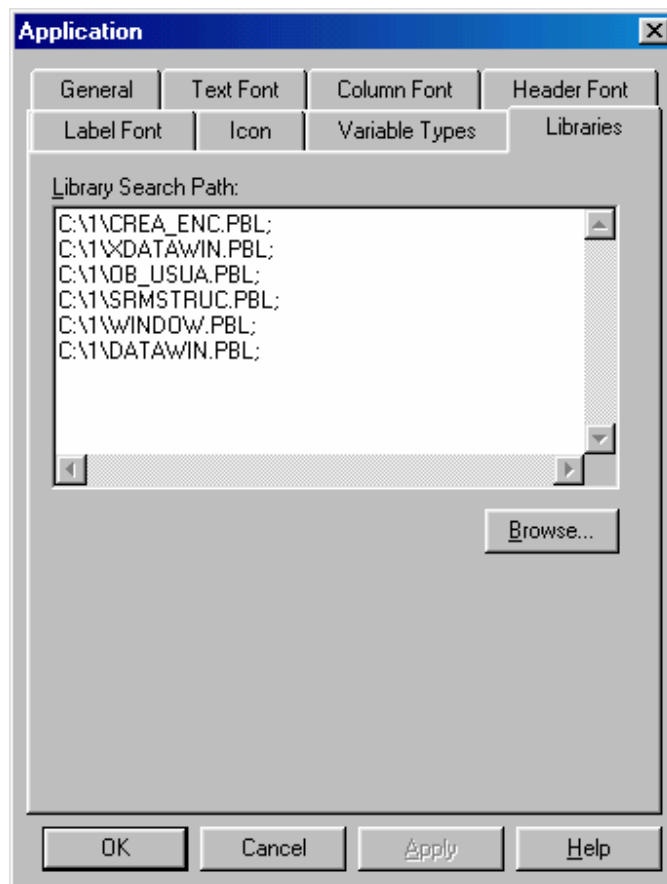




## Library Search Path

El 'library search path', es una lista de las librerías que participan en la aplicación; y donde están almacenados los distintos tipos de objetos que la conforman. Los nombres de librería están separados por punto y coma.

PowerBuilder usa esta lista para encontrar los objetos referenciados en la aplicación.



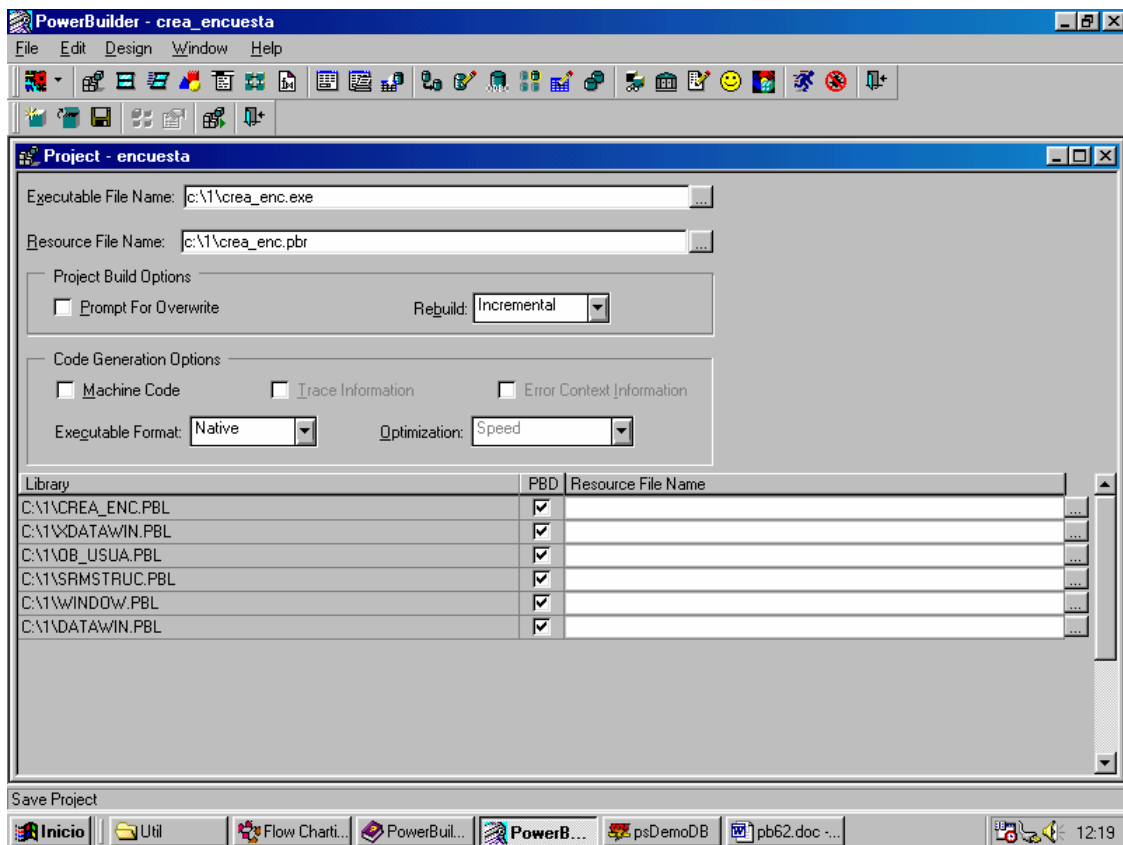
### Nota

Para eliminar una librería del search path, seleccione la librería de la lista en el listbox del Library Search Path y presione Del.

Se pueden incluir librerías desde diferentes drives y directorios en el search path.

## Proyecto de la aplicación

Cuando la aplicación cuenta con todos los objetos necesarios para cumplir con los requerimientos fijados en el diseño, y funciona sin errores al correrla en el ambiente PowerBuilder, puede crearse un '**proyecto**' para generar el programa ejecutable (EXE) de la aplicación. Este podrá correr por sí mismo en el ambiente Windows.



## Estructura de una aplicación PowerBuilder

En la esencia de una aplicación PowerBuilder se distinguen los siguientes elementos básicos:

- **Una conexión a una base de datos**  
Mediante un **objeto transacción**, la aplicación se conecta a una base de datos.

- **Una interfaz gráfica y objetos**

**Objeto Aplicación** que da el inicio a la aplicación

**Menús**

**Ventanas** y sus controles

DataWindow control (usa el objeto transacción) + DataWindow Object

Button

RadioButton

CheckBox

EditMask

SingleLineEdit

StaticText

Gráficos

MemoLineEdit

ListBox

DropDownListBox

PictureButton

TreeView

Otros

**Estructuras**

**Funciones**

**Objetos de usuario**









**Objeto Proyecto** para la creación del ejecutable

- **Lenguaje PowerScript**

Para efectuar procesamiento

## Construcción de la primera aplicación PowerBuilder

Antes de comenzar a construir la aplicación, asegurar que exista conexión a la base de datos. Luego, siga los pasos listados para crear la primera aplicación.

-  Cree una tabla en la base de datos. Este paso se requiere sólo si la base de datos no contiene los datos que se desea acceder desde la aplicación.
-  Construya un objeto aplicación.
-  Construya una ventana. Ponga controles en la ventana para comunicarse con el usuario. Dando click a un control o seleccionando un ítem de una lista, el usuario puede iniciar una actividad, responder o requerir desde la aplicación, traer datos de una base de datos o actualizarla. Las ventanas con sus controles constituyen el lugar donde el usuario se comunica mayormente con la aplicación.
-  Construya un DataWindow Object para presentar, manipular y actualizar información de una base de datos relacional u otra fuente de datos.
-  Ponga un DataWindow control en la ventana, asócielo un DataWindow object y construya scripts para conectarse a la base de datos, traer, cambiar y actualizar los datos.
-  Construya un menú para la ventana. El menú debiera listar las opciones disponibles para la ventana
-  Asocie el menú con la ventana.
-  Construya un proyecto y cree el programa ejecutable.

Luego de completar estos pasos, se tiene una aplicación simple PowerBuilder. Se puede usar o mejorarla utilizando otros pintores y características PowerBuilder.

## **Conexión y objeto transacción**

## *Conexión y objeto transacción*

Generalmente al iniciar la aplicación, en el evento Open del objeto aplicación se realiza la conexión a la base de datos. Esta consiste de setear ciertos parámetros del objeto transacción, y luego ejecutar la sentencia connect para conectar con la base de datos. Luego, cuando debe emplearse un DataWindow para efectuar alguna operación en la base de datos, se le dice qué objeto transacción debe usar. Esto se debe, a que una aplicación puede tener varios objetos transacción para conexión a distintas bases de datos. Usualmente, esto se realiza en el evento Constructor del DataWindow Control.

### **Objeto transacción**

Los objetos transacción PowerBuilder son el puente de comunicación entre un script y la base de datos. Cada objeto transacción tiene 15 campos de datos o atributos; diez se usan para conectar con la base de datos, y cinco para recibir información desde la base de datos acerca del éxito o falla de cada operación.

Cuando se construye una nueva aplicación, automáticamente se tiene un objeto transacción por defecto llamado SQLCA. Si la aplicación lo requiere, pueden crearse otros objetos transacción, usando la sentencia CREATE.

Antes de usar el objeto transacción por defecto (SQLCA) u otro objeto transacción, se debe asignar valores a los atributos que se usarán. Para asignar un valor a un atributo , use las sentencias de asignación y la notación punto PowerBuilder.

Los atributos del objeto transacción se describen a continuación.

Atributo	Tipo de dato	Descripción
DBMS	String	El nombre del tipo de base de datos (database vendor). Por ejemplo, ODBC, Sybase, ORACLE).
Database	String	El nombre de la base de datos.
UserId	String	El nombre o ID del usuario en la base de datos.
DBParm	String	Específico de cada DBMS.
DBPass	String	El password que se usará para conectar con la base de datos.
Lock	String	El nivel de aislación.
LogId	String	El nombre o ID del usuario que accesará (log on) el servidor.
LogPass	String	El password usado para logon on al servidor.
ServerName	String	El nombre del servidor en que reside la base de datos.
AutoCommit	Boolean	Indicador de commit automático. <ul style="list-style-type: none"> <li>- TRUE. Hace commit automáticamente después de cada actividad en la base de datos.</li> <li>- FALSE (default). No hace commit automáticamente.</li> </ul> <p><b>Tip.</b> Autocommit debe estar a TRUE para crear tablas temporales.</p>
SQLCode	Long	Indica el éxito o falla de la más reciente operación. <ul style="list-style-type: none"> <li>- 0. Indica éxito.</li> <li>- 100. Indica sin retorno de conjunto resultado.</li> <li>- -1. Indica error (use SQLDBCode o SQLErrMsgText para obtener más detalle)</li> </ul>
SQLNRows	Long	El número de filas afectadas. El database vendor entrega este valor, sin embargo el significado puede no ser el mismo en cada DBMS.
SQLDBCode	Long	Código de error del database vendor.
SQLErrMsgText	String	El mensaje de error del database vendor.
SQLReturnData	String	Específico del database vendor.

## Ejemplo de conexión a una base de datos Sybase SQLAnywhere.


```

sqlca.dbms      = "ODBC"
sqlca.database  = "encuesta.db"
sqlca.userid    = "dba"
sqlca.dbpass    = "sql"
sqlca.dbparm    = "ConnectString='DSN=Encuesta;UID=dba;PWD=sql'"

connect;

// Comprobar conexión
If sqlca.sqlcode <> 0 then
    MessageBox ("Error en conexión", &
        "No puede efectuarse conexión a base de datos.~n~r~n~r" + & sqlca.sqlerrtext )
    halt close
    return
End if

```

En este caso, **Encuesta** es el nombre de la fuente de datos que se definió previamente en Configure ODBC .

## Ejemplo de conexión a una base de datos Sybase.

```

sqlca.DBMS      = "SYC Sybase System 10"
sqlca.ServerName = "Sybase_NT"
sqlca.Database   = "operaciones"
sqlca.UserID     = "dbo_ope"
sqlca.Dbpass     = "centurion"

sqlca.LogId      = "opera"
sqlca.Logpass    = "operaciones"

sqlca.DbParm     = "appname='Patio Valpo.', host='enlace' "
sqlca.autocommit = False

// Comprobar error
Connect Using Sqlca;
If (Sqlca.sqlcode <> 0) Then
    MessageBox("Error en la Conexión",Sqlca.Sqlerrtext,StopSign!)
    Halt Close
End If

```



## Archivo de inicialización (.INI)

En los dos casos anteriores, los valores de los atributos fueron seteados ‘en duro’, es decir, como parte del código. Esto puede ser poco práctico a la hora de querer cambiar algún atributo cuando la aplicación ya se distribuyó en modo ejecutable.

Para evitar este inconveniente, se opta por crear un archivo de inicialización de la aplicación. En este archivo, se pone una “sección” que contenga los atributos de conexión a la base de datos, los cuales podrán cambiarse con cualquier editor de texto cuando se requiera.

La estructura para el primer ejemplo, es:

### Encuesta.ini

```
[sqlca]
dbms=ODBC
database=encuesta.db
userid=dba
dbpass=sql
DbParm=ConnectionString='DSN=Encuesta;UID=dba;PWD=sql'
```

y la forma de leerlo desde la aplicación:

```
sqlca.dbms      = ProfileString("c:\windows\encuesta.ini","sqlca","dbms","")
sqlca.database  = ProfileString("c:\windows\encuesta.ini","sqlca","database","")
sqlca.userid    = ProfileString("c:\windows\encuesta.ini","sqlca","userid","")
sqlca.dbpass    = ProfileString("c:\windows\encuesta.ini","sqlca","dbpass","")
sqlca.dbparm    = ProfileString("c:\windows\encuesta.ini","sqlca","dbparm","")
```

La estructura para el segundo ejemplo, es:

### Operación.ini

```
[Database]
DBMS=SYC Sybase System 10
ServerName=Sybase_NT
Database=operaciones
UserID=dbo_ope
DbPass=centurion
LogId=opera
```

---

```
LogPass=operaciones
DbParm=appname='Patio Valpo.',host='enlace'
AutoCommit=FALSE
```

y la forma de leerlo desde la aplicación:

String ArchIni

ArchIni = "opera.ini"

```
Sqlca.DBMS           = ProfileString(ArchIni,"database","dbms","SYC Sybase System 10")
Sqlca.ServerName     = ProfileString(ArchIni,"database","servername","Sybase_NT")
Sqlca.Database       = ProfileString(ArchIni,"database","database","operaciones")

Sqlca.UserID         = ProfileString(ArchIni,"database","Userid","dbo_ope")
Sqlca.Dbpass         = ProfileString(ArchIni,"database","dbpass","centurion")

Sqlca.LogId          = ProfileString(ArchIni,"database","Logid","opera")
Sqlca.Logpass        = ProfileString(ArchIni,"database","Logpass","operaciones")
Sqlca.DbParm         = ProfileString(ArchIni,"database","Dbparm","")
```


### Tips

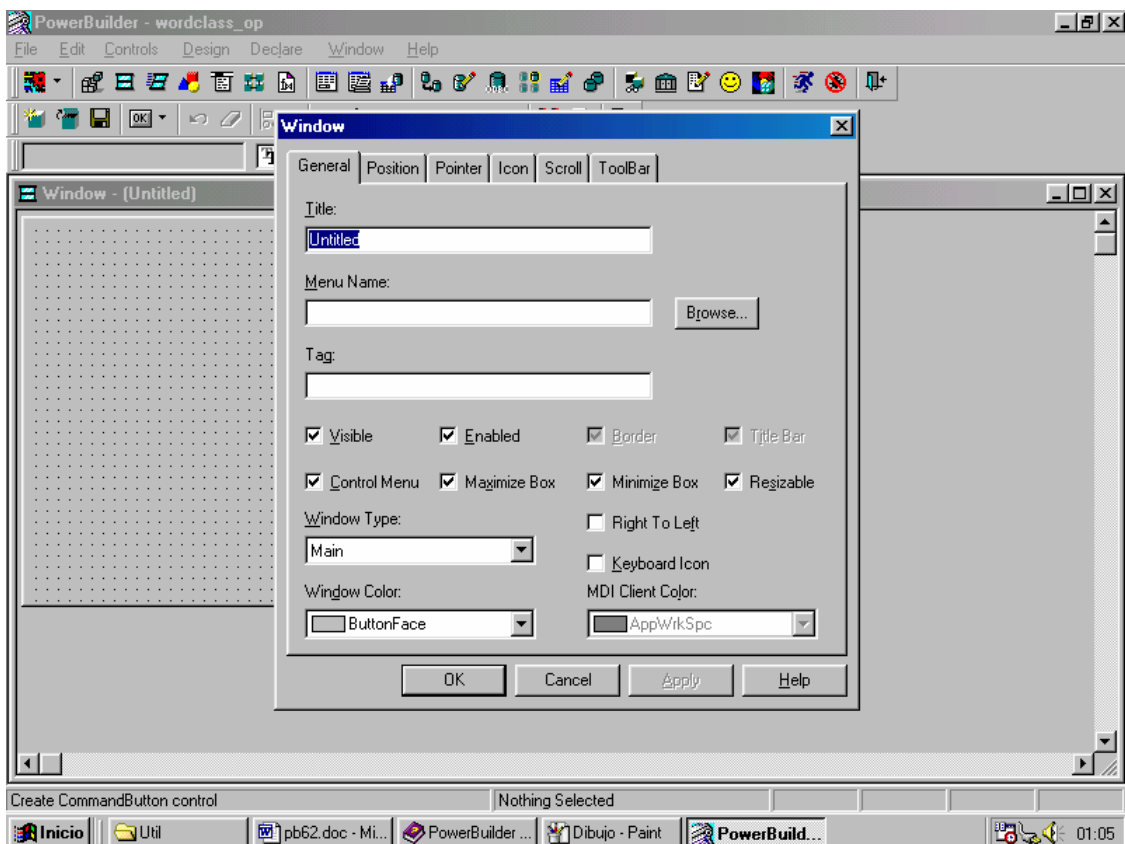
- El archivo ini puede incorporar otras secciones para uso de la aplicación, por ejemplo, para guardar las últimas preferencias usadas en algún proceso, de modo tal que la próxima vez que inicie la aplicación, estas preferencias sean las por defecto.
- Use ProfileInt para obtener un valor integer desde un archivo ini.
- Para grabar valores en las secciones, use la función SetProfileString.

## ***Ventanas y sus controles***

## Ventanas y sus controles

Las ventanas son la interfáz entre el usuario y una aplicación PowerBuilder. Se usan para desplegar información, requerir información del usuario, y responder a acciones del mouse o el teclado.

Las ventanas se construyen con el pintor de ventanas . En este pintor se define la apariencia, estilo (style), tipo (type), tamaño (size), y posición de la ventana.



## Controles en la ventana

Los controles en la ventana son todos los objetos que se pueden poner en una ventana, es decir, controles (objetos en los cuales pueden ocurrir eventos) y objetos de dibujo (objetos sin eventos) tales como líneas, rectángulos, círculos.

En una ventana los controles (ej. CheckBox, CommandButton, Line Edit), permiten a la aplicación requerir y recibir información del usuario. Los objetos de dibujo sirven para mejorar el aspecto de la ventana. Una vez que se pone un objeto en una ventana, se puede especificar su estilo, tamaño, posición y crear script para sus eventos.

Cada ventana y control tiene un estilo que determina como luce al usuario. Los valores de los atributos de la ventana o control determinan el estilo.

Los scripts puestos en la ventana y en los controles forman el comportamiento que tendrá la ventana. Los scripts controlan la acción iniciada cuando ocurre un evento en la ventana o en los controles.

Ejemplo.

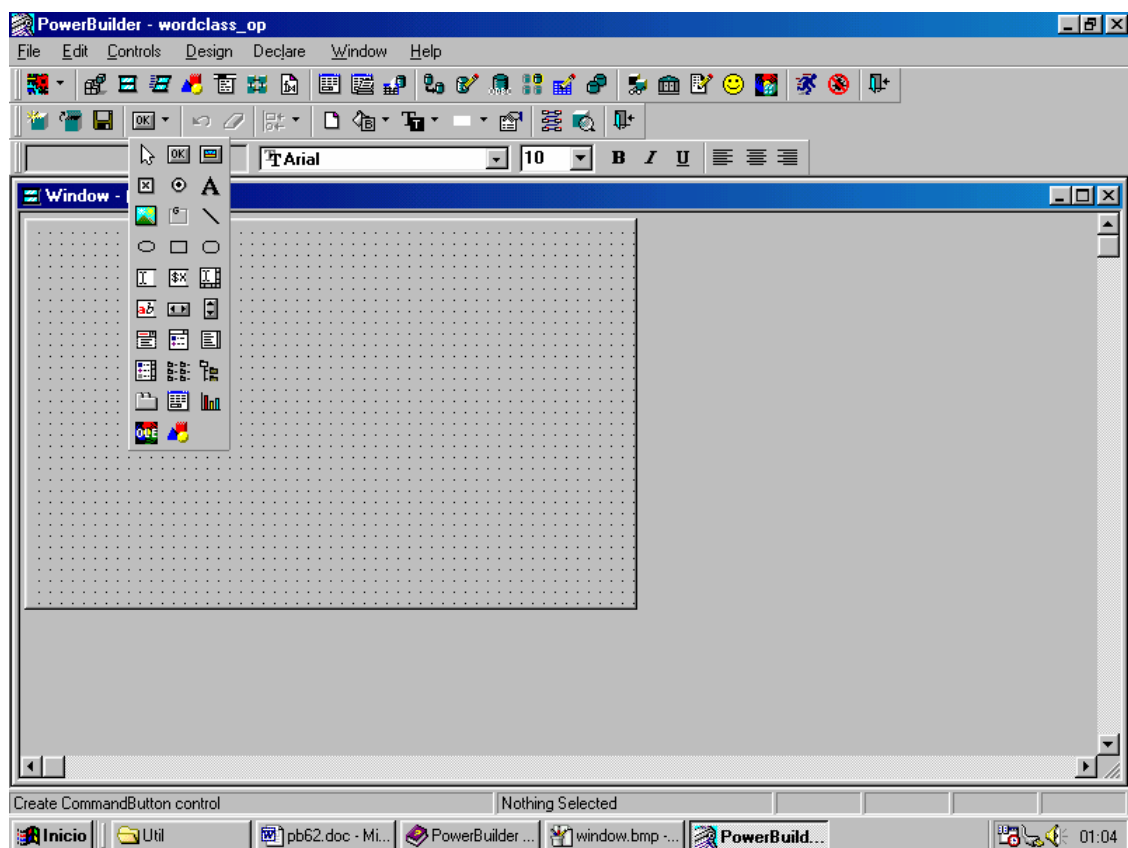
The screenshot shows a Windows-style dialog box titled "Modificar Personal". It contains two main sections of controls. The left section contains text labels and input fields for personal information: "RUT:" with a value of "1-9", "Ap. Paterno:" with "PEREZ", "Ap. Materno:" with "CARVAJAL", "Nombres:" with "CAROLINA", "Nivel:" with "E", "Cargo:" with "JEFE DEPTO. PLANIFIC Y GESTIO", "Fono1:" with "894736", "Fono2:" (empty), "Tipo usuario:" with a dropdown menu showing "Personal", "Viático Hac. (\$):" (empty), and "Viático Inter. (\$):" (empty). The right section is titled "CONTRATO" and contains: "Ubicación:" with a dropdown menu showing "VALPARAISO", "Unidad Negocio:" with a dropdown menu showing "ADMINISTRACION CENTRAL", "Unidad Tecnica:" with a dropdown menu showing "DEPARTAMENTO PLANIF. Y G", "Tipo personal:" with a dropdown menu showing "IFOP", "Fecha:" with a date field showing "00/00/0000", and "Activo:" with a checked checkbox. At the bottom right is a "Password:" label and an empty text field. At the bottom center are two buttons: "Aceptar" and "Cancelar".

## Controles frecuentemente usados en ventanas

La siguiente es una lista de los controles que se utilizan con mayor frecuencia en las ventanas.

Cada control tiene atributos, eventos y funciones. En la lista se seleccionan los atributos, eventos y funciones que se usan con mayor frecuencia.

La figura muestra como acceder a los controles que pueden ser puestos en las ventanas.



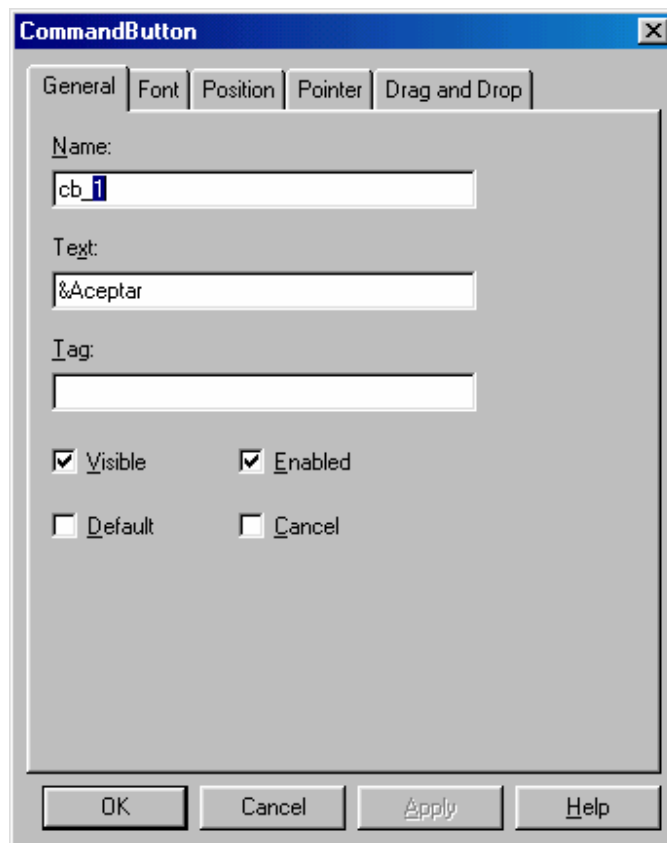
## Command Button

Los Command Button o botones, se usan para llevar a cabo alguna acción. Por ejemplo, usar un botón 'Aceptar' para confirmar una eliminación, o un botón 'Cancelar' para cancelar una eliminación.

Ej. 

## Atributos

Para acceder a los atributos del botón dé dobleclick sobre él.



Atributo	Tipo de dato	descripción
Cancel	Boolean	Especifica si el botón actúa como Cancel button, es decir, si recibe un clic cuando el usuario presione ESC. Los valores son TRUE y FALSE.
Default	Boolean	Especifica si el botón es el control por defecto, es decir, si recibe un click cuando el usuario presiona ENTER. Los valores son TRUE y FALSE.
Enabled	Boolean	Especifica si el botón está habilitado (puede recibir un click). Los valores son TRUE y FALSE.
Text	String	Especifica el texto que se muestra en el botón.

Algunos atributos pueden setearse en tiempo de diseño y también en tiempo de ejecución.

Ej. Para deshabilitar el botón 'Grabar' de nombre cb\_grabar.

```
Cb_grabar.enabled = False
```

Ej. Para cambiar el texto de un botón.

```
If cb_accion.text = 'Seleccionar' Then  
    Cb_accion.text = 'Deseleccionar'  
Else  
    Cb_accion.text = 'Seleccionar'  
End If
```

## **Eventos**

Para poner código en los eventos del botón haga click – derecho sobre él. Seleccione 'script...'

El evento que más se usa en los botones, es el evento Clicked. Se gatilla al dar un click de mouse sobre él, o cuando se tiene en el foco y se da barra de espacio.

## **CheckBox**

Las checkBox son pequeñas cajas cuadradas usadas para setear opciones individuales. Cuando están seleccionadas, contienen una 'X', y cuando no están seleccionadas están vacías.

Ya que las checkbox son independientes una de otra, se las puede agrupar sin afectar su comportamiento. Los grupos de checkbox hacen más fácil al usuario entender su uso.

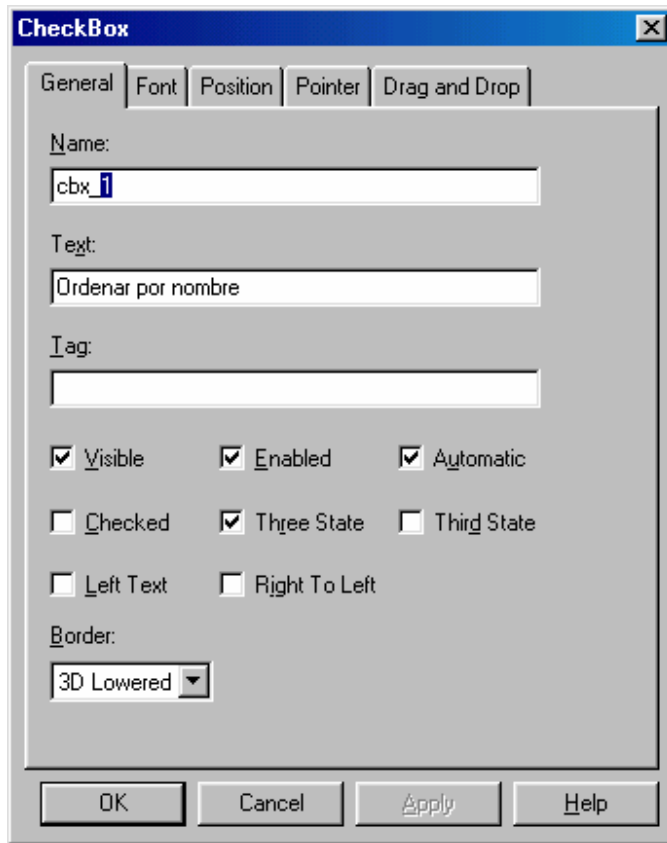
Normalmente las checkbox tienen dos estados: On y Off. Sin embargo se puede utilizar un tercer estado para representar un caso desconocido o no esperado.

Ej. ☐ Ordenar por nombre

## **Atributos**

Para acceder a los atributos del checkbox dé dobleclick sobre él.





Atributo	Tipo de dato	Descripción
Checked	Boolean	Especifica si el checkbox está seleccionado. TRUE, FALSE.
Enabled	Boolean	Especifica si el checkbox está habilitado (puede recibir un click). TRUE, FALSE.
LeftText	Boolean	Especifica si el texto se muestra a la izquierda del control: TRUE ,FALSE.
Text	String	Especifica el texto que muestra el checkbox.

Algunos atributos pueden setearse en tiempo de diseño y también en tiempo de ejecución.

Ej. Para deshabilitar el checkbox 'Usar sólo mayúsculas' de nombre cbx\_upper.

```
cbx_upper.enabled = False
```

Ej. Para obtener el estado de un checkbox.

```
If cbx_todos.Checked Then
```

```
.....
```

```
End If
```

O bien,

Boolean bTodos

bTodos = cbx\_todos.checked

If bTodos Then

...

End If

## **Eventos**

Para poner código en los eventos del checkbox haga click – derecho sobre él. Seleccione ‘script...’

El evento que más se usa en los checkbox, es el evento Clicked. Se gatilla al dar un click de mouse sobre él cambiando su estado.

## **StaticText**

Muestra texto que puede seleccionarse, pero que no se puede modificar. Es empleado para títulos y etiquetas.

## **RadioButton**

Un RadioButton es un pequeño botón redondo usado para cambiar una opción a On/Off. Cuando está On, el botón tiene el centro sombreado. Cuando está Off, el centro está blanco. Los radiobutton a menudo se agrupan usando un GroupBox.

Ej.

☐ Amarillo

☒ Rojo

☐ Verde

## **Nota**

---

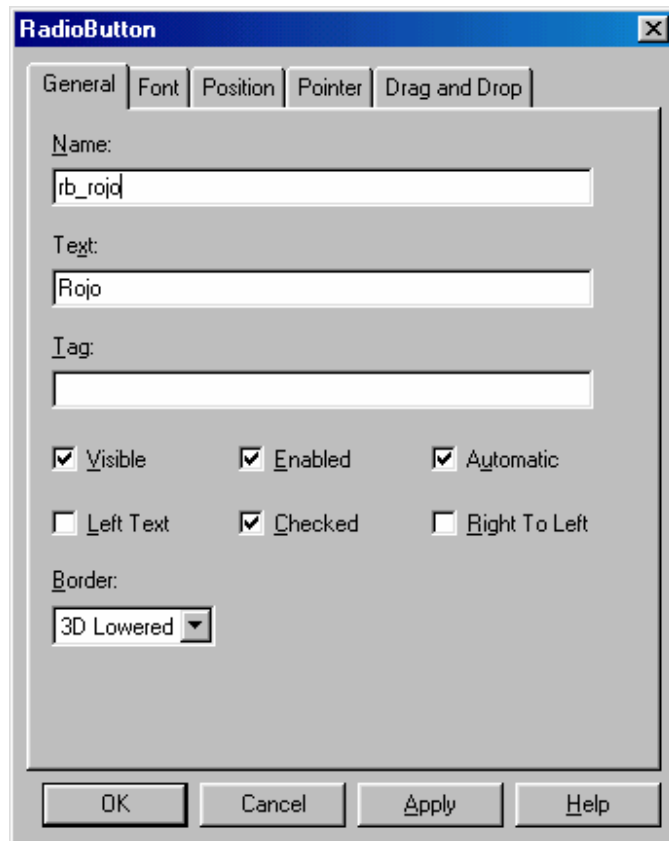
Cuando los radiobutton están agrupados, el usuario puede seleccionar sólo uno de ellos en el grupo, y el grupo usualmente tiene un radiobutton en On por defecto.

---

Se pueden tener varios grupos de radiobutton y cada grupo actúa por separado.

## Atributos

Para acceder a los atributos de un radiobutton dé dobleclick sobre él.



Atributo	Tipo de dato	Descripción
Checked	Boolean	Especifica si el radiobutton está seleccionado. (centro sombreado) TRUE, FALSE.
Enabled	Boolean	Especifica si el checkbox está habilitado (puede seleccionarse). TRUE, FALSE.
LeftText	Boolean	Especifica si el texto se muestra a la izquierda del control: TRUE ,FALSE.
Text	String	Especifica el texto que muestra el radiobutton.

Algunos atributos pueden setearse en tiempo de diseño y también en tiempo de ejecución.

Ej. Para obtener el estado de un radiobutton.

```
If rb_rojo.Checked Then  
.....  
End If
```

## **Eventos**

Para poner código en los eventos del radiobutton haga click – derecho sobre él. Seleccione ‘script...’

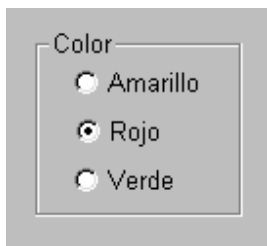
El evento que más se usa en los radiobutton, es el evento Clicked. Se gatilla al dar un click de mouse sobre él cambiando su estado.

## **GroupBox**

Un GroupBox es una caja para agrupar controles relacionados. Por ejemplo, se puede usar un GroupBox para agrupar una serie de RadioButtons o ComandButtons. El usuario no puede seleccionar el GroupBox, pero puede seleccionar los controles dentro de él. Si el GroupBox contiene RadioButtons, el usuario puede seleccionar sólo un radiobutton a la vez.

Un GroupBox no tiene eventos.

Ej.



## SingleLineEdit

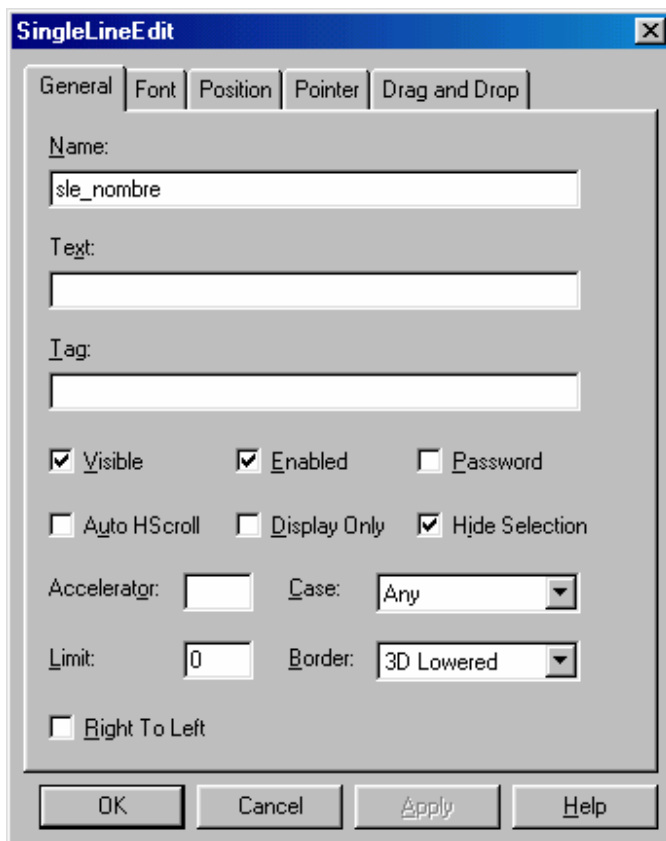
Un SingleLineEdit es una caja donde el usuario puede digitar una línea de texto. Normalmente se usa como campo de ingreso de datos.

Ej.



## Atributos

Para acceder a los atributos de un SingleLineEdit dé dobleclick sobre él.



Atributo	Tipo de dato	Descripción
AutoHScroll	Boolean	Especifica si el control automáticamente hace scroll horizontal cuando el dato es entrado o eliminado. TRUE, FALSE.
BorderStyle	BorderStyle	Especifica el estilo del borde del control. Los valores son: StyleBox! StyleLowered! StyleRaised! StyleShadowBox!
DisplayOnly	Boolean	Especifica si el texto en el control es display-only y no puede ser cambiado por el usuario. Los valores son: - True – El texto no puede ser cambiado por el usuario. - False – El texto puede ser cambiado.
Enabled	Boolean	Especifica si el control está habilitado. TRUE, FALSE
Password	Boolean	Especifica si el control es un campo de password y el texto entrado aparece como asteriscos. TRUE, FALSE.
Text	String	El texto que tiene el control.
TextCase	TexCase	Especifica mayúsculas, minúsculas o cualquiera. Los valores son: - Anycase! - Lower! - Upper!

Algunos atributos pueden setearse en tiempo de diseño y también en tiempo de ejecución.

Ej. Para obtener el texto escrito en sle\_nombre.

String cNombre

cNombre = sle\_nombre.text

Ej. Cambiar a DisplayOnly

Sle\_nombre.DisplayOnly = TRUE

## **Eventos**

Para poner código en los eventos del singleLineEdit haga click – derecho sobre él. Seleccione ‘script...’

Evento	Ocurre
GetFocus	Justo antes que el control reciba el foco (antes que se seleccione y pase a ser activo).
LoseFocus	Cuando el control pierde el foco.
Modified	Cuando el texto en el control ha cambiado y el usuario presiona ENTER o TAB o cambia el foco a otro control.
RbuttonDown	Cuando un click-derecho del mouse se da en el control.

Ej. Evento Modified

En este ejemplo, el código en el evento Modified ejecuta validación del texto que el usuario entra en el SingleLineEdit sle\_color. Si el usuario no entra RED, WHITE o BLUE, un mensaje le dice qué entradas son válidas. Al validar Ok los textos, el color de los textos cambia.

Este no es ejemplo muy realista, pero sirve para ilustrar cuando se gatilla el evento Modified.

String ls\_color

This.BackColor = RGB(150,150,150)

ls\_color = Upper(This.Text)

CHOOSE CASE ls\_color

    CASE "RED"

        This.TextColor = RGB(255,0,0)

    CASE "BLUE"

        This.TextColor = RGB(0,0,255)

    CASE "WHITE"

        This.TextColor = RGB(255,255,255)

    CASE ELSE

        This.Text = ""

        MessageBox("Entrada no válida", &

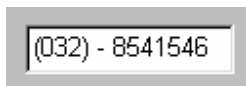
            "Ingrese RED, WHITE, o BLUE.")

END CHOOSE

## **EditMask**

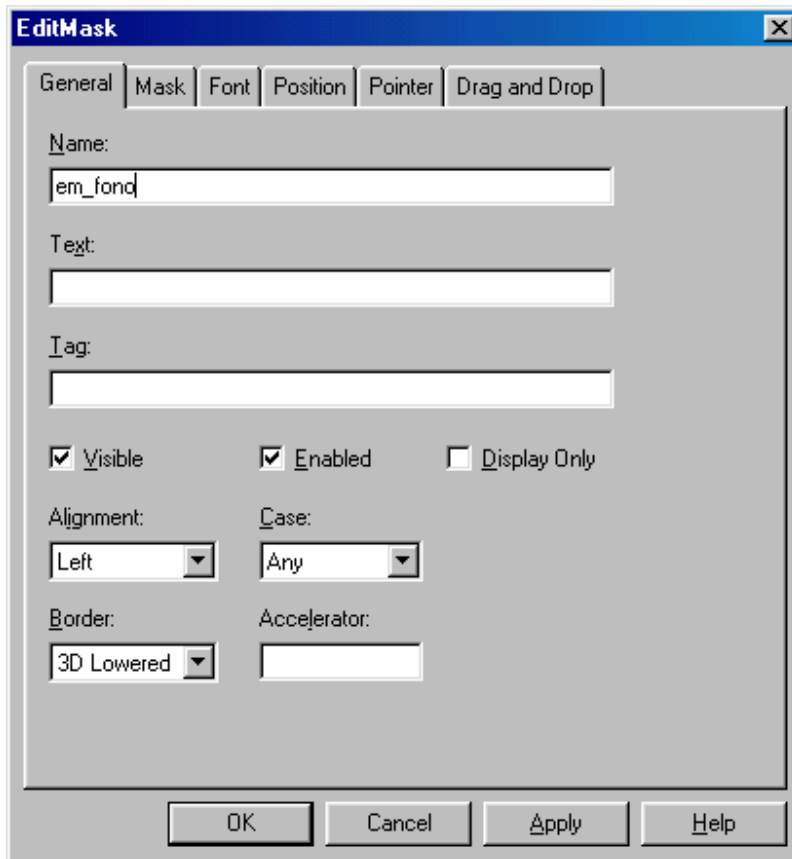
Un EditMask es una caja similar al SingleLineEdit, en la cual el usuario puede entrar y editar una línea de texto. El tipo, apariencia y número de caracteres está restringido por el edit mask. Por ejemplo, se puede usar un EditMask para entrar números telefónicos o fechas formateadas automáticamente cuando el usuario ingresa datos al EditMask.

Ej.



## Atributos

Para acceder a los atributos de un EditMask dé dobleclick sobre él.



Atributo	Tipo de dato	Descripción
Alignment	Alignment	Especifica la alineación del texto en el control. Los valores son: Center! Justify! Left! Right!
AutoHScroll	Boolean	Especifica si PowerBuilder automáticamente hace scroll a la izquierda o derecha cuando los datos son entrados o eliminados. TRUE, FALSE.
AutoSkip	Boolean	Especifica si salta al próximo control cuando se ingresa el último carácter en el control. TRUE, FALSE
Mask	String	Especifica la máscara usada para formatear y editar datos en el control.
MaskDataType!	MaskDataType!	Especifica el tipo de dato del control. Los valores son: DateMask! DateTimeMask! DecimalMask! NumericMask! StringMask! TimeMask!
Text	String	Especifica el texto en el control.



Ej. Para obtener el texto escrito en em\_fono.

String cFono

cFono = em\_fono.text

## **Eventos**

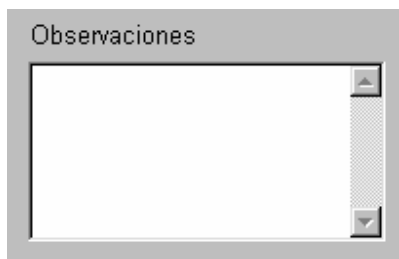
Para poner código en los eventos del EditMask haga click – derecho sobre él. Seleccione ‘script...’

Evento	Ocorre
GetFocus	Justo antes que el control reciba el foco (antes que se seleccione y pase a ser activo).
LoseFocus	Cuando el control pierde el foco.
Modified	Cuando el texto en el control ha cambiado y el usuario presiona ENTER o TAB o cambia el foco a otro control.
RbuttonDown	Cuando un click-derecho del mouse se da en el control.

## **MultiLineEdit**

Un MultiLineEdit es una caja donde el usuario puede entrar y editar más de una línea de texto.

Ej.



Atributos

Para acceder a los atributos de un MultiLineEdit dé dobleclick sobre él.

Atributo	Tipo de dato	Descripción
Alignment	Alignment	Especifica la alineación del texto en el control. Los valores son: Center! Justify! Left! Right!
AutoHScroll	Boolean	Especifica si PowerBuilder automáticamente hace scroll a la izquierda o derecha cuando los datos son entrados o eliminados. TRUE, FALSE.
AutoVScroll	Boolean	Especifica si el control automáticamente hace scroll verticalmente cuando se ingresa o elimina texto. TRUE, FALSE.
Enabled	Boolean	Especifica si el control está habilitado. TRUE,FALSE.
Limit	Integer	Especifica el máximo número de caracteres que pueden ser entrados (0 a 32767) en el control. 0 significa ilimitado.
Text	String	Especifica el texto en el control.

Ej. Para obtener el texto escrito en mle\_observacion

String cObs

cObs = mle\_observacion.text

Eventos

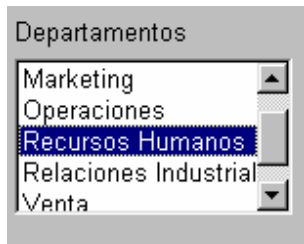
Para poner código en los eventos del EditMask haga click – derecho sobre él. Seleccione ‘script...’

Evento	Ocorre
GetFocus	Justo antes que el control reciba el foco (antes que se seleccione y pase a ser activo).
LoseFocus	Cuando el control pierde el foco.
Modified	Cuando el texto en el control ha cambiado y el usuario presiona ENTER o TAB o cambia el foco a otro control.
RbuttonDown	Cuando un click-derecho del mouse se da en el control.

## ListBox

Un ListBox despliega opciones o valores disponibles. Si existen opciones que no se ven, puede hacer scroll con los scrollbars (vertical / horizontal).

Ej.



## Atributos

Para acceder a los atributos de un ListBox dé dobleclick sobre él.

Atributo	Tipo de dato	Descripción
BorderStyle	BorderStyle	Especifica el estilo de borde del control. Los valores son: StyleBox! StyleLowered! StyleRaised! StyleShadowBox!
Item[]	String	Especifica los ítems en el control
Multiselect	Boolean	Especifica si el usuario puede seleccionar múltiples ítems en el control. TRUE, FALSE.
Sorted	Boolean	Especifica si los ítems aparecen ordenados alfabéticamente en el control. TRUE, FALSE.

## Eventos

El evento más utilizado del ListBox es 'SelectionChanged', el cual ocurre cuando se selecciona un ítem de la lista.

## Funciones

- AddItem

Agrega un nuevo ítem a la lista de valores de un ListBox.

## Sintaxis

Listboxname.AddItem(item)

Item, es un string cuyo valor es el texto del ítem que se desea agregar.

## Retorno

Integer. La función retorna la posición del nuevo ítem. Si la lista está ordenada, la posición retornada es la posición luego de ordenar la lista.

Una lista puede tener ítems duplicados. Los ítems en la lista son referenciados por su posición, no su texto.

## Ejemplo

Este ejemplo agrega el ítem 'Edit File' al ListBox lb\_actions.

```
integer rownbr
```

```
string s
```

```
s = "Edit File"
```

```
rownbr = lb_Actions.AddItem(s)
```

Si lb\_actions contenía 'Add' y 'Run' y la propiedad Sorted es FALSE, la sentencia anterior retorna 3. Si la propiedad Sorted es TRUE, la sentencia retorna 2.

- DeleteItem

Elimina un ítem de la lista de valores del ListBox.

## Sintaxis

Listboxname.DeleteItem( index )

Index, indica la posición del ítem que se desea eliminar.

## Retorno

Integer. La función retorna el número de ítems que quedan en el ListBox luego de eliminar.

### Ejemplos

Si lb\_actions contiene 10 ítems, esta sentencia elimina el ítem 5 y retorna 9.

```
lb_actions.DeleteItem(5)
```

Estas sentencias eliminan el primer ítem seleccionando de lb\_actions.

```
integer li_Index
```

```
li_Index = lb_actions.SelectedIndex( )  
lb_actions.DeleteItem(li_Index)
```

La siguiente sentencia elimina el ítem 'Persona' del ListBox lb\_purpose.

```
lb_purpose.DeleteItem( lb_purpose.FindItem( "Personal", 1) )
```

- FindItem

Encuentra el próximo ítem en un ListBox que comience con el texto de búsqueda especificado.

### **Sintaxis**

```
Listboxname.FindItem( text, index )
```

Text: Un string cuyo valor es el texto de comienzo del ítem que se desea encontrar.

Index: El número del ítem antes del primer ítem donde comenzará la búsqueda. Para buscar en toda la lista, especificar 0.

### **Retorno**

Integer. La función retorna el index del primer ítem que coincide. Para coincidir, el texto del ítem debe comenzar con el texto especificado. Si no encuentra, retorna -1.

### **Ejemplo**

Asuma que el ListBox lb\_actions contiene la siguiente lista:

Index	Texto del ítem
1	Open files
2	Close files
3	Copy files
4	Delete files

Entonces, la sentencia siguiente comienza la búsqueda de 'Delete' con el ítem 2 (Close files). FindItem setea nIndex a 4.

Integer nIndex

```
nIndex = lb_actions.FindItem( "Delete", 1)
```

#### - InsertItem

Inserta un ítem en la lista de valores de un ListBox.

#### Sintaxis

```
Listboxname.InsertItem(item, index)
```

Item: Un string cuyo valor es el texto del ítem que se desea insertar.

Index: El número del ítem que quedará después del ítem insertado.

#### Retorno

Integer. Retorna la posición final del ítem. Retorna -1 si hubo error.

#### Ejemplo

Esta sentencia inserta el ítem "Run Application" antes del 5to ítem en lb\_actions.

```
lb_actions.InsertItem("Run Application", 5)
```

#### - Reset

Elimina todos los ítems de un ListBox.

#### Sintaxis

```
Lb_actions.Reset()
```

#### - SelectedIndex

Retorna el index del ítem en el ListBox que está actualmente seleccionado. Si hay más de ítem seleccionado, retorna el index del primer ítem seleccionado.

### Ejemplo

Si el ítem 5 de lb\_actions está seleccionado, li\_Index queda setado con 5.

integer li\_Index

```
li_Index = lb_actions.SelectedIndex( )
```

- SelectedItem

Retorna el texto del primer ítem seleccionado.

### Ejemplo

Si el texto del ítem seleccionado en el ListBox lb\_shortcuts es 'F1', entonces ls\_Item queda seteado a 'F1'.

string ls\_Item

```
ls_Item = lb_Shortcuts.SelectedItem( )
```

- SelectItem

Encuentra y destaca un ítem en el control.

### Sintaxis 1

```
Listboxname.SelectItem( item, index )
```

Item: Un string cuyo valor es el texto de comienzo del ítem que se desea destacar.

Index: El número del ítem después del cual se desea comenzar la búsqueda.

Ejemplo.

Si el ítem 5 en lb\_actions es 'Delete Files', este ejemplo comienza buscando después del ítem 2, encuentra y destaca 'Delete Files', seteando li\_Index a 5.

integer li\_Index

```
li_Index = lb_Actions.SelectItem("Delete Files", 2)
```

## Sintaxis 2

Listboxname.SelectItem( itemnumber )

Itemnumber: Un integer cuyo valor es el index del ítem que se desea destacar.  
Especifique 0 para deseleccionar el ítem seleccionado.

- Text

Obtiene el texto de ítem en un ListBox.

## Sintaxis

Listboxname.text( index )

Index: El número del ítem para el cual se desea el texto.

## Ejemplo

Asuma que lb\_cities contiene:

Atlanta  
Boston  
Chicago  
Denver

Entonces, esta sentencia almacena el texto del ítem 3 ( 'Chicago' ) en current\_city.

string current\_city

current\_city = lb\_Cities.Text(3)

- TotalItems

Retorna el número total de ítems en el control.

## Ejemplos

Si lb\_actions contiene un total de 5 ítems, este ejemplo setea nTotal a 5.

integer Total

Total = lbx\_Actions.TotalItems( )



Este FOR es ejecutado para cada ítem en lb\_actions.

```
integer Total, n
```

```
Total = lb_Actions.TotalItems( )
```

```
FOR n = 1 to Total
```

```
    ... // algún proceso
```

```
NEXT
```

```
- TotalSelected
```

Retorna el número total de ítems seleccionados en el control.

## Ejemplos

Si hay tres ítems seleccionados en lb\_actions, este ejemplo setea SelectedTotal a 3.

```
integer SelectedTotal
```

```
SelectedTotal = lb_Actions.TotalSelected( )
```

Estas sentencias en el evento SelectionChanged de lb\_actions muestra un mensaje si el usuario intenta seleccionar más de 3 ítems.

```
IF lb_Actions.TotalSelected( ) > 3 THEN
```

```
    MessageBox("Atención", &
```

```
    "Sólo seleccione hasta 3 ítems!.")
```

```
ELSE
```

```
    ... // algún proceso
```

```
END IF
```

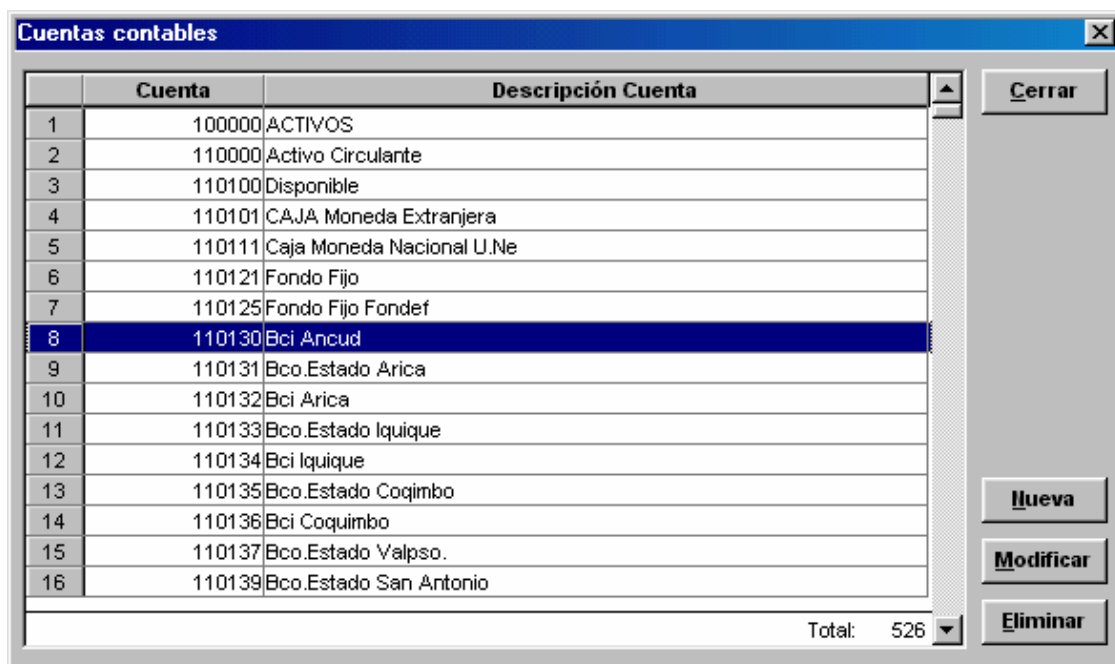
## DataWindow control

Se ponen DataWindows controls en una ventana u objeto de usuario y se le asocia el DataWindow object que se desea usar para visualizar y manipular los datos.

Para asociar un DataWindow object en modo ejecución:

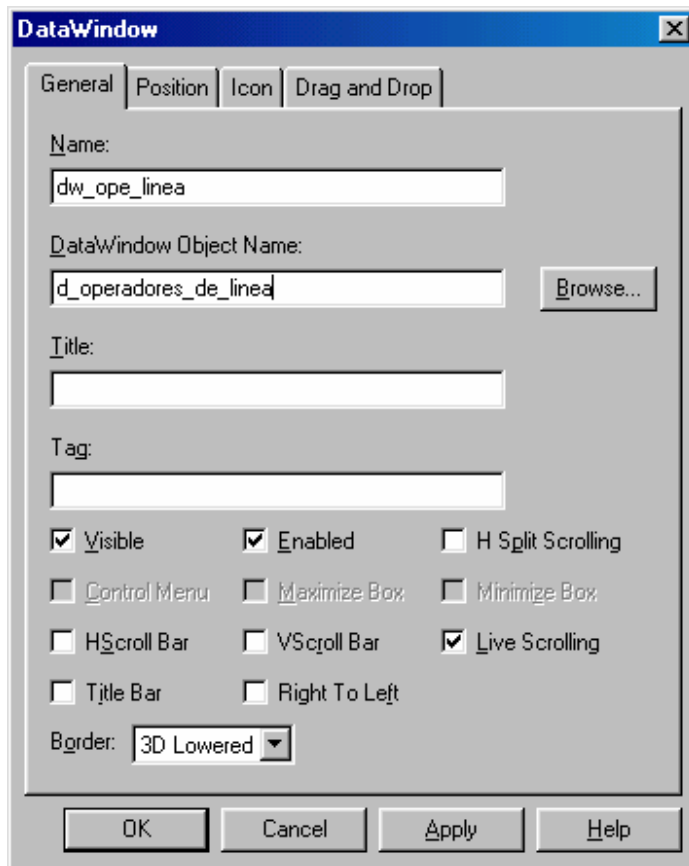
```
dw_1.DataObject = "d_operadores_de_linea"
```

Ej:



### Atributos

Para acceder a los atributos de un DataWindow control dé dobleclick sobre él.



Algunos atributos del DataWindow control.

Atributo	Tipo de dato	Descripción
BorderStyle	BorderStyle	Especifica el estilo de borde del control. Los valores son: StyleBox! StyleLowered! StyleRaised! StyleShadowBox!
Dataobject	String	Especifica el nombre del DataWindow object o Report asociado al control.
HScrollBar	Boolean	Especifica si una barra de scroll horizontal se muestra cuando no se ven todos los datos a la derecha. TRUE, FALSE.
VScrollBar	Boolean	Especifica si una barra de scroll vertical se muestra cuando no se ven todos los datos. TRUE, FALSE.

## Eventos

Algunos eventos tienen códigos de retorno usados para determinar que acción toma lugar después que el evento ocurre. Los códigos de retorno se setean usando la sentencia `Return { expresión }`.

Evento	Ocorre
Clicked	<p>Cuando el usuario da un click de mouse en un campo no editable o entre campos.</p> <p>Códigos de Retorno:  0 – (default) Continúa proceso.  1 – detiene proceso.</p>
Constructor	Inmediatamente antes que ocurra el evento Open de la ventana que contiene al DataWindow control.
DBError	<p>Cuando ocurre un error de base de datos en el DataWindow control.</p> <p>Códigos de retorno:  0 – (default) Muestra el mensaje de error.  1 – No muestra el mensaje de error.</p>
Destructor	Inmediatamente después que ocurre el evento Close de la ventana.
DoubleClicked	Cuando el usuario da dobleclick sobre un campo no editable o entre campos en el DataWindow control.
DragDrop	Cuando el usuario arrastra un control y suelta el botón del mouse para dejar caer el control al DataWindow.
DragEnter	Cuando un objeto arrastrado entra al DataWindow control.
DragLeave	Cuando un objeto arrastrado deja el DataWindow control.
DragWithin	Cuando un objeto arrastrado pasa por encima del DataWindow control.
EditChanged	Ocorre por cada tecla que el usuario tipea en un edit control en el DataWindow.
Error	Ocorre cuando se produce un error de dato.
GetFocus	Justo antes que el DataWindow control reciba el foco (antes que se seleccione y llegue a ser activo).
ItemChanged	<p>Cuando un campo en el DataWindow ha sido modificado y pierde el foco (por ejemplo, el usuario presiona ENTER, la tecla TAB, una tecla de flecha, o da click en otro campo dentro del DataWindow).</p> <p>Códigos de retorno:  0 – (default) Acepta el valor del dato.  1 – Trae el valor anterior y no permite que cambie el foco.  2 – Trae el valor anterior y permite que cambie el foco.</p>
ItemError	<p>Cuando un campo ha sido modificado, el campo pierde el foco (por ejemplo, el usuario presiona ENTER, la tecla TAB, una tecla de flecha, o da click en otro campo dentro del DataWindow), y el campo no pasa la regla de validación para esa columna.</p> <p>Códigos de Retorno:  0 – (default) Trae el valor anterior y muestra un mensaje de error.  1 – Trae el valor anterior sin mensaje de error.  2 – Acepta el valor del dato.  3 – Trae el valor anterior y permite que cambie el foco.</p> <p>Si el código de retorno es 0 o 1, el campo con el dato incorrecto gana el foco.</p>

ItemFocusChanged	Ocurre cuando el ítem actual en el control cambia.
LoseFocus	Cuando el DataWindow pierde el foco (queda inactivo).
PrintEnd	Cuando finaliza la impresión del DataWindow.
PrintStart	Cuando comienza la impresión del DataWindow.
RbuttonDown	Cuando se presiona el botón derecho del mouse sobre el DataWindow.
Resize	Cuando el usuario o un script redimensiona el DataWindow.
RetrieveEnd	Cuando finaliza un retrieve.
RetrieveRow	Después que se recupera (retrieve) una fila.  Códigos de retorno: 0 – (default) Continúa. 1 – Detiene el retrieve.
RetrieveStart	Cuando el retrieve de un DataWindow va a comenzar.  Códigos de retorno: 0 – (default) Continúa. 1 – No ejecuta el retrieve. 2 – No hace reset a las filas antes de recuperar los datos de la base de datos.
RowFocusChanged	Cuando la fila actual en el DataWindow cambia.
UpdateEnd	Cuando finalizan todas las actualizaciones en el DataWindow.
UpdateStart	Después que se llama la función update del DataWindow y justo antes que los datos sean enviados a la base de datos.  Códigos de retorno: 0 – (default) Continúa. 1 – No ejecuta el update.

## Funciones

Se describen a continuación algunas de las funciones más empleadas.

Función	Tipo de dato que retorna	Descripción
AcceptText	Integer	Aplica el contenido del edit control al ítem actual en el buffer del DataWindow.
Create	Integer	Crea un DataWindow Object usando el código especificado y reemplaza el actual DataWindow object con el nuevo.
DBCcancel	Integer	Cancela un retrieve en proceso.
DeletedCount	Long	Retorna el número de filas que han sido eliminadas del DataWindow control pero que aún no han sido actualizadas en la tabla asociada.
DeleteRow	Integer	Elimina la fila especificada del DataWindow control.
Describe	String	Retorna información solicitada acerca de la estructura del DataWindow control.
Drag	Integer	Comienza o finaliza el arrastre (drag&drop) del DataWindow control.
Filter	Integer	Muestra filas específicas de un DataWindow control en base al filtro actual del DataWindow control.
FilteredCount	Integer	Retorna el número de filas que no son visibles debido al actual filtro.
FindGroupChange	Long	Busca la fila del próximo break para el grupo especificado y a

		partir de una fila dada.
GetChild	Integer	Almacena en la variable especificada el nombre del child DataWindow en la columna especificada.
GetClickedColumn	Integer	Retorna el número de la columna a la cual el usuario le dio click o doble-click.
		GetClickedColumn es una función obsoleta. La columna clickeada está disponible ahora como un argumento del evento.
GetClickedRow	Long	Retorna el número de la fila a la cual el usuario le dio click o doble click.
		GetClickedRow es una función obsoleta. La fila clickeada está disponible ahora como un argumento del evento.
GetColumn	Integer	Retorna el número de la columna actual.
GetColumnName	String	Retorna el nombre de la columna actual.
GetItemDate	Date	Retorna la fecha en la fila y columna especificada.
GetItemDateTime	DateTime	Retorna el dato DateTime en la fila y columna especificada.
GetItemDecimal	Decimal	Retorna el dato Decimal en la fila y columna especificada.
GetItemNumber	Double	Retorna el dato numérico en la fila y columna especificada.
GetItemStatus	dwItemStatus	Retorna el estado del ítem en la fila y columna especificada.
		Los valores son:
		DataModified!
		New!
		NewModified!
		NotModified!
GetString	String	Retorna el dato string en la fila y columna especificada.
GetItemTime	Time	Retorna el dato time en la fila y columna especificada.
GetNextModified	Long	Retorna el número de la primera fila que fue modificada en el buffer especificado después de una fila específica.
GetRow	Long	El número de la fila actual.
GetSelectedRow	Integer	Retorna el número de la primera fila seleccionada después de una fila específica.
GetSQLPreview	String	Retorna la actual sentencia SQL del DataWindow control.
		Esta función está obsoleta. La sintaxis SQL está ahora disponible como argumento del evento.
GetSQLSelect	String	Retorna la actual sentencia SELECT del DataWindow control.
GetText	String	Retorna el texto en el edit control sobre la fila y columna actual.
GetValue	String	Obtiene el valor de un ítem en una lista de valores o tabla de códigos asociada a una columna.
GroupCalc	Integer	Recalcula los break en los grupos que tenga el DataWindow.
ImportFile	Long	Copia datos desde un archivo al DataWindowcontrol.
ImportString	Long	Copia datos desde un string al DataWindow control.
InsertRow	Long	Inserta una nueva fila antes de la fila especificada.
IsSelected	Boolean	Retorna True si la fila especificada está seleccionada; retorna False si la fila no está seleccionada o es mayor que el número de filas del DataWindow.
ModifiedCount	Long	Retorna el número de filas que han sido modificadas en el DataWindow control pero que no han sido actualizadas a la base

		de datos aún.
Modify	String	Usa la especificación contenida en un string para modificar el DataWindow control.
Print	Integer	Envía el contenido del DataWindow control a la impresora actual.
Reset	Integer	Limpia todos los datos del DataWindow.
ResetUpdate	Integer	Resetea el flag de update del DataWindow.
Retrieve	Long	Recupera filas desde la base de datos.
RowCount	Long	Retorna el número de filas actualmente disponibles en el DataWindow. (Las filas recuperadas desde la base de datos menos las eliminadas y filtradas, más las insertadas).
RowsCopy	Integer	Copia un rango de filas desde un DataWindow control a otro.
RowsDiscard	Integer	Descarga un rango de filas. Las filas no pueden ser restauradas sin recuperarlas nuevamente desde la base de datos.
RowsMove	Integer	Elimina un rango de filas de un DataWindow y las inserta en otro.
SaveAs	Integer	Graba el contenido del DataWindow al archivo especificado.
ScrollToRow	Integer	Hace scroll a la fila especificada.
SelectRow	Integer	Selecciona o deselecciona la fila especificada.
SetActionCode	Integer	Define la acción que tomará el DataWindow después que ocurra un evento.
		Esta función está obsoleta. Los códigos de retorno se setean ahora usando la sentencia Return {expresión} en el script del evento.
SetColumn	Integer	Hace la columna especificada la columna actual.
SetFilter	Integer	Define el criterio de filtro para el DataWindow.
SetFocus	Integer	Da el foco al DataWindow.
SetItem	Integer	Setea el valor de la fila/columna especificada.
SetItemStatus	Integer	Setea el estado de una fila en una columna y buffer especificado.
SetRedraw	Integer	Controla redibujado automático del DataWindow después de cada cambio en sus propiedades o contenido.
SetRow	Integer	Hace la fila especificada la fila actual en el DataWindow.
SetRowFocusIndicator	Integer	Setea el indicador de fila actual del DataWindow.
SetSort	Integer	Define el criterio de ordenamiento del DataWindow. El ordenamiento es ejecutado por la función sort().
SetText	Integer	Reemplaza el texto en el edit control para la fila/columna actual.
SetTransObject	Integer	Setea el objeto transacción para el DataWindow.
ShareData	Integer	Comparte datos entre un DataWindow primario y uno secundario.
Sort	Integer	Ordena las filas en el DataWindow basado en el actual criterio de ordenamiento.
TriggerEvent	Integer	Gatilla un evento específico en el DataWindow y ejecuta el script contenido en él.
Update	Integer	Envía a la base de datos todos los insert, delete y update dados al DataWindow control.

## **Comportamiento del DataWindow Control**



## *Comportamiento del DataWindow Control*

Este documento describe el DataWindow control, el DataWindow Object, los niveles de validación para DataWindows, y pautas para el uso de algunas funciones PowerBuilder con DataWindows.

Para esta discusión, estos términos se definen como sigue:

- Un DataWindow control actúa como un visor en una ventana, define los atributos, y reserva espacio para el DataWindow object.
- Un DataWindow object es un objeto creado con el pintor de DataWindow.
- Un ítem es una celda dentro de un DataWindow control.
- Un edit control es un campo de texto que actúa como un repositorio del DataWindow control. Mantiene la información que es ingresada hasta que ésta es validada.

### **Items y Edit control en un DataWindow Control**

Un DataWindow control es un control PowerBuilder que permite al usuario mostrar, manipular, y actualizar información en la base de datos. Para entender como trabaja el DataWindow control, es importante considerar que un DataWindow control consiste de dos partes: el edit control, y los ítems.

#### **Edit control**

El edit control es un campo de texto que mantiene la información que es ingresada, hasta que es validada. Cada DataWindow control tiene un edit control.

#### **Items**

Los ítems (o celdas) pueden ser de uno a seis tipos, y son almacenados en los buffers internos del DataWindow control.

#### **Validación de los Datos**

PowerBuilder valida los datos, y los mueve desde el edit control al ítem bajo él. La validación comienza cuando el usuario modifica los datos en el edit control y una de éstas condiciones ocurre cuando:

- El usuario presiona la tecla Enter.

- El usuario intenta moverse (cambiar foco) a un campo diferente en el DataWindow control.
- Un script ejecuta la función AcceptText.

### **Actualizando el ítem**

Durante el proceso de validación, los valores en el ítem y el edit control pueden ser diferentes. El edit control recibe los datos ingresados y los mantiene hasta que son validados. La información en el ítem es actualizada con el valor del edit control, sólo después que el dato nuevo en el edit control ha pasado los cuatro niveles de validación que se discuten más adelante.

### **Actualizando la base de datos**

Los valores de la base de datos no cambian hasta que la función Update se ejecuta de modo satisfactorio.

### **Niveles de validación**

Hay cuatro niveles de validación:

- Algo cambió.
- Chequeo de tipo de dato
- Reglas de validación
- Evento Itemchanged

Cuando una columna pasa un test de validación, continúa al próximo nivel hasta que pasa los cuatro niveles. Cuando el contenido del edit control pasa todos los niveles de validación, el ítem es actualizado, pero la base de datos no cambia. Para actualizar la base de datos, se debe ejecutar exitosamente la función Update.

### **Algo cambió**

En este primer nivel de validación, el DataWindow control testea el contenido del edit control para ver si es diferente del dato en el ítem. Cuando el contenido del edit control es diferente, se cumple esta condición, y pasa al siguiente test. El usuario no tiene control sobre este test.

### **Sin cambios**

Cuando el usuario hace Tab o el mouse se mueve a otro campo, el DataWindow control determina si hubo algún cambio. Si no lo hubo, no ejecuta más validaciones.

### **Los valores cambian**

Cuando se ingresan datos en el edit control, el DataWindow control compara los contenidos del edit control y el ítem bajo él, para ver si son iguales. Si los valores son iguales, no hay cambio. La validación se detiene y no se pasa a los siguientes niveles. El dato en el DataWindow no cambia.

### **Chequeo de tipo de dato**

En el segundo test de validación, PowerBuilder chequea para ver si el tipo de dato entrado en el edit control, coincide con el tipo de dato del ítem.

Por ejemplo, si una columna de tipo integer contiene 123 y el usuario entra ABC en el edit control, esta acción pasa el primer nivel de validación (algo cambió). Sin embargo, el segundo test falla pues ABC no es integer válido.

### **Falla validación de tipo de dato**

Cuando la validación de tipo de dato falla, se produce un evento ItemError. El programador puede dejar que PowerBuilder maneje el error, o puede atrapar el error y ejecutar algún otro proceso. Por ejemplo, puede escribir su propio mensaje de error.

### **Manejo del error por defecto**

Cuando se deja que PowerBuilder maneje el error de validación, el usuario recibe un mensaje standard: 'el ítem no pasa test de validación'. El cursor retorna al edit control. Para descartar la entrada y comenzar de nuevo, el usuario puede presionar la tecla escape. Esto restaura el valor original del ítem al edit control.

### **Manejo del error por el usuario**

Cuando el programador escribe un mensaje de error propio, necesita saber lo que el usuario ha entrado en el edit control. Para recuperar el valor del edit control, use:

```
dw_x.GetText()
```

Para comparar el valor del edit control con el ítem, use una de las siguientes funciones, basadas en los tipos de dato. Se debe conocer el tipo de dato del ítem. Si no lo sabe, puede determinarlo con la función Describe.

```
dw_x.GetItemString()  
dw_x.GetItemNumber()  
dw_x.GetItemDate()  
dw_x.GetItemTime()  
dw_x.GetItemDateTime()  
dw_x.GetItemDecimal()
```

Recuerde:

- El edit control es siempre de tipo de dato string.
- PowerBuilder siempre compara lo que hay en el edit control con el contenido del ítem.
- El valor en el ítem puede no ser el que piensa, especialmente cuando usa tablas de código.
- Lo que se ve en el edit control no es necesariamente lo que tiene el ítem.
- Haga un `GetItemxxx` para ver qué contiene el ítem antes de continuar.

El usuario puede obtener un mensaje de error cuando intenta borrar un valor del edit control. El string vacío tiene tipo de dato de texto, debido a que es un string. Si el tipo de dato del ítem no es string, falla la validación. Para manejar este error, setée a nulo y permita que el foco se mueva de columna.

## Reglas de validación en DataWindow controls

El tercer test de validación chequea cualquier regla de validación definida para una columna en el DataWindow object. Se pueden usar reglas de validación en el DataWindow object para validación de rangos, o chequeos específicos de valores. Cuando una columna falla en una regla de validación de un DataWindow control, se produce un evento `ItemError`.

Para definir una regla de validación en el pintor de DataWindow:

- click el botón derecho del mouse en la columna
- seleccione el ítem de menú 'Validation'
- entre la expresión de validación en la ventana 'Column Validation Definition'

Se puede también definir su propio mensaje de error en este lugar. Recuerde que el mensaje es una expresión string, y por lo tanto debe evaluar a string. Por ejemplo:

"El valor: " + `GetText()` + ", debe ser mayor que 500."

### Nota

---

Se pueden también definir reglas de validación en el pintor de base de datos (DataWindow painter) como atributos extendidos de columna. Cuando se definen las reglas de validación en el pintor de base de datos, las reglas son copiadas sólo a 'nuevos' DataWindow objects. Los objetos DataWindow que ya existen, permanecen sin cambiar.

---

## Evento ItemChanged

El evento ItemChanged es el cuarto y último nivel de validación. Use este nivel de validación para chequeos de integridad y otras validaciones. Puede usarse una amplia variedad de funciones para acompañar la validación, y puede invocarse el evento ItemError usando la función SetActionCode o Return.

### Ejemplo

Intente este ejemplo para ver como trabajan los cuatro niveles de validación:

- Cree una tabla con 2 columnas definidas como integer.
- Haga una columna la columna clave.
- Cree un DataWindow object conteniendo esta columna.
- En el pintor de DataWindow, seleccione 'Preview' desde el 'Design menu'.
- Click en el ícono 'Insert' en la barra de herramientas.
- Antes que ingrese cualquier regla de validación:
- Tipée ABC en la columna.
- Presione la tecla Enter- para comenzar el proceso de validación.

Hubo cambio ?

La entrada pasa el primer test de validación: algo cambió ? (SI). Entonces va al segundo test de validación.

Tipo de dato ?

Es el tipo de dato entrado, del mismo que el del ítem ? (NO). El segundo nivel de validación rechaza la entrada, debido a que se entró un dato no numérico dentro de una columna numérica. Si hubiera entrado 123, se habrían pasado los dos primeros niveles de validación.

Regla de validación

Ahora cree una regla de validación para la columna #2 que chequea por un rango entre 25000 y 50000.

#2 < 50000 and #2 > 25000

Si se ingresa 4000, no pasa el test de validación, y obtiene un error. Si se ingresa 40000, tampoco pasa la validación. Porqué no ?. Porque la validación tal como está, no está mirando el texto que se ingresó (que aún no ha sido aceptado), sino que está mirando el valor de la columna 2, la cual contiene el valor inicial especificado. NULL es el defecto cuando no se entra un valor inicial.

Recuerde que el edit control 'recibe' los datos hasta que los datos son validados.

Para mirar en el texto del edit control, entre la regla de validación como sigue:

```
NUMBER( GetText() ) > 25000 and NUMBER( GetText() ) < 50000
```

Esto permite mirar el valor del edit control y procesar correctamente. Ahora, cuando se ingresa 40000, es aceptado, pero 4000 aún es rechazado.

### **Aceptando o rechazando valores basados en un evento**

La función SetActionCode permite aceptar o rechazar valores basándose en un evento. El formato para SetActionCode es:

```
dw_x.SetActionCode( código )
```

Para setear el código de acción para la acción previa, a 1, cree un script con esta línea:

```
dw_x.SetActionCode(1)
```

Importante:

El valor que se pasa a la función SetActionCode, depende del evento en el que se está cuando se llama la función. Por ejemplo, cuando se setea el código de acción a 1 en el script para el evento ItemChanged, causa un evento ItemError.

Nota: Algunas funciones del DataWindow control pueden resetear el código de acción. Si el comando SetActionCode no es la última línea en el script, ponga un comando Return después de la línea para terminar el script. De otra manera, se puede obtener un resultado que no se desea. Por ejemplo:

```
dw_x.SetActionCode(1)  
Return
```

### **Usando Códigos de Acción**

Los códigos de acción para el evento ItemChanged son:

- 0 (defecto) Acepta el valor del dato
- 1 Rechaza el valor del dato
- 2 Rechaza el valor del dato, permite que cambie el foco.

Los códigos de acción para el evento ItemError son:

0 (defecto) – Rechaza el valor del dato, muestra un mensaje de error del sistema.

1 - Rechaza el valor del dato, no muestra el mensaje del sistema.

2 - Acepta el valor del dato.

3 - Rechaza el valor del dato, permite que cambie el foco.

Usando los códigos de acción en varias combinaciones, se puede lograr un control óptimo de los datos que se ingresan en el DataWindow. Por ejemplo, seteando el código de acción a 1 en el evento ItemChanged, se rechaza el valor del dato sin mirar el hecho que haya pasado los tres niveles previos de validación. Esto puede causar un evento ItemError. Dependiendo del código de acción seteado en el evento ItemError, se puede mostrar el mensaje de error del sistema, mostrar un mensaje propio, aceptar el valor (incluso si fué rechazado en el evento ItemError) o permitir que cambie el foco después que el valor es rechazado.

### Borrando el contenido de un edit control

Antes de la Versión 3.0, cuando en una columna con un tipo de dato distinto de string se borraba el contenido del edit control y entonces se intentaba mover a otra columna, se estaba tratando de forzar a un string de longitud cero una columna no string. Esta acción fallaba en el test de validación del tipo de dato. Para setear la columna a Null, y permitir que el foco cambie de columna, agregue este código en el evento ItemError:

```

Int NULL_NUM      // contendrá un valor nulo
Int nCol
Long nRow

SetNull(NULL_NUM)

nRow = this.GetRow()    // obtiene la fila actual
nCol = this.GetCol()    // obtiene la columna actual

If nCol = 2 Then
  If this.GetText() = "" Then // setea el valor de la columna...
    this.SetItem( nRow, nCol, NULL_NUM )    // ...a valor nulo
    this.SetActionCode(3)
  Return
End If
End If

```

Este código asume que la columna 2 es numérica. Si es date, time, ó datetime, reemplace la primera línea (Int NULL\_NUM) con una declaración apropiada al tipo de dato.

Note que el SetActionCode le dice al evento ItemError que rechace el valor entrado en el edit control y permita que cambie el foco.

Comenzando con la Versión 3.0, hay una opción para una columna en un DataWindow llamada 'Empty string is NULL'. Para acceder esta opción, click con el botón derecho del mouse en la columna y seleccione 'Edit Styles>Edit'. En la ventana response Edit style chequee la casilla 'Empty string is NULL'. Cuando está ON, al borrar el contenido de un edit control sobre una columna no string, no se produce un error pues el string vacío es interpretado como un valor nulo.

### **Validando la última columna de un DataWindow control**

Un DataWindow control tiene los eventos LoseFocus y GetFocus. Cuando se usa la tecla tab o el mouse para moverse fuera de un DataWindow control a otro control de la ventana, el DataWindow no se da cuenta que el usuario se ha movido. Así, el DataWindow control no sabe que el ítem que ha perdido foco, puede haber cambiado (el ítem actúa como si aún tuviera foco).

Para validar la última columna, en el script para el evento LoseFocus, añada:

```
dw_x.AcceptText()
```

### **Saltándose validaciones en el DataWindow control**

Para saltarse todos los niveles de validación, excepto el chequeo de tipo de dato, y poner un valor en un ítem, use esta función:

```
dw_x.SetItem( row, col, valor )
```

### *Evitando la falla de protección general por 'Stack Fault' (falla de pila)*

Siga estas reglas. Cada una de estas funciones invoca otro evento ItemChanged, causando un loop sin fin, provocando una falla de protección general.

- Nunca ponga la función AcceptText dentro de un evento ItemChanged.
- Nunca use SetColumn en un evento ItemChanged.
- Nunca use SetRow en un evento ItemChanged.

Usando la función Update en un evento ItemChanged

La función Update tiene un argumento que puede ser TRUE o FALSE:



`dw_x.Update( argumento )`

El *argumento* cuando es TRUE, ejecuta un AcceptText. De este modo, si se llama `dw_x.Update(TRUE)` dentro del evento ItemChanged, es lo mismo que llamar a la función AcceptText dentro del evento ItemChanged, causando un error de protección general.

Sin embargo, es aceptable llamar `dw_x.Update(FALSE)`, desde el evento ItemChanged.

#### **Nota**

---

Sea cuidadoso cuando use este último en el evento ItemChanged. Recuerde que el ítem no es automáticamente actualizado hasta que el evento ItemChanged es completado exitosamente.

---

#### **Mostrando el valor en un edit control**

En un evento ItemChanged, SetItem sólo cambia el ítem bajo él. No cambia el valor en el edit control. Para mostrar el nuevo valor en el edit control, use la función SetText luego de un SetItem.

## **Operaciones sobre el DataWindow**

---

## Operaciones sobre el DataWindow

En este capítulo veremos las principales operaciones que se efectúan sobre un DataWindow control.

La mayoría de las funciones y operaciones básicas que se muestran, se aplican a Datawindow controls, child DataWindows y DataStore objects.

### *SetTransObject*

Antes de realizar cualquier operación sobre el DataWindow control, debe indicarse el objeto transacción que usará. Este objeto provee la información necesaria para la comunicación con la base de datos, y sus valores usualmente se setean en el evento open de la aplicación, antes de realizar la sentencia connect.

#### *Sintaxis*

*dwcontrol*.SetTransObject( *transaction* )

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control.
<i>transaction</i>	El nombre del objeto transacción que usará el dwcontrol.

#### *Retorno*

Integer. Retorna 1 si hubo éxito. -1 si hubo error.

#### *Uso*

Antes de conectarse a la base de datos usando la sentencia *connect*, se deben setear los parámetros del objeto transacción. PowerBuilder proporciona un objeto transacción global llamado SQLCA.

Ej.

```
dw_secuencial_embarque.SetTransObject( Sqlca )
```

### *Retrieve*

Obtiene filas desde una base de datos y las muestra en el DataWindow control. Si se incluyen argumentos, éstos los usan los argumentos de retrieve de la sentencia SELECT del DataWindow object o DataWindow child.

### Sintaxis

*dwcontrol.Retrieve( {,argument, argument, ...} )*

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control.
<i>argument</i> (opcional)	Uno o más valores que se desea usar como argumentos de retrieve en la sentencia SELECT definida para el dwcontrol.

### Retorno

Long. Retorna el número de filas que obtiene y muestra (buffer primary!) , y -1 si hay error. Si cualquier valor de argumento es NULL, Retrieve retorna NULL.

### Uso

Después que se obtienen las filas, se aplica el filtro que tenga el DataWindow object. De esta forma, las filas que no coincidan con el criterio del filtro, se mueven inmediatamente al buffer filter!, y no se incluyen en la cuenta retornada.

Normalmente, cuando se llama Retrieve(), se descargan del Datawindow control las filas que tenga, y se ponen las obtenidas con el último retrieve. Si no se desea que funcione de esta forma, en el evento RetrieveStart digite: Return 2. En este caso, Retrieve agrega las filas obtenidas a las que existían previamente.

Ej. 1

```
dw_saldos.SetTransObject( Sqlca )
dw_saldos.Retrieve()
```

Ej. 2

Long nRows

```
dw_working_program.SetTransObject( Sqlca )
nRows = dw_working_program.Retrieve( ano_ope, cor_ope, cor_sec, codlugar )
```

If nRows = 0 Then

.....

End If

Ej 3.

```
dw_facturacion.Retrieve( sle_agencia.text, em_ano.text, em_mes.text )
```

## GetItemString

Obtiene datos cuyo tipo es string desde el buffer especificado de un DataWindow control o DataStore. Se pueden obtener los datos que fueron originalmente obtenidos por retrieve, así como valores en los buffer primary!, delete! o Filter!.

### Sintaxis

```
dwcontrol.GetItemstring( row, column, {,dwbuffer,originalvalue} )
```

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control, DataStore o child DataWindow, desde el que se desea obtener undato tipo string en la fila/columna especificada.
<i>row</i>	Un long que indica la fila.
<i>column</i>	La columna del dato. El tipo de dato de la columna debe ser string. Column puede ser un número de columna (integer) o un nombre de columna (string).
<b>Tips</b> Para obtener el contenido de un campo calculado, especifique el nombre del campo calculado en <i>column</i> . Los campos calculados no tienen número.	
<i>dwbuffer</i> (opcional)	Un valor del tipo de dato enumerado dwBuffer, que especifica el buffer del DataWindow desde el cual se desea obtener los datos: <ul style="list-style-type: none"> <li>• Primary! – (default) Datos en el buffer Primary! (datos que no han sido eliminados o filtrados.</li> <li>• Delete! – Datos en el buffer Delete! (datos eliminados del DataWindow)</li> <li>• Filter! – Datos en el buffer Filter! (datos que no cumplieron con el filtro)</li> </ul>
<i>originalvalue</i> (opcional)	Un boolean que indica si se desea el valor original o el valor actual del DataWindow, para la fila/columna especificada: <ul style="list-style-type: none"> <li>• True – Retorna el valor original, esto es, el valor inicialmente recuperado (retrieve) desde la base de datos.</li> <li>• False – Retorna el valor actual.</li> </ul>
Si se especifica <i>dwbuffer</i> , también debe especificarse <i>originalvalue</i> .	

Retorno

String. Retorna NULL si el valor de la columna es NULL. Retorna un string vacío ("" ) si ocurre un error. Si el valor de cualquier argumento es NULL, GetString retorna NULL.

#### *Uso*

Use GetString para obtener información desde los buffers del DataWindow. Para obtener lo que el usuario entra en la columna actual, antes que sea aceptado por el DataWindow, use GetText. En los eventos ItemChanged o ItemError, use el argumento de evento *data*.

#### **Mismatched data types**

Si el tipo de dato de la columna en el DataWindow no coincide con el tipo de dato de la función (en este caso string), se produce un error de ejecución.

#### Ej. 1

Extrae desde la fila 3 del buffer Primary!, el valor actual de emp\_name.

String LName

```
LName = dw_employee.GetString(3, "emp_name")
```

#### Ej. 2

Extrae desde la fila 3 del buffer Delete!, el valor actual de emp\_name.

String LName

```
LName = dw_employee.GetString(3, "emp_name", Delete!,FALSE)
```

#### Ej. 3

Extrae desde la fila 3 del buffer Delete!, el valor original de emp\_name.

String LName

```
LName = dw_employee.GetString(3, "emp_name", Delete!, TRUE)
```

#### **Nota**

Para obtener datos de otros tipos, usar las funciones:

GetItemDate  
 GetItemDateTime  
 GetItemDecimal  
 GetItemNumber  
 GetItemTime

## SetItem

Setea el valor de una fila y columna en un DataWindow control o DataStore al valor especificado.

*Sintaxis*

*dwcontrol.SetItem( row, column, value )*

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control, DataStore o child DataWindow, al cual se desea especificar un valor para una fila/columna.
<i>row</i>	Un long que indica la fila
<i>column</i>	Localización de columna del dato. <i>Column</i> puede ser un número de columna (integer) o un nombre de columna (string).
<i>value</i>	El valor que se desea setear. El tipo de dato de <i>value</i> debe ser del mismo tipo que la columna.

### Retorno

Integer. Retorna 1 si es exitoso. -1 si hubo error.

Ej. 1

```
dw_order.SetItem(3, "fecha_ingreso", 1993-06-07)
```

Ej. 2

Cuando un usuario comienza a editar una columna numérica y la deja sin entrar dato, PowerBuilder intenta asignar un string vacío a la columna. Esto produce un error en la validación de tipo de dato. En este ejemplo, el código en el evento ItemError setea el valor de la columna a NULL y permite que cambie el foco.

El ejemplo asume que el tipo de dato de la columna 2 es numérico. Si es date, time, o datetime, reemplace la primera línea (integer null\_num) con una declaración apropiada al tipo de dato.

---

```

integer null_num           // para contener valor nulo

SetNull(null_num)

// Procesamiento especial para la columna 2
If dwo.ID = 2 Then
    // Si el usuario no ingresa algo (""), setea a null
    If data = "" THEN
        This.SetItem(row, dwo.ID, null_num)
        Return 2
    End If
End If

```

Ej. 3

El siguiente ejemplo es un script para el evento ItemError. Se salta caracteres que no sean números.

```

String snum, c

integer cnt

FOR cnt = 1 to Len(data)
    c = Mid(data, cnt, 1) // Obtiene caracter
    If IsNumber(c) THEN snum = snum + c
NEXT
This.SetItem(row, dwo.ID, Long(snum))
RETURN 3

```

## DataObject

Para cambiar el DataWindow object de un DataWindow control durante la ejecución de la aplicación, use:

```
dwcontrol.DataObject = 'd_otro_dw_object'
```

## InsertRow

Inserta una fila en un DataWindow o DataStore. Si existen columnas que tienen valores por defecto, la fila se inicializa con estos valores antes de mostrarse.



---

*Sintaxis**dwcontrol.InsertRow( row )*

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control.
<i>row</i>	Un long que indica la fila antes de la cual se desea insertar la fila. Para insertar una fila al final, especifique 0.

**Retorno**

Long. Retorna el número de la fila que se agregó, y -1 si ocurrió un error.

*Uso*

InsertRow simplemente inserta la fila sin cambiar el display actual (lo que se ve). Para hacer scroll a la fila (si no fuera visible) y hacerla actual, llame la función ScrollToRow. Para sólo hacerla actual, llame a SetRow.

**Ej.1**

Esta sentencia inserta una fila antes de la fila 7 en dw\_empleado.

```
dw_empleado.InsertRow(7)
```

**Ej. 2**

Este ejemplo, inserta una fila al final en dw\_empleado, y luego hace scroll a la fila haciéndola actual.

Long ll\_newrow

```
ll_newrow = dw_empleado.InsertRow(0)  
dw_empleado.ScrollToRow(ll_newrow)
```

---

## DeleteRow

Elimina una fila de un DataWindow control, DataStore objet o child DataWindow.

### *Sintaxis*

*dwcontrol.DeleteRow( row )*

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control.
<i>row</i>	Un long que indica la fila que se desea eliminar. Para eliminar la fila actual, especifique 0.

### **Retorno**

Integer. Retorna si es exitosamente eliminada. -1 si hubo error.

### *Uso*

DeleteRow elimina la fila del buffer primary! del DataWindow control.

Si el DataWindow object no tiene capacidad de ‘actualizable’ (updateable), todos los almacenamientos asociados con la fila son borrados. Si el DataWindow tiene capacidad de actualizable, DeleteRow mueve la fila al buffer delete!. Posteriormente usa esta información al momento de construir la sentencia SQL ( al llamar la función update() ) para actualizar la base de datos.

Al llamar a DeleteRow, la fila no es eliminada de la base de datos, hasta que se llama la función Update. Luego de realizar el update exitosamente, y los flags de update son reseteados, entonces el almacenamiento asociado a la fila eliminada se limpia.

### Ej. 1

Esta sentencia elimina la fila actual de dw\_nave.

```
dw_nave.DeleteRow(0)
```

### Ej. 2

Estas sentencias eliminan la fila 5 de dw\_nave y actualizan a la base de datos el cambio.

```
dw_nave.DeleteRow(5)  
dw_nave.Update( )
```

## Find

Encuentra la siguiente fila en un DataWindow o DataStore en la cual los datos coinciden con una condición especificada.

### Sintaxis

*dwcontrol*.Find( *expression*, *start*, *end* )

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control en el cual se desea buscar.
<i>expression</i>	Un string cuyo valor es una expresión Boolean que se desea usar como criterio de búsqueda. La expresión incluye nombres de columnas.
<i>start</i>	Un long que especifica la fila en la cual comienza la búsqueda. <i>Start</i> puede ser mayor que el número de filas.
<i>end</i>	Un long que especifica la fila en la cual terminar la búsqueda. <i>End</i> puede ser mayor que el número de filas. Para buscar hacia atrás, haga <i>end</i> menor que <i>start</i> .

### Retorno

Long. Retorna el número de la primera fila que coincide con el criterio de búsqueda dentro del rango especificado. Retorna 0 si no encuentra, y un número negativo si ocurrió un error.

### Uso

La búsqueda es case-sensitive. Cuando se compara texto con el valor de una columna , el case debe coincidir.

Si se usa Find en un loop que busca a través de todas las filas, puede ocurrir un loop sin fin si la última fila satisface la condición de búsqueda. Cuando el valor *start* llega a ser mayor que *end*, la búsqueda se invierte y Find siempre encontrará, lo cual terminará en un loop sin fin.

Para resolver este problema, se puede hacer el valor *end* uno más que el número de filas. Otra forma, como se muestra a continuación, consiste en testear dentro del loop si la fila actual es mayor que el número de filas, y si es así, terminar la búsqueda.

Ej. 1

```
long ll_find = 1, ll_end
```

```
ll_end = dw_main.RowCount( )
```

---

```

ll_find = dw_main.Find(searchstr, ll_find, ll_end)
DO WHILE ll_find > 0
    ... // procesa fila encontrada
    ll_find++
    // previene loop sin fin
    IF ll_find > ll_end THEN EXIT
    ll_find = dw_main.Find(searchstr, ll_find, ll_end)
LOOP

```

## Ej. 2

Se busca la primera fila en dw\_status en la cual el valor de la columna emp\_salary sea mayor que 100.000. La búsqueda comienza en la fila 3 y continúa hasta el total de filas del DataWindow.

```
dw_status.Find("emp_salary > 100000", 3, dw_status.RowCount( ))
```

## E. 3

Para testear en más de una columna, use operadores boolean para unir expresiones condicionales. El siguiente ejemplo, busca los empleados de nombre Smith cuyo salario es mayor que 100.000.

```
dw_status.Find( "emp_lname = 'Smith' and emp_salary > 100000", &
    1, dw_status.RowCount( ))
```

## Ej. 4

Se busca la primera fila en dw\_emp que coincida con el valor que el usuario entró en el SingleLineEdit llamado Name. Notar las comillas simples en la expresión de búsqueda alrededor del nombre.

```
string ls_lname_emp
```

```
long ll_nbr, ll_foundrow
```

```
ll_nbr = dw_emp.RowCount( )
```

```
// Quita espacios en blanco a la izquierda y derecha.
```

```
ls_lname_emp = Trim(sle_Name.Text)
```

```
ll_foundrow = dw_emp.Find( "emp_lname = '"+ ls_lname_emp + "'", 1, ll_nbr)
```

## Ej. 5

Este script busca la primera fila que tiene un valor nulo en la columna emp\_id. Si no encuentra, el script actualiza el DataWindow. Si encuentra un nulo, muestra un mensaje.

```
If dw_status.AcceptText( ) = 1 Then

    If dw_status.Find("IsNull(emp_id)", 1, dw_status.RowCount( )) > 0 Then
        MessageBox("Advertencia", "Valor nulo en emp_id. No se actualiza.")
    Else
        dw_status.Update( )
    End If

End If
```

## Ej. 6

El siguiente script asociado a un command button 'Buscar siguiente', busca la primera fila que coincide con el criterio especificado y hace scroll a la fila. Cada vez que se da click al botón, el número de la fila se almacena en la variable de instancia il\_found. La siguiente vez que el usuario da click al botón 'Buscar siguiente', la búsqueda continúa desde la siguiente fila. Cuando la búsqueda llega a su fin, se muestra un mensaje al usuario diciendo que no hay más coincidencias. La próxima vez, la búsqueda continúa desde la primera fila.

Notar que el criterio de búsqueda está en duro. Un escenario más realista, debería incluir un botón 'Buscar', que pidiera al usuario el criterio de búsqueda. El criterio se debería almacenar en una variable de instancia.

```
long ll_row
// Obtiene el número de fila desde la variable de instancia il_found para
// comenzar la búsqueda
ll_row = il_found

// Busca usando criterio predefinido
ll_row = dw_main.Find( "item_id = 3 or item_desc = 'Nails'", ll_row, dw_main.RowCount( ) )
If ll_row > 0 Then
    // Fila encontrada, hace scroll para hacerla actual
    dw_main.ScrollToRow( ll_row )
Else
    // No se encontró
    MessageBox("Buscar", "No se encontró.")
End If

// Graba el número de la próxima fila para el comienzo de la
// próxima búsqueda. Si no se encontró, ll_row es 0, haciendo
```

```
// il_found 1, de tal modo de iniciar la siguiente búsqueda
// desde el comienzo.
```

```
il_found = ll_row + 1
```

Ej. 7

Este ejemplo busca en todas las filas de dw\_main, y contruye una lista de nombres que incluyan la letra minúscula 'a'. Note que el valor final de la búsqueda es uno mayor que el total de filas, evitando un loop sin fin si el nombre en la última fila satisface la búsqueda.

```
long ll_find, ll_end
```

```
string ll_list
```

```
// El valor final es uno mayor que el total
```

```
ll_end = dw_main.RowCount( ) + 1
```

```
ll_find = 1
```

```
ll_find = dw_main.Find("Pos(last_name,'a') > 0", ll_find, ll_end)
```

```
DO WHILE ll_find > 0
```

```
    // une nombre
```

```
    ll_list = ll_list + '~r' + dw_main.GetItemString( ll_find, 'last_name')
```

```
    // Busca nuevamente
```

```
    ll_find++
```

```
    ll_find = dw_main.Find("Pos(last_name,'a') > 0", ll_find, ll_end )
```

```
LOOP
```

## SetFilter

Especifica el criterio de filtro para un DataWindow control o DataStore.

### Sintaxis

*dwcontrol*.SetFilter(*format* )

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control, DataStore o child Datawindow al que se desea especificar su criterio de filtro.
<i>format</i>	Un string cuyo valor es una expresión boolean que se desea usar como criterio de filtro. La expresión incluye nombres o números de columnas. Un número de columna debe ser precedido por el signo #. Si format es NULL, PowerBuilder pone una ventana de diálogo solicitando el filtro.

### Retorno

Integer. 1 si es exitoso. -1 si hubo error.

### Uso

Un DataWindow object puede tener un criterio de filtro como parte de su definición. Después que se recuperan los datos, las filas que no cumplen con el criterio de filtro, son inmediatamente transferidas desde el buffer primary! al buffer Filter!.

La función SetFilter reemplaza el criterio definido para el DataWindow object, si lo hubiera.

Para aplicar el criterio de filtro al DataWindow control o DataStore, llame la función Filter.

La expresión filtro, consiste de columnas, operadores relacionales, y valores que se comparan con columnas. Las expresiones boolean pueden conectarse con los operadores lógicos AND y OR . También se puede usar el operador de negación NOT. Use paréntesis para controlar el orden de evaluación.

Ejemplos de expresiones:

```
Item_id > 5
NOT item_id = 5
(NOT item_id = 5) AND customer > "Leiva"
item_id > 5 AND customer = "Villalón"
#1 > 5 AND #2 = "Horacio"
```

La expresión filtro es un string y debe contener variables. Sin embargo, se puede construir el string durante la ejecución usando los valores de variables en el script. Dentro del filtro, los valores que son string deben encerrarse con comillas simples (ver ejemplos).

Si la expresión contiene números, el DataWindow espera estos números en formato U.S. Cuide que las funciones de string que formatean números, lo hagan usando el sistema actual de seteo.

### **Removiendo un filtro**

Para remover un filtro, llame SetFilter con el string vacío ("") por formato y entonces llame a Filter. Si habían filas en el buffer Filter, se restauran al buffer primary!.

Para permitir que los usuarios especifiquen su propio filtro para el DataWindow control, se debe pasar un string nulo a SetFilter. PowerBuilder muestra la caja de diálogo para especificación de filtro. Entonces se llama la función Filter para aplicar el criterio de filtro al DataWindow. No se puede pasar un string nulo a SetFilter para un DataStore.

#### **Ej. 1**

Esta sentencia define la expresión de filtro para dw\_puertos, como el valor de formato1.

```
dw_Employee.SetFilter(format1)
```

#### **Ej. 2**

Las siguientes sentencias definen una expresión de filtro y la setean como el filtro para dw\_empleado. Con este filtro, sólo aquellas filas en las que la columna cust\_qty sea mayor que 100 y la columna cust\_code sea mayor que 30 se mostrarán. La sentencia final llama a Filter aplicando el filtro.

String DWfilter2

```
DWfilter2 = "cust_qty > 100 and cust_code >30"  
dw_empleado.SetFilter(DWfilter2)  
dw_empleado.Filter( )
```

#### **Ej. 3**

La sentencia siguiente define un filtro para que se muestren los empleados cuyo estado sea "ME".



```
string Var1
```

```
Var1 = "ME"  
dw_Employee.SetFilter("emp_state = '"+ var1 + " ")
```

Ej. 4

```
Integer max_qty, min_qty
```

```
min_qty = 100  
max_qty = 1000
```

```
dw_inv.SetFilter("#1=" + String( min_qty) + " and #2=" + String(max_qty))
```

Ej. 5

Se muestra una caja de diálogo para que el usuario defina el criterio de filtro.

```
String null_str
```

```
SetNull(null_str)  
dw_main.SetFilter(null_str)  
dw_main.Filter( )
```

## Filter

Muestra las filas que cumplen con el criterio de filtro actual de un DataWindow. Las filas que no cumplen con el criterio se mueven automáticamente al buffer Filter!. Están presentes en el DataWindow pero son invisibles al usuario.

*Sintaxis*

```
dwcontrol.Filter( )
```

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control, DataStore o child Datawindow que se desea filtrar.

**Retorno**

Integer. 1 si fue exitoso. -1 si hubo error.

*Uso*

Filter usa el criterio de filtro actual. Para cambiar el criterio de filtro, use la función *SetFilter*. Si no se llama *SetFilter* para especificar un criterio de filtro antes de llamar a *Filter*, éste usa el criterio de filtro especificado en la definición del *DataWindow Object*, si lo hubiera.

Cuando la función *Retrieve* obtiene datos para el *DataWindow*, PowerBuilder aplica el filtro definido para el *DataWindow Object*. Se necesita llamar a *Filter* después que se cambia el criterio de filtro con *SetFilter* o si los datos han cambiado debido a entrada de datos.

Si está seteado 'Retrieve As Needed', la función *Filter* cancela este efecto. *Filter* causa que se recuperen todas las filas y entonces aplica el filtro.

*Filter* no tiene efecto en *DataWindows* presentes en un reporte *composite*.

**Filtros y grupos**

Cuando se filtra un *DataWindow* con grupos, puede ser necesario llamar la función *GroupCalc* después de llamar a *Filter*.

Ej.

Esta sentencia muestra las filas en *dw\_empleado* basado en su criterio actual de filtro.

```
dw_empleado.Filter( )
```

## GetItemStatus

Reporta el status (estado) de modificación de una fila o columna dentro de una fila. El status de modificación determina el tipo de sentencia SQL que la función Update generará para la fila o columna.

### Sintaxis

*datawindowname.GetItemStatus ( row, column, dwbuffer )*

Parámetro	Descripción
<i>Datawindowname</i>	El nombre del datawindow control o Datawindow child del cual se desea obtener el status de una fila o columna en una fila.
<i>Row</i>	Un long que identifica la fila para la que se desea obtener el status.
<i>Column</i>	La columna para la cual se desea el status. <i>Column</i> puede ser un número de columna (integer) o un nombre de columna. Especifique 0 para obtener el status de la fila completa.
<i>Dwbuffer</i>	Un valor del tipo de dato enumerado dwBuffer, identificando el buffer del DataWindow que contiene la fila: <ul style="list-style-type: none"> <li>• Primary! – Los datos en el buffer primario ( los datos que no han sido eliminados o filtrados)</li> <li>• Delete! – Los datos en el buffer delete ( los datos que han sido eliminados desde el DataWindow object).</li> <li>• Filter! – Los datos en el buffer filter ( los datos que no coincidieron con el filtro).</li> </ul>

### Retorno

dwItemStatus. Un valor del tipo de dato enumerado dwItemStatus. Retorna el status del ítem en la fila/columna del *datawindowname* en el buffer *dwbuffer*. Si la columna es 0, GetItemStatus retorna el status de la fila.

### Uso

Los valores para dwItemStatus son:

- NotModified! – La información en la fila o columna está sin cambiar desde que se recuperó.
- DataModified! – La información en la columna o una de las columnas en la fila cambió desde que se recuperó.
- New! – La fila es nueva, pero no se han especificado valores para sus columnas. ( Se aplica a filas solamente, no a columnas individuales).
- NewModified! – La fila es nueva y se han asignado valores a sus columnas. Además de los cambios causados por entradas del usuario o la función SetItem, una nueva fila obtiene status NewModified!, cuando una de sus columnas tiene valor por defecto. (Se aplica a filas solamente, no a columnas individuales.)

Use `GetItemStatus` para comprender qué sentencia SQL se generará para información nueva o modificada, cuando se actualice la base de datos con la función `Update`.

**Buffers primary y filter.** La función `Update` genera una sentencia `INSERT` para filas con status `NewModified!`. Genera una sentencia `UPDATE` para filas con `DataModified!`. Sólo columnas con status `DataModified!` se incluyen en la sentencia `UPDATE`.

**Buffer delete.** La función `Update` genera una sentencia `DELETE` para filas con status `NotModified!` O `DataModified!` que hayan sido eliminadas del `DataWindow`. No genera una sentencia `DELETE` para filas con status `New!` o `NewModified!`.

Ej.

Estas sentencias almacenan en la variable `l_status`, el status de la columna llamada `emp_status` en la fila 5 en el buffer primary de `dw_1`:

```
dwItemStatus l_status
```

```
l_status = dw_1.GetItemStatus(5, "emp_status", Primary!)
```

Ej. 2

Estas sentencias almacenan en la variable `l_status` el status de la columna llamada `Salary` en la fila actual en el buffer primary de `dw_emp`:

```
dwItemStatus l_status
```

```
l_status = dw_emp.GetItemStatus(dw_emp.GetRow(), "Salary", Primary!)
```

Ver uso de `SetItemStatus`.

## Update

Actualiza la base de datos con los cambios hechos en el DataWindow control o DataStore. Update también puede llamar la función AcceptText para la fila/columna actual antes de actualizar la base de datos.

### Sintaxis

```
dwcontrol.Update( {accept {,resetflag} } )
```

Argumento	Descripción
<i>dwcontrol</i>	El nombre del datawindow control que contiene la información que se va a actualizar en la base de datos.
<i>accept</i> (opcional)	Un boolean que especifica si el DataWindow control o DataStore deben ejecutar automáticamente la función AcceptText antes de ejecutar el Update. <ul style="list-style-type: none"> <li>• TRUE (default) Ejecuta el AcceptText. El Update es cancelado si falla en la validación.</li> <li>• FALSE No ejecuta AcceptText.</li> </ul>
<i>resetflag</i> (opcional)	Un Boolean que especifica si dwcontrol debe automáticamente resetear los flag de update. <ul style="list-style-type: none"> <li>• TRUE (default) Resetea los flag.</li> <li>• FALSE No resetea los flag</li> </ul>

### Retorno

Integer. Retorna 1 si fue exitoso. -1 si ocurrió un error.

### Uso

Antes de ejecutar la función Update, es necesario usar SetTrans o SetTransObject, para especificar la conexión a la base de datos. Cuando se usa SetTransObject, la más eficiente de las dos, se debe realizar un manejo propio de la transacción, lo que incluye enviar las sentencias SQL COMMIT o ROLLBACK para finalizar el Update.

### Nota

Es buena práctica testear el código de éxito o falla después de llamar la función Update. Además de chequear el valor retornado por Update, se puede chequear la propiedad SQLNRows del objeto transacción, para asegurar que el update actuó al menos sobre una fila.

Por defecto, Update resetea los flags de update después de completar exitosamente. Sin embargo, esto se puede evitar hasta que se ejecuten otras validaciones y se haga commit a los cambios. Cuando se esté satisfecho con las validaciones, llame a ResetUpdate para limpiar los flag de tal modo que los ítems no queden marcados como modificados, eliminados o insertados modificados. Si se opta por esta técnica, sólo puede usar SetTransObject, que da control de cuando a los datos se les hace commit.

Si se está actualizando múltiples DataWindow controls como parte de una transacción, setée el argumento *resetflag* a FALSE para cada uno de los Update. Esto evitará que un DataWindow ‘olvide’ que filas debe actualizar en caso que uno de los Update falle. Llame a RollBack , intente corregir el problema, y actualice nuevamente. Una vez que todos los Update se completan sin error, use COMMIT para finalizar la transacción, y luego ResetUpdate para resetear los flags de cada DataWindow control.

Si se llama Update con el argumento *resetflag* a FALSE, y luego no se llama ResetUpdate, el DataWindow intentará enviar la misma sentencia SQL de nuevo, la próxima vez que se llame Update.

#### Ej. 1

Este ejemplo se conecta a la base de datos, especifica un objeto transacción para el DataWindow control con SetTransObject, y hace Update con los cambios hechos en dw\_nave. Por defecto, se ejecuta AcceptText en los datos en el edit control para la fila/columna actual, y los flags de status se resetean.

CONNECT USING SQLCA;

```
dw_nave.SetTransObject(SQLCA)
... // algún proceso
dw_nave.Update( )
```

#### Ej. 2

Este ejemplo se conecta a la base de datos, especifica un objeto transacción para el DataWindow control con SetTransObject, y luego actualiza la base de datos con los cambios realizados en dw\_operador. El update resetea los flags de status, pero no ejecuta AcceptText antes de actualizar la base de datos.

CONNECT USING SQLCA;

```
dw_operador.SetTransObject(SQLCA)
... // algún proceso
dw_operador.Update(FALSE, TRUE)
```

#### Ej. 3

Como antes, este ejemplo se conecta a la base de datos, especifica un objeto transacción para el DataWindow control con SetTransObject, y actualiza la base de datos con los cambios hechos en dw\_armador. Después que se ejecutó el update, el ejemplo, chequea el código de retorno, y dependiendo del éxito del update, ejecuta un COMMIT o ROLLBACK.

integer rtn

Connect Using Sqlca;  
dw\_armador.SetTransObject(Sqlca)  
rtn = dw\_armador.Update( )

If rtn = 1 AND Sqlca.SQLNRows > 0 Then  
    COMMIT USING SQLCA;  
Else  
    ROLLBACK USING SQLCA;  
End If

## **Técnicas y códigos de uso frecuente**



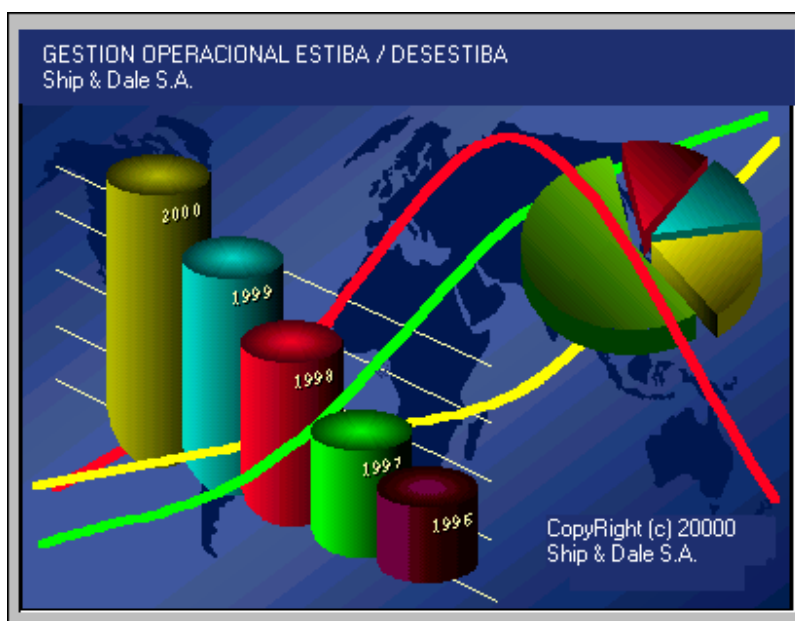
## *Técnicas y códigos de uso frecuente*

Se dan en este capítulo, algunas técnicas y código frecuente en scripts de eventos para algunos objetos seleccionados por su importancia de uso. Más adelante, veremos mayor cantidad de código interesante, en el capítulo que recoge los tips de PowerBuilder.

### **Objeto Aplicación**

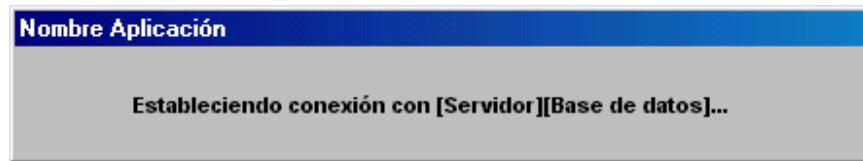
El objeto aplicación es quién comienza la aplicación, conectándose a la base de datos y abriendo la ventana principal, la cual a su vez tiene asociado el menú principal de la aplicación. Un detalle elegante para la partida de la aplicación consiste en abrir una ventana de presentación, la cual debiera llevar el nombre de la aplicación y un bitmaps tipo logo de la aplicación. La ventana de presentación se presenta durante algunos segundos, desaparece y se abre la ventana principal de la aplicación.

w\_presentacion



Otro detalle, es poner una ventana popup, sin código, que indica que está intentando conectarse a la base de datos. Al terminar el connect, se cierra y se indica si hubo o no error en la conexión.

w\_conectar\_a\_db



Con el objeto de ahorrar conexiones a la base de datos, se emplean dos funciones, f\_conectar y f\_desconectar; las cuales conectan y desconectan de la base de datos usando el objeto transacción que se les entrega como parámetro. Así, cada vez que se requiera realizar una transacción con la base de datos, deben usarse ambas funciones. Esta técnica puede o no parecer útil y realmente depende de situaciones particulares. De todos modos, se expone aquí como una alternativa. Lo otro es simplemente reemplazar en el código, f\_conectar() con “connect using sqlca;”, y lograr una conexión siempre abierta para la aplicación.

```
f_conectar ( transaction otran ) -> integer
```

```
-----
```

```
f_desconectar( otran )
```

```
connect using otran;
```

```
return otran.sqlcode
```

```
f_desconectar( transaction otran ) -> integer
```

```
-----
```

```
disconnect using otran;
```

```
return otran.sqlcode
```

El siguiente código de ejemplo ilustra estos pasos:

#### Evento Open de la aplicación

```
// cArchIni es una variable global, definida:
```

```
// String cArchIni = “c:\windows\app.ini”
```

```
Setpointer(hourglass!)
```

```
// Verifica si se está ejecutando el programa
```

```
If Handle(This, TRUE) > 0 Then
```

```
    MessageBox(This.AppName,"La aplicación ya está ejecutándose")
```

```
    HALT CLOSE
```

```
End If
```

```

// Toolbar en español al dar click derecho a la barra de menú.
this.toolbarframetitle = "Barra de Menú"
this.ToolBarPopupMenuText = "Izquierda, Arriba, Derecha, Abajo, Flotante, Ver Texto,
Ver Tips"

// Parámetros de conexión
sqlca.dbms      = ProfileString(cArchIni,"sqlca","dbms","")
sqlca.logid     = ProfileString(cArchIni,"sqlca","logid","")
sqlca.logpass   = ProfileString(cArchIni,"sqlca","logpass","")
sqlca.database  = ProfileString(cArchIni,"sqlca","database","")
sqlca.servername = ProfileString(cArchIni,"sqlca","servername","")

Open( w_conectar_a_db )

f_conectar( Sqlca )

Close( w_conectar_a_db )

If sqlca.sqlcode <> 0 then
    MessageBox ("Error en conexión", "No puede efectuarse conexión a base de
    datos.~n~r~n~r" + sqlca.sqlerrtext + ".~n~r~n~rSQLdbCode: " +
    STRING(sqlca.sqldbcode) )
    halt close;
    return
End if

f_desconectar( Sqlca )

// Nombre corto de la aplicación para que aparezca en la barra de microhelp al
// iniciar la aplicación.
this.MicroHelpDefault = "Mi aplicación"

//Abre ventana principal
// inicialmente está seteada visible = False
open (w_main)

w_main.enabled = False

// Abre ventana de presentacion
// se presenta algunos segundos y luego es cerrada por la ventana principal.
open (w_presentacion,w_main)

```

La ventana de presentación es una ventana tipo popup, sin título y sólo con un picture en el cual va el archivo .bmp. El control de los segundos de aparición y posterior cierre de la ventana de presentación, lo realiza la ventana principal.

Cómo se dijo, la ventana de presentación inicialmente está invisible, y el evento open de la aplicación la deja además enabled = False. Veamos el código en la ventana principal.

#### Variable de Instance

```
int aux_tiempo = 0
```

#### Open

```
Setpointer(hourglass!)
timer(2,This)
```

#### timer

```
// Se hace visible la ventana principal
this.visible = TRUE
aux_tiempo = aux_tiempo + 1

If aux_tiempo >= 1 Then
    timer(0,This)
    // Cierra ventana de presentacion
    Close ( w_presentacion )
    this.enabled = True
End If
```

#### Close

```
Disconnect Using Sqlca;
```

Como se aprecia, con un mínimo código podemos dotar al inicio de nuestras aplicaciones un toque profesional.

## Ventana

### Evento Open

El evento open de la ventana, en general sirve para setear variables, algunos atributos de los controles que contiene la ventana, gatillar algún evento que muestre datos al usuario, y en general, preparar la ventana para dejarla en condición de servir de interfaz con el usuario.

La ventana se muestra al usuario cuando *el evento open ha terminado*. Por esto si la ventana contiene algún procesamiento que tiene larga duración, el usuario verá el reloj de arena durante todo este tiempo sin ver la ventana. Si se desea abrir la ventana, mostrarla al usuario inmediatamente mientras se realiza el proceso largo, crear un evento de usuario ue\_postOpen donde se pondrá el código, y en el evento open poner lo siguiente:

#### Open

```
This.PostEvent( 'ue_postOpen' )
```

Esto terminará inmediatamente el evento open y seguirá con algún procesamiento dejado a la cola, en este caso, el evento ue\_postOpen.

Por lo general cuando una ventana tiene un DataWindow control, en el evento open de la ventana, se gatilla algún evento del DataWindow control que contiene el código necesario para traer los datos.

Ej.

```
dw_1.TriggerEvent( 'ue_retrieve' )
```

### **Evento CloseQuery**

Un evento Closequery ocurre cuando se va a cerrar una ventana. PowerBuilder revisa el valor de Message.ReturnValue de cualquier ventana activa (child, popup y sheets) que se cierra. Si este valor es 1, la ventana no puede ser cerrada.

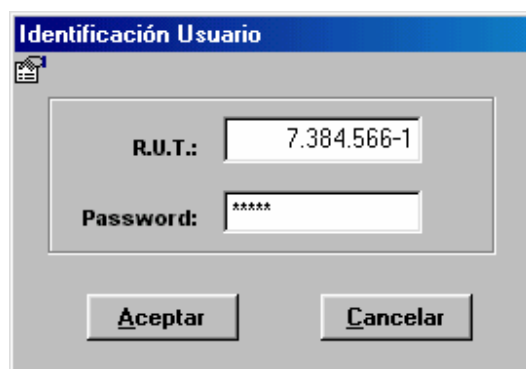
Para ejecutar algún proceso o chequeo antes de cerrar una ventana, setee el valor de Message.ReturnValue a 1, y entonces escriba un script para el evento CloseQuery que ejecute el proceso deseado. Asegúrese de setear luego Message.ReturnValue = 0, cuando el proceso se complete, para que la ventana pueda cerrarse.

Nota: La sentencia que setea Message.ReturnValue a 1 para prevenir que la ventana se cierre, debe ser la última sentencia del script para el evento CloseQuery.

Lo anterior nos entrega una forma de evitar que el usuario cierre una ventana, por ejemplo, usando Alt+F4. Esto es de vital importancia cuando solicitamos un password de ingreso a la aplicación.

Ej.

Tenemos la siguiente ventana (La ventana es response sin control menú):



y no queremos que el usuario la cierre usando por ejemplo, Alt + F4. Lo único aceptable es que presione los botones 'Aceptar' o 'Cancelar'.

Veamos el código:

Variable Instance

Boolean bCerrar = FALSE

Open

Message.ReturnValue = 1

em\_1.SetFocus()

CloseQuery

Int nRet =0

If NOT bCerrar Then

    nRet = 1

End If

Message.ReturnValue = nRet

Botón 'Cancel'

Halt Close;

Botón 'Aceptar'

String cRut, cPass

Int nRet

// Inicializa bCerrar a False.

bCerrar = FALSE

```

/*****
/* COMIENZA VALIDACION          */
*****/

```

cRut = em\_1.text

cPass = sle\_1.text

If ISNULL(cRut) Then

    MessageBox( "Aceptar", "Ingrese R.U.T." )

    em\_1.SetFocus()

    Return -1

End If

If f\_valrut( cRut ) Then

    cRutJP = cRut

Else

    MessageBox( parent.title, "Rut ingresado no es válido!" )

    em\_1.SetFocus()

    Return -1

End If

// Verifica rut y password

// 1. RUT debe ser personal activo

// 2. RUT debe ser jefe de proyecto sin fecha de término

// 3. Password debe ser el del RUT.

declare login procedure for @nRet = pa\_login

    @rut = :cRutJP,

    @pass = :cPass;

f\_conectar( sqlca )

execute login;

If sqlca.sqlcode <> 0 Then

    MessageBox( "Error en login", sqlca.sqlerrtext )

    f\_desconectar( sqlca )

```

    Return -1
End If

Fetch login into :nRet;

close login;

f_desconectar( sqlca )

Choose case nRet
case -1
    MessageBox( "Error en login", "RUT no está activo." )
    em_1.SetFocus()
    Return -1
case -2
    MessageBox( "Error en login", "Password no corresponde" )
    sle_1.SetFocus()
    Return -1
end choose

/* FINALIZA VALIDACION */

// Login OK
// Se pone bCerrar = True, lo cual permite cerrar la ventana.

bCerrar = TRUE
Close( parent )

```

### Evento Timer

Este evento se gatilla cuando ha pasado un número específico de segundos después de haber llamado la función timer.

La función timer puede ser llamada desde cualquier lugar de la ventana, y causa que se gatille un evento timer en la ventana cada cierta cantidad de tiempo.

La sintaxis para la función timer es:

Timer (*interval* {, *windowname* } )

Argumento	Descripción
<i>interval</i>	El número de segundos (0-65) que se desea entre eventos timer. Si <i>interval</i> es 0, se acaba el timer y no se sigue gatillando el evento timer de la ventana.



---

<i>windowname</i> (opcional)	La ventana en la que se desea gatillar el evento timer. La ventana debe ser una ventana abierta. Si no se especifica una ventana, el evento timer ocurre en la ventana actual.
---------------------------------	--

---

### Tips

- No llame la función timer desde el evento timer.
- Cuando se selecciona un intervalo entre 0 y .055 ( cerca de 1/18 de segundo), el evento timer se gatilla cada .055 segundos, lo más fino que permite Windows.

## DataWindow

Se exponen aquí, códigos de ejemplo para los eventos comúnmente usados en los DataWindows control. El lector puede utilizar sus propias técnicas en cada uno de los puntos tratados aquí. Sin embargo, los códigos de ejemplo son una referencia válida para el tratamiento de los distintos eventos.

### Evento Clicked

El siguiente código escrito para PowerBuilder 4 y PowerBuilder 5, permite ordenar por una columna en modo ascendente. Para lograr esto, el usuario debe dar un click al header de la columna que desea ordenar.

Al crear un datawindow object mediante el pintor de DataWindow, se crean en forma automática los header, los cuales tienen la forma:

nombrecolumna\_t

Por ejemplo, si existe una columna llamada rut\_cliente, también existirá por defecto, su header (si no lo ha eliminado o cambiado) de nombre rut\_cliente\_t.

### PowerBuilder 4

#### Clicked

```
String cObjeto, cColumna, cHeader
Int  nCol, nPos
nCol  = this.GetClickedColumn()
cObjeto = this.Describe( "#" + STRING(nCol) + ".Name")
```

```
If cObjeto = "!" Then
  cObjeto = this.GetObjectAtPointer()
  nPos  = POS( cObjeto , "_t" )
  cHeader = LEFT( cObjeto, nPos + 1)
  cColumna = LEFT( cObjeto , nPos - 1 )
  this.Modify( cHeader + ".border='5'" )
  this.SetSort( cColumna + " A" )
  this.Sort()
  this.Modify( cHeader + ".border='6'" )
  // Luego del Sort, queda en el 1er registro.
  this.PostEvent( RowFocusChanged! )
End If
```

### PowerBuilder 4

Clicked

```
String cHeader, cColumna
If dwo.type = "text" Then
    cHeader = dwo.Name
    cColumna = Left( cHeader, Len(cHeader) - 2 )
    this.Modify( cHeader + ".border='5'" )
    this.SetSort( cColumna + " A" )
    this.Sort()
    this.Modify( cHeader + ".border='6'" )
    // Luego del Sort, queda en el 1er registro.
    this.PostEvent( RowFocusChanged! )
End If
```

**Nota**


---

Para hacer el orden inverso, ponga el mismo código en el evento rbuttonDown cambiando la 'A' por 'D' al aplicar la función SetSort(...).

---

**Evento Constructor**

```
// Indica qué objeto de transacción debe usar el DataWindow control para
// acceder a la base de datos.
// en este ejemplo, asumiremos que usará el objeto transacción por defecto
// SQLCA.
```

```
this.SetTransObject( Sqlca )
```

```
// Setea indicador de fila actual
```

```
this.SetRowFocusIndicator( indicador )
```

indicador puede ser:

- Off!. Sin indicador
- FocusRect!. Pone un rectángulo de línea punteada en la fila actual.
- Hand!. Usa un puntero en forma de mano.
- Nombre de un archivo BMP.

## Evento ItemFocusChanged

Para mostrar microhelp (help en barra inferior) para cada campo editable de un DataWindow hacer lo siguiente:

En la lista de Tag del DataWindow object, poner los help que se desea por columna.

En el evento ItemFocusChanged

```
String cTag = ""
Int    nCol
nCol = this.GetColumn()
If nCol > 0 Then
    cTag = this.Describe( "#" + STRING( nCol ) + ".tag" )
End If
```

```
w_main_mdi.SetMicroHelp( cTag )
```

donde w\_main\_mdi, es la ventana principal tipo mdi con microhelp.

## Evento RowFocusChanged

En este evento es usual colorear la fila actual. Se dan códigos para PowerBuilder 4 y 5.

### PowerBuilder 4

```
Long nRow, nRowSel
```

```
nRow = This.GetRow()
If nRow < 1 Then Return
```

```
nRowSel = this.GetSelectedRow(0)
If nRowSel > 0 Then
    this.SelectRow( nRowSel , FALSE )
End If
this.SelectRow( nRow , TRUE )
```

### PowerBuilder 5

```
Long nRowSel
```

```

If currentrow < 1 Then Return
nRowSel = this.GetSelectedRow(0)
If nRowSel > 0 Then
    this.SelectRow( nRowSel , FALSE )
End If
this.SelectRow( currentrow , TRUE )

```

### Evento ue\_retrieve

Generalmente se crea un evento de usuario en el DataWindow, por ejemplo, 'ue\_retrieve', el cual se encarga de setear el objeto transacción para el DataWindow (si no lo hizo en el evento constructor) y realizar un retrieve de los datos.

Ej.

Long nRows

```

this.SetTransObject( Sqlca )
nRows = this.Retrieve( listadeargumentos )

```

```

If nRows = 0 Then
    // inhabilitar botones, por ejemplo
Else
    // deshabilitar botones
End If

```

```

// Para colorear fila actual
this.TriggerEvent( RowFocusChanged! )

```

## Estructuras

Una estructura es una colección de una o más variables relacionadas del mismo o diferente tipo, agrupadas bajo un nombre único. En algunos lenguajes, tales como Pascal o Cobol, las estructuras se llaman registros.

Las estructuras permiten referenciar entidades relacionadas como una unidad, en lugar de individualmente. Por ejemplo, si se define rut, apellido paterno, apellido materno, nombre, nivel de acceso, y un picture (bitmap) con la foto del empleado, como una estructura llamada user\_struc, todos los datos en la estructura se referencian por medio del nombre user\_struc.

Hay dos tipos de estructuras:


- Estructuras a nivel de objeto, las cuales están asociadas con un tipo particular de objeto tal como una ventana o menú. Estas estructuras son visibles desde cualquier parte del objeto en el que se crearon. Para accederla de otro lugar, se debe incluir el nombre del objeto en el que se creó. Para crear una estructura a nivel de objeto, seleccione Declare – tipodeobjeto Structures...

Por ejemplo.

En una Ventana : seleccione Declare – Window Structures...

En un Menú : seleccione Declare – Menú Structures...

En un objeto de usuario : selecciones Declare User Object Structures..

- Estructuras globales, las cuales no están asociadas con ningún objeto en la aplicación. Estas estructuras se pueden referenciar desde cualquier lugar de la aplicación. Para crear una estructura global, de click al botón  del menú bar.

Ejemplo de estructura, conteniendo información de un usuario.

<i>user_struct</i>	<i>tipo</i>
rut	string
ap_pat	string
ap_mat	string
nombres	string
nivel_acceso	integer
pic_foto	string

## Referenciando una variable de una estructura

Para referenciar una variable de estructura, se usa la notación *punto*.

Ej

```
String cName
cName = user_struct.nombres
```

## Asignando valores a variables de una estructura

Las variables en una estructura son similares a los atributos de un objeto PowerBuilder. Para asignar un valor a una estructura en un script, use la notación punto PowerBuilder.

Por ejemplo, esta sentencia asigna un valor a *nombres* en la estructura *user\_struct*:

```
user_struct.nombres = "Carlos enrique"
```

## Crear instancias de una estructura

Para crear una instancia de una estructura, defina una variable del tipo de la estructura.

Ej.

Supongamos que no queremos usar directamente los valores de una estructura definida , sino que necesitamos crear una instancia separada del mismo tipo. Para crear la instancia se debe definir una variable del tipo de la estructura.

```
user_struct miusuario
```

```
miusuario.nombres = "Patricio"
```

## Objeto de sistema Message

El objeto Message, es un objeto de sistema que se usa para:

- Procesar eventos que no son eventos PowerBuilder.
- Comunicar parámetros entre ventanas cuando se abren o cierran
- Rescatar parámetros opcionales cuando se usan en TriggerEvent o PostEvent.

También se puede crear una versión propia del objeto Message, definiendo una clase user object heredada del objeto Message de sistema.

Para referenciar los atributos del objeto Message, se usa al igual que una estructura, la notación punto.

### Atributos del objeto Message

Los primeros cuatro atributos del objeto Message corresponden a los primeros cuatro atributos de la estructura de mensajes Microsoft Windows.

Atributo	Tipo de dato	Descripción
Handle	Integer	El handle o puntero de la ventana o control
Number	Integer	El número que identifica el evento (este número viene desde Windows).
WordParm	UnsignedInt	El parámetro <i>word</i> para el evento (este parámetro viene desde Windows). El valor del parámetro y su significado están determinados por el evento.
LongParm	Long	El parámetro <i>long</i> para el evento (este número viene desde Windows). El valor del parámetro y su significado están determinados por el evento.
DoubleParm	Double	Un número o variable numérica.
StringParm	String	Un string o variable string.
PowerObjectParm	PowerObject	Cualquier tipo de objeto PowerBuilder, incluyendo estructuras.
Processed	Boolean	Un valor Boolean seteado en el script para el evento definido por el usuario o el evento <i>Other</i> . <ul style="list-style-type: none"> <li>• TRUE – el script procesó el evento (no llama el procedimiento Windows por defecto <i>DefWindowProc</i> después que se procesa el evento.</li> <li>• FALSE – (default) Llama a <i>DefWindowProc</i> después que el evento ha sido procesado.</li> </ul>
ReturnValue	Long	El valor que se desea retornar a la ventana cuando Message.processed es TRUE. Cuando Message.processed es FALSE, este atributo se ignora.



## Paso de información a una ventana

Para abrir una ventana y pasarle información use la función `OpenWithParm`.

### OpenWithParm

#### Descripción

Abre un objeto ventana, la muestra y hace todas sus propiedades y objetos disponibles a los script. También almacena un parámetro en el objeto de sistema `Message`.

#### Sintaxis

`OpenWithParm ( windowvar, parameter {, parent } )`

Argumento	Descripción
<i>windowvar</i>	El nombre de la ventana que se desea abrir.
<i>parameter</i>	El parámetro que se desea almacenar en el objeto <code>Message</code> cuando la ventana se abra. <i>Parameter</i> debe tener uno de estos tipos de datos: <ul style="list-style-type: none"> <li>• String</li> <li>• Numeric</li> <li>• PowerObject</li> </ul>
<i>parent</i>	La ventana que se desea hacer padre de la ventana <i>child</i> , <i>popup</i> o <i>response</i> que se está abriendo. Si se abre una ventana <i>child</i> o <i>popup</i> , y se omite <i>parent</i> , <code>PowerBuilder</code> asocia la ventana que se está abriendo, con la ventana activa.

#### Retorno

Retorna 1 si es exitosa la apertura. -1 si hay error.

#### Uso

El objeto de sistema `Message` tiene tres propiedades para almacenar datos. Dependiendo del tipo de dato del parámetro especificado para `OpenWithParm`, el script en la ventana abierta debería chequear una de las siguientes propiedades:

Propiedad del objeto message	Tipo de dato del argumento
<code>Message.DoubleParm</code>	Númerico
<code>Message.PowerObject</code>	<code>PowerObject</code> (objetos <code>PowerBuilder</code> y estructuras definidas por el usuario).
<code>Message.StringParm</code>	String

En la ventana abierta, es una buena idea acceder inmediatamente (evento `Open` de la ventana) los valores pasados en el objeto `Message`, debido a que otros script pueden usar el objeto `Message` para otros propósitos.

## Evitando referencias a objetos nulos

Cuando se pasa un PowerObject como parámetro, se está pasando una referencia al objeto. El objeto debe existir cuando se referencia más tarde, sino se obtendrá una referencia a un objeto nulo, lo cual producirá un error. Por ejemplo, si se pasa el nombre de un control de una ventana que se está cerrando, ese control no existirá cuando un script accese el parámetro.

## Pasando varios valores como una estructura

Para pasar varios valores, cree una estructura definida por el usuario para mantener los valores y accese la propiedad PowerObjectParm del objeto Message en la ventana que se abre. La estructura es pasada por valor, no por referencia, de este modo, se puede acceder la información incluso si la estructura original ha sido destruida.

### Ej. 1

Esta sentencia abre una instancia de una ventana llamada w\_empleado y almacena un string en Message.StringParm. El script para el evento Open de la ventana usa el parámetro string como texto para el control StaticText llamado st\_empnombre.

*El script que abre la ventana tiene la siguiente sentencia:*

```
OpenWithParm(w_employee, "James Newton")
```

*El script del evento Open de la ventana tiene la siguiente sentencia:*

```
st_empname.Text = Message.StringParm
```

### Ej. 2

Las siguientes sentencias abren una instancia de una ventana del tipo w\_to\_open. Ya que el parámetro es un número, éste se almacena en Message.DoubleParm:

```
w_employee w_to_open
```

```
integer age = 50
```

```
OpenWithParm(w_to_open, age)
```

### Ej. 3

La siguiente sentencia abre una instancia de una ventana child llamada cw\_datos y hace w\_empleado la ventana padre. La ventana w\_empleado debe estar abierta. El parámetro 'Plan de beneficios' es un string, y por lo tanto se almacena en Message.StringParm:

```
OpenWithParm(cw_datos, "Plan de beneficios", w_empleado)
```

## Ej . 4

Las siguientes sentencias, crean una instancia de la estructura user\_struc:

<u>user_struc</u>	<u>tipo</u>
rut	string
ap_pat	string
ap_mat	string
nombres	string
nivel_acceso	integer
pic_foto	string

setean algunos valores de la estructura, pasan la estructura como parámetro almacenado en PowerObjectParm al abrir la ventana w\_empleado. La ventana w\_empleado tiene como variable de instance la variable miusuario del tipo user\_struc. En el script para el evento Open de la ventana se toma el parámetro PowerObjectParm y se deja en la variable miusuario.

*El script que abre la ventana tiene las siguientes sentencias:*

```
user_struc miusuario
```

```
miusuario.rut = "1993487-0"
miusuario.ap_pat = "Araneda"
miusuario.ap_mat = "Lopez"
miusuario.nombres = "Carlos"
miusuario.nivel_acceso = 9
miusuario.pic_foto = "caraneda.bmp"
```

```
OpenWithParm( w_empleado, miusuario, parent )
```

*El script del evento Open de la ventana tiene lassiguientessentencias*

```
miusuario = Message.PowerObjectParm
```

```
st_nombre.text = miusuario.nombres
cRut = miusuario.rut
```

```
dw_empleado.Retrieve( cRut)
```

## SQL Embedded

## SQL Embedded

### Resumen de SQL

Un sistema de administración de base de datos requiere un lenguaje de consulta para habilitar a los usuarios el acceso a los datos. El SQL: **Structured Query Language** ( lenguaje de consulta estructurada ) es el lenguaje usado por la mayoría de los sistemas de base de datos relacionales.

El lenguaje SQL fué desarrollado en un prototipo de sistema de base de datos relacional - System R - por IBM en los años 70.

### Características de SQL

- SQL es un lenguaje como el Inglés. Usa palabras tales como select, insert, delete, como parte de su set de comandos.
- SQL es un lenguaje no-procedural: se especifica *qué* información se requiere; no cómo obtenerla. En otras palabras, SQL no requiere que se especifique el método para accesar los datos. Todas las sentencias ( statements ) SQL usan el optimizador - una parte del RDBMS - para determinar el modo más rápido de recuperar los datos especificados. Esta característica permite que el usuario centre su atención en la obtención del resultado deseado.
- SQL procesa conjuntos de registros más que registros individuales a la vez. La forma más común de un conjunto de registros es una tabla.
- SQL puede ser usado por un rango de usuarios, incluyendo DBAs, aplicaciones, personal de administración, y muchos otros tipos de usuarios finales.
- SQL provee comandos para una variedad de tareas, incluyendo:
  - consulta de datos.
  - insert, update, delete filas en una tabla.
  - crear, modificar y eliminar objetos de la base de datos.
  - controlar acceso a la base de datos y a los objetos en ella.
  - garantizar la consistencia de la base de datos.

Anteriormente, a menudo los sistemas de administración de bases de datos usaban un lenguaje separado para cada una de las categorías anteriores. SQL unifica todas estas tareas en un único lenguaje.

SQL ha llegado a ser de hecho, el lenguaje estandar para bases de datos relacionales. El American National Standards Institute (ANSI) adoptó SQL como el lenguaje estandar para las RDBMS en 1986. El International Standards Organization (ISO) también adoptó SQL como lenguaje estandar para las RDBMS. Las principales RDBMS soportan alguna forma de SQL , y la mayoría de fabricantes de RDBMS intentan ser compatibles con el estandar ANSI.

## SQL Embedded

PowerBuilder permite ejecutar diversas instrucciones, llamadas en su conjunto SQL Embedded (SQL incrustado) como parte de su lenguaje. Estas instrucciones sirven para efectuar acciones en varios tipos de bases de datos relacionales, tales como Sybase, Oracle, DB2, etc. Podemos pensar del Sql Embedded como un lenguaje simbólico SQL, el cual sirve para cualquiera de estas bases de datos; ya que es finalmente traducido a la sentencia real SQL, dependiendo de la base de datos a la que esté conectada la aplicación. Las sentencias SQL embedded pueden ponerse en cualquier script y en cualquier lugar de la aplicación.

Las sentencias SQL embedded que PowerBuilder soporta para envío de SQL dentro de los script, son las siguientes:

<b>Conexión y cierre de transacción</b>	
CONNECT	Conecta a una base de datos especificada.
COMMIT	Actualiza de modo permanente todas las operaciones realizadas a la base de datos desde el último COMMIT, ROLLBACK, o CONNECT para el objeto transacción especificado.
ROLLBACK	Cancela todas las operaciones realizadas a la base de datos desde el último COMMIT, ROLLBACK, o CONNECT para el objeto transacción especificado
DISCONNECT	Desconecta de la base de datos especificada.
<b>Operaciones sobre la base de datos</b>	
SELECT	Obtiene una fila única desde las tablas especificadas.
INSERT	Inserta una fila en la tabla especificada.
DELETE	Elimina las filas especificadas en la sentencia DELETE.
UPDATE	Actualiza la fila especificada en la sentencia SELECT.
<b>Cursores</b>	
DECLARE CURSOR	Declara el cursor especificado.
OPEN	Causa que el cursor ejecute la sentencia SELECT especificada cuando se declaró el cursor.
FETCH	Avanza a la siguiente fila del cursor. No se permite usar aquí la cláusula USING TransactionObject. El objeto transacción se declara junto con la declaración del cursor o procedimiento.
DELETE Where Current of Cursor	Elimina la fila en la que está posicionado el cursor.
UPDATE Where Current of	Actualiza la fila en la que está posicionado el cursor.

---

Cursor	
CLOSE CURSOR	Cierra el cursor especificado.
<b>Llamada a procedimientos almacenados</b>	
DECLARE PROCEDURE	Declara el procedimiento almacenado especificado.
EXECUTE	Ejecuta el procedimiento almacenado en la base de datos.
FETCH	Ver uso de Fetch en procedimientos más adelante.
CLOSE PROCEDURE	Cierra el procedimiento almacenado especificado.



---

## Conexión y cierre de transacción

### CONNECT

Conecta a una base de datos específica. Esta sentencia debe ser ejecutada antes que cualquier acción (tal como insert, update, o delete) sea procesada usando el objeto transacción por defecto o el objeto transacción especificado.

Sintaxis

CONNECT {USING *TransactionObject*} ;

Parámetro	Descripción
<u><i>TransactionObject</i></u>	El nombre del objeto transacción que contiene la información de conexión requerida para la base de datos a la cual se desea conectar. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia CONNECT.

---

Ej. 1

Esta sentencia conecta con la base de datos especificada en el objeto transacción por defecto SQLCA.

CONNECT ;

Es equivalente a :

Connect Using Sqlca;

Ej. 2

Esta sentencia conecta a la base de datos especificada en el objeto transacción llamado emp\_tran.

Connect Using Emp\_tran ;



---

## COMMIT

Actualiza de modo permanente todas las operaciones realizadas a la base de datos desde el último COMMIT, ROLLBACK, o CONNECT para el objeto transacción especificado. COMMIT no causa una desconexión, pero cierra todos los cursores y procedimientos abiertos. (note que la sentencia DISCONNECT realiza un COMMIT antes de desconectar de la base de datos ).

Sintaxis

COMMIT {USING *TransactionObject*} ;

Parámetro	Descripción
<u><i>TransactionObject</i></u>	El nombre del objeto transacción para el cual se desea actualizar de modo permanente todas las operaciones a la base de datos desde el último commit, rollback o connect. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia COMMIT.

---

Ej. 1

Esta sentencia hace commit a todas las operaciones para la base de datos especificada en el objeto transacción por defecto SQLCA.

COMMIT ;

Es equivalente a:

Commit Using Sqlca;

Ej.

Esta sentencia hace commit a todas las operaciones especificadas en el objeto transacción llamado emp\_tran.

Commit Using Emp\_tran ;

---

## ROLLBACK

Cancela todas las operaciones realizadas a la base de datos desde el último COMMIT, ROLLBACK, o CONNECT para el objeto transacción especificado. ROLLBACK no causa una desconexión, pero cierra todos los cursores y procedimientos abiertos.

Sintaxis

ROLLBACK {USING *TransactionObject*} ;

Parámetro	Descripción
<u><i>TransactionObject</i></u>	El nombre del objeto transacción para el cual se desea cancelar todas las operaciones realizadas a la base de datos desde el último commit, rollback o connect. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia ROLLBACK.

---

Ej. 1

Esta sentencia cancela todas las operaciones a la base de datos especificada en el objeto transacción por defecto.

ROLLBACK ;

Es equivalente a:

Rollback Using Sqlca;

Ej. 2

Esta sentencia cancela todas las operaciones a la base de datos especificada en el objeto transacción emp\_tran.

ROLLBACK USING emp\_tran ;

---

## DISCONNECT

Ejecuta una sentencia COMMIT para el objeto transacción especificado y luego desconecta de la base de datos especificada en el objeto transacción.

Sintaxis

DISCONNECT {USING *TransactionObject*} ;

Parámetro	Descripción
<u><i>TransactionObject</i></u>	El nombre del objeto transacción que identifica la base de datos de la cual se desea hacer Commit y luego desconectar. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia DISCONNECT.

---

Ej. 1

Esta sentencia desconecta de la base de datos especificada en el objeto transacción por defecto.

DISCONNECT ;

Es equivalente a:

Disconnect Using Sqlca;

Ej. 2

Esta sentencia desconecta de la base de datos especificada en el objeto transacción llamado emp\_tran..

DISCONNECT USING Emp\_tran ;

---

## SELECT

Obtiene una fila de las tablas especificadas en *RestOfSelectStatement*. Si la sentencia SELECT obtiene más de una fila se produce un error.

Sintaxis

```
SELECT RestOfSelectStatement
{USING TransactionObject} ;
```

Parámetro	Descripción
<u><i>RestOfSelectStatement</i></u>	El resto de la sentencia SELECT (la lista de columnas INTO, FROM, WHERE, y otras cláusulas).
<u><i>TransactionObject</i></u>	El nombre del objeto transacción que identifica la base de datos que contiene las tablas. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

## Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia SELECT.

---

Ej.

Las siguientes sentencias obtienen datos de las columnas emp\_lname y emp\_fname de una fila en la tabla employee y los ponen en los SingleLineEdit sle\_lname y sle\_fname. El objeto transacción usado es emp\_tran.

```
Int    Emp_num
```

```
Emp_num = Integer(sle_Emp_Num.Text)
```

```
SELECT Emp_LName, Emp_FName
      INTO :sle_LName.text, :sle_FName.text
      FROM Employee
      WHERE Emp_nbr = :Emp_num
      USING Emp_tran ;
```

```
If Emp_tran.SQLCode = 100 then
    MessageBox("Solicitud de Empleado", "Empleado no existe. ")
Elseif Emp_tran.SQLCode > 0 then
    MessageBox("Error ", Emp_tran.SQLErrMsg, Exclamation!)
End If
```

---

## INSERT

Inserta una o más filas en la tabla especificada en *RestOfInsertStatement*.

Sintaxis

```
INSERT RestOfSelectStatement  
{USING TransactionObject} ;
```

Parámetro	Descripción
<u><i>RestOfSelectStatement</i></u>	El resto de la sentencia INSERT (la cláusula INTO, lista de columnas y valores.
<u><i>TransactionObject</i></u>	El nombre del objeto transacción que identifica la base de datos que contiene la tabla. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia INSERT.

---

Ej. 1

Esta sentencia inserta una fila con los valores *EmpNbr* y *EmpName* en las columnas *Emp\_nbr* y *Emp\_name* de la tabla *Employee*, identificada en el objeto transacción por defecto.

```
Int    EmpNbr  
String EmpName  
...  
INSERT INTO Employee (Emp_nbr, Emp_name)  
VALUES (:EmpNbr, :EmpName) ;
```

Ej. 2

Estas sentencias insertan una fila con los valores entrados en el *SingleLineEdit* *sle\_number* y *sle\_name* en las columnas *Emp\_nbr* y *Emp\_name* de la tabla *Employee* especificada en el objeto transacción *Emp\_tran*.

```
Int          EmpNbr  
  
EmpNbr = Integer(sle_number.Text)  
INSERT INTO Employee (Emp_nbr, Emp_name)
```

---

VALUES (:EmpNbr, :sle\_name.Text) USING Emp\_tran ;



---

## DELETE

Elimina las filas en *TableName* especificadas por *Criteria*.

Sintaxis

```
DELETE FROM TableName WHERE Criteria  
{USING TransactionObject} ;
```

Parámetro	Descripción
<u><i>TableName</i></u>	El nombre de la tabla en la que se desea eliminar filas.
<u><i>Criteria</i></u>	Criterio que especifica qué filas eliminar.
<u><i>TransactionObject</i></u>	El nombre del objeto transacción que identifica la base de datos que contiene la tabla. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia DELETE.

---

Ej. 1

Esta sentencia elimina las filas de la tabla Employee en la base de datos especificada en el objeto transacción por defecto, donde Emp\_num es menor que 100.

```
DELETE FROM Employee WHERE Emp_num < 100 ;
```

Ej. 2

Estas sentencias eliminan las filas en la tabla Employee en la base de datos nombrada en el objeto transacción Emp\_tran, para Emp\_num igual al valor entrado en el SingleLineEdit sle\_number.

```
int    Emp_num
```

```
Emp_num = Integer(sle_number.Text)  
DELETE FROM Employee  
WHERE Employee.Emp_num = :Emp_num ;
```

---

## Cursores

Una sentencia SELECT retorna 0 o más filas. Si la sentencia SELECT retorna varias filas, éstas se pueden manipular individualmente usando cursores.

Un cursor es un nombre simbólico que está asociado con una sentencia SELECT a través de una sentencia de declaración. Consiste de las siguientes partes:

- **Conjunto resultado del cursor** – el set de filas resultante de la ejecución de un query asociado al cursor.
- **Posición del cursor** – un puntero a una fila dentro del conjunto resultado del cursor.

Un ejemplo simple nos permitirá visualizar los distintos elementos de un cursor:

Ejemplo

```
// Declaración del cursor
DECLARE emp_curs CURSOR FOR
SELECT emp_nombre FROM EMPLEADO
WHERE ciudad = :sle_1.text;

// Declara una variable destino para el nombre de un empleado.
String emp_nombre_var

// Abrir cursor
OPEN emp_curs;

// Posiciona el cursor en la primera fila del conjunto resultado,
// extrae emp_nombre y lo pone en la variable emp_nombre_var
FETCH emp_curs INTO :emp_nombre_var;

// Recorre el conjunto resultado hasta el final
DO WHILE SQLCA.sqlcode = 0
    // Pone un mensaje con el nombre del empleado
    MessageBox("Empleado",emp_nombre_var)
    // Avanza el cursor a la siguiente posición del conjunto resultado.
    FETCH emp_curs INTO :emp_nombre_var;
LOOP

// Cierra el cursor
CLOSE emp_curs;
```

La posición del cursor indica la fila actual del cursor. Se puede explícitamente eliminar o modificar la fila en la que está posicionado el cursor usando las sentencias:

DELETE Where Current of Cursor  
UPDATE Where Current of Cursor

La posición del puntero del cursor se cambia usando una operación llamada FETCH. Un FETCH mueve la posición actual del cursor, una o más filas adelante.

En general, los cursores sólo avanzan (secuenciales), y una vez que llegan al final del conjunto resultado no es posible retroceder para accederlo nuevamente.

Los cursores permiten recorrer un conjunto resultado fila a fila, de modo similar a como un lenguaje de programación recorre un archivo registro a registro.

Después de declarar el cursor, éste puede adoptar dos estados:

- Cerrado – El conjunto resultado no existe y no es posible leer información de él. Los cursores están inicialmente en este estado. Para usarlo se debe abrir (OPEN) explícitamente. Una vez abierto, se puede cerrar en cualquier momento.
- Abierto – Las filas dentro del conjunto resultado del cursor están disponibles para lectura o actualización.

Se puede cerrar un cursor y luego reabrirlo. Reabrir un cursor, recrea el conjunto resultado del cursor y posiciona el cursor antes de la primera fila. Esto permite procesar el conjunto resultado de un cursor las veces que sea necesario. El cursor se puede cerrar en cualquier momento si no es necesario recorrerlo completo.

Todas las operaciones, como avanzar el cursor o actualizar la fila, son relativas a la fila actual del cursor. Las actualizaciones a la fila del cursor involucran cambios en los datos o eliminación. No se puede usar cursores para insertar filas directamente. Las actualizaciones afectan a las filas de las tablas bases incluidas en la declaración del cursor.

---

Se describen a continuación los distintos elementos presentes en el uso de un cursor:

## DECLARE CURSOR

### Sintaxis

```
DECLARE CursorName CURSOR FOR SelectStatement  
{USING TransactionObject} ;
```

Parámetro	Descripción
<i>CursorName</i>	Cualquier nombre válido PowerBuilder.
<i>SelectStatement</i>	Cualquier sentencia válida SELECT.
<i>TransactionObject</i>	El nombre del objeto transacción para el que se desea declarar el cursor. Esta cláusula se requiere sólo para objetos transacción que no sean el defecto SQLCA.

### Descripción

Declara un cursor para el objeto transacción especificado. DECLARE cursor no es un comando ejecutable, y es análogo a declarar una variable.

Para declarar un cursor global, shared, o instance, seleccione Declare - Global Variables, Declare - Shared Variables, o Declare - Instance Variables en la Ventana, Objeto de usuario, Menú, o Script PowerBuilder. Para declarar un cursor local puede ayudarse dando click al botón Paint SQL disponible al escribir un script.

Ej.

Este ejemplo, declara el cursor llamado Emp\_cur para la base de datos especificada en el objeto transacción. También referencia la variable Sal\_var, la que debe estar seteada a un valor apropiado antes de ejecutar el comando OPEN Emp\_cur.

```
DECLARE Emp_cur CURSOR FOR  
SELECT emp_number, emp_name  
FROM employee  
WHERE emp_salary > :Sal_var ;
```

---

## OPEN

### Sintaxis

OPEN *CursorName* ;

Parámetro	Descripción
<i>CursorName</i>	El nombre del cursor que se desea abrir.

### Descripción

Causa que se ejecute el comando SELECT especificado cuando se declaró el cursor. La cláusula USING TransactionObject no se permite con OPEN; el objeto transacción fue especificado en la sentencia que declaró el cursor.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia OPEN.

---

Ej.

Esta sentencia abre el cursor Emp\_curs.

OPEN Emp\_curs ;

## FETCH

### Sintaxis

FETCH *Cursor / Procedure* INTO *HostVariableList* ;

Parámetro	Descripción
<i>Cursor</i> o <i>Procedure</i>	El nombre del cursor o procedimiento almacenado al que se desea hacer un Fetch.
<i>HostVariableList</i>	Variables PowerScript en los que se almacenarán los datos de la fila del cursor.

### Descripción

Toma la fila después de la que está posicionado el cursor o procedimiento.

---

Si el motor de base de datos soporta otros formatos es posible usar FETCH NEXT, FETCH FIRST, FETCH PRIOR, o FETCH LAST.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia FETCH.

---

#### Ej.1

Esta sentencia toma datos obtenidos por la cláusula SELECT en la declaración del cursor llamado Emp\_cur y los pone en Emp\_num y Emp\_name.

```
Int Emp_num  
String Emp_name
```

```
FETCH Emp_cur INTO :Emp_num, :Emp_name ;
```

#### Ej. 2

Si sle\_emp\_num y sle\_emp\_name son SingleLineEdits, estas sentencias toman datos del cursor Emp\_cur, almacenan los datos en Emp\_num y sle\_emp\_name, y convierte Emp\_num desde un integer a un string, poniéndolo en sle\_emp\_num.

```
Int Emp_num
```

```
FETCH Emp_cur into :emp_num, :sle_emp_name.Text ;  
sle_emp_num.Text = string(Emp_num)
```

### DELETE Where Current Of Cursor

#### Sintaxis

```
DELETE FROM TableName WHERE CURRENT OF CursorName ;
```

Parámetro	Descripción
<i>TableName</i>	El nombre de la tabla en la que se desea eliminar una fila.
<i>CursorName</i>	El nombre del cursor en el cual se especificó la tabla.

#### Descripción

---

Elimina la fila en la cuál está posicionado el cursor. No todas las DBMS suportan DELETE Where Current of Cursor.

---

Ej.

Esta sentencia elimina en la tabla Employee la fila en la cual está posicionado el cursor Emp\_cur.

```
DELETE FROM Employee WHERE current of Emp_curs ;
```

## UPDATE Where Current of Cursor

Sintaxis

```
UPDATE TableName SetStatement  
WHERE CURRENT OF CursorName;
```

Parámetro	Descripción
<i>TableName</i>	El nombre de la tabla en la que se desea actualizar la fila.
<i>SetStatement</i>	La palabra SET seguida por una lista separada por comas de la forma <i>ColumnName=value</i> .
<i>CursorName</i>	El nombre del cursor en el cual se referencia la tabla.

Descripción

Actualiza la fila en la cual el cursor está posicionado, usando los valores en *SetStatement*.

Ej.

Esta sentencia actualiza la fila en la tabla Employee en la cual está posicionado el cursor Emp\_curs.

```
UPDATE Employee  
SET salary = 17800  
WHERE CURRENT of Emp_curs ;
```

## CLOSE Cursor

Sintaxis

```
CLOSE CursorName ;
```

Parámetro	Descripción
-----------	-------------



---

<i>CursorName</i>	El nombre del cursor que se desea cerrar.
-------------------	---

---

### Descripción

Cierra el cursor *CursorName*; finaliza proceso de *CursorName*.

CLOSE a menudo se usa cuando SQLcode = 100 (no encontró) después de un FETCH.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia CLOSE.

---

Ej.

Esta sentencia cierra el cursor Emp\_cursor.

CLOSE Emp\_cursor ;

## Llamada a procedimientos almacenados

### *Procedimientos almacenados*

Los procedimientos almacenados (Stored Procedures) son una colección de sentencias SQL y lenguaje de control de flujo grabados en la base de datos. Se crean para ejecutar procesamiento directo en la base de datos liberando recursos del cliente. Un procedimiento almacenado puede ejecutar sólo procesamiento, obtención de información, o una mezcla de ambas operaciones. En este capítulo veremos el modo en que se llaman desde PowerBuilder.

Los procedimientos almacenados pueden :

- Recibir parámetros.
- Llamar a otros procedimientos almacenados.
- Retornar un conjunto resultado (filas).
- Retornar un valor que indica éxito o falla. Retornar un mensaje de error.
- Retornar valores en parámetros *output*.
- Ejecutarse en servidores remotos.

Los procedimientos almacenados mejoran la potencia, eficiencia, y flexibilidad de SQL y aumentan dramáticamente la performance de sentencias SQL o batch.

Los procedimientos almacenados difieren de las sentencias ordinarias SQL o batch, en que los procedimientos son pre-compilados. La primera vez que se corre un procedimiento, el servidor lo analiza y prepara un plan de ejecución que es almacenado en las tablas de sistema. Posteriormente, el procedimiento se ejecuta de acuerdo a este plan. Debido a que la mayor parte del trabajo de procesamiento del query ya ha sido ejecutado, los procedimientos almacenados se ejecutan velozmente.

Sybase SQL Server, proporciona una variedad de procedimientos almacenados llamados 'system procedures', los cuales son convenientes herramientas para el usuario.

### *Llamada de procedimientos desde PowerBuilder*

Con el objeto de clarificar los aspectos de comunicación entre un procedimiento almacenado y su llamada desde PowerBuilder, se usarán dos ejemplos clarificadores:

- el primero, considera un SELECT simple en el procedimiento, y como capturar su conjunto resultado y valor de retorno.

- el segundo agrega parámetros de tipo *output* , de tal modo de obtener el conjunto resultado por una parte, y luego obtener los valores calculados de tipo *output*, además del valor de retorno.

**Caso 1** – Llamada a un proceso que realiza una sentencia SELECT.

Procedimiento almacenado1

```
Create procedure sp_deptoinfo
( @depto integer )
as
Select ap_paterno, ap_materno, nombres, salario
From empleado
Where depto_id = @depto
Return 0
```

Llamada PowerBuilder1-1

```
Long ll_depto, nRet
String ls_appat, ls_apmat, ls_nombres
Int li_procesados

ll_depto = 100
li_procesados = 0

DECLARE c_deptoinfo PROCEDURE FOR @nRet = sp_deptoinfo
@depto = :ll_depto
USING Sqlca;
EXECUTE c_deptoinfo;
CHOOSE CASE SQLCA.sqlcode
CASE 0
    // éxito. Hay al menos un conjunto resultado. Loop a través de él.
    DO
        FETCH c_deptoinfo INTO :ls_appat, :ls_apmat, :ls_nombres;
        CHOOSE CASE SQLCA.sqlcode
        CASE 0
            // Procesar datos
            li_procesados++
        CASE 100
            // Exito. no hay más datos
            MessageBox( "Fin", String(Sqlca.sqldbcode) + "," + sqlca.sqlerrtext)
        CASE -1
            MessageBox("Falló Fetch", string(sqlca.sqldbcode) + "," +
sqlca.sqlerrtext)
        END CHOOSE
    LOOP WHILE sqlca.sqlcode = 0
CASE 100
```

```
        // Exito. No hay más datos.
CASE ELSE
    MessageBox( "Ejecución falló", string(sqlca.sqldbcode) + " " +
sqlca.sqlerrtext)
END CHOOSE

// Si se efectúa un FETCH adicional, se obtiene el valor que retorna el
procedimiento,
// en este caso, 0 (return 0). Esto sólo es correcto si no se produjo un error.
FETCH c_deptoinfo INTO :nRet;

CLOSE c_deptoinfo;
```

En este ejemplo, se tratan exhaustivamente los tres valores que puede retornar sqlca.sqlcode:

0 : éxito de la ejecución.  
100 : no más conjunto resultado  
-1 : se produjo un error

Además, se muestra como obtener el valor retornado por la sentencia *return* (valor) en el procedimiento almacenado. En la declaración PowerBuilder, debe ir una variable numérica que almacenará este valor, y debe anteponerse el signo arroba (@). Para obtener el valor retornado debe ejecutarse un último fetch. En el segundo ejemplo, veremos que un último fetch, trae también los valores de parámetros de tipo *output* definidos en el procedimiento almacenado.

En general y dependiendo de lo que se quiera obtener, el tratamiento de los valores de sqlca.sqlcode puede variar.

Por ejemplo, en la siguiente llamada desde PowerBuilder al mismo procedimiento, se opta por simplificar este tratamiento, y sólo se toma en cuenta que no exista error, no importando que no exista conjunto resultado, ni la obtención del retorno del procedimiento.

#### Llamada PowerBuilder1-2

Long ll\_depto  
String ls\_appat, ls\_apmat, ls\_nombres  
Int li\_procesados

ll\_depto = 100  
li\_procesados = 0

DECLARE c\_deptoinfo PROCEDURE FOR sp\_deptoinfo  
@depto = :ll\_depto  
USING Sqlca;

EXECUTE c\_deptoinfo;

FETCH c\_deptoinfo INTO :ls\_appat, :ls\_apmat, :ls\_nombres;

DO WHILE sqlca.sqlcode = 0  
    // procesar datos  
    li\_procesados++  
    .  
    .  
    .  
    // siguiente conjunto de datos

```
        FETCH c_deptoinfo INTO :ls_appat, :ls_apmat, :ls_nombres;  
LOOP  
  
CLOSE c_deptoinfo;
```

**Caso 2** – Llamada a un proceso que realiza una sentencia SELECT, obtiene la suma y promedio del salario de un departamento usando variables output, y recupera el valor retornado por el procedimiento.

Procedimiento almacenado2

```
Create procedure sp_deptinfo  
( @depto integer ,  
  @totsal double precision output,  
  @avgsal double precision output )
```

as

```
Declare @num_de_emp integer
```

```
Select ap_paterno, ap_materno, nombres, salario  
From empleado  
Where depto_id = @depto
```

```
Select @totsal      = sum(salario),  
       @avgsal      = avg(salario),  
       @num_de_emp = count(*)  
From empleado  
Where depto_id = @depto
```

```
Return @num_de_emp
```

Llamada PowerBuilder2-1

```
Long   ll_depto, nRet  
String ls_appat, ls_apmat, ls_nombres  
Int     li_procesados  
Double  dSalario, dTotSal, dAvgSal
```

```
ll_depto = 100  
li_procesados = 0
```

```
DECLARE c_deptinfo PROCEDURE FOR @nRet = sp_deptinfo  
@depto  = :ll_depto,  
@totsal = :dTotSal OUT,  
@avgsal = :dAvgSal OUT  
USING Sqlca;  
EXECUTE c_deptinfo;  
CHOOSE CASE SQLCA.sqlcode  
CASE 0  
    // éxito. Hay al menos un conjunto resultado. Loop a través de él.  
DO
```

```

    FETCH c_deptoinfo INTO :ls_appat, :ls_apmat, :ls_nombres;
    CHOOSE CASE SQLCA.sqlcode
    CASE 0
        // Procesar datos
        li_procesados++
    CASE 100
        // Exitó. no hay más datos
        MessageBox( "Fin", String(Sqlca.sqldbcode) + "," + sqlca.sqlerrtext)
    CASE -1
        MessageBox("Falló Fetch", string(sqlca.sqldbcode) + "," +
sqlca.sqlerrtext)
    END CHOOSE
    LOOP WHILE sqlca.sqlcode = 0

    // Si se efectúa un FETCH adicional, se obtiene el valor que retorna el
    procedimiento,
    // (el número de empleados ) y los valores de output (suma y promedio de
    salarios).
    FETCH c_deptoinfo INTO :nRet, :dTotSal, :dAvgSal;
    CHOOSE CASE SQLCA.sqlcode
    CASE 0
        // éxito
        MessageBox( "Retorno y valores de output", &
            string(nRet) + "~n~r" + &
            "Salario Total : " + String(dTotSal) + "~n~r" + &
            "Salario Promedio: " + String(dAvgSal) )
    CASE 100
        MessageBox( "Retorno y valores de output", "No generados." )
    CASE ELSE
        MessageBox( "Retorno y valores de output", "Falló~n~r" + &
            "SQLDBCode : " + String(SQLCA.sqldbcode) + "~n~r" +
&
            "Error : " + SQLCA.sqlerrtext)
    END CHOOSE
    CLOSE c_deptoinfo;

CASE 100
    // Exitó. No hay más datos.
CASE ELSE
    MessageBox( "Ejecución falló", string(sqlca.sqldbcode) + "," +
sqlca.sqlerrtext)
END CHOOSE

```



Como en el ejemplo anterior, el tratamiento puede ser más simple:

Llamada PowerBuilder2-2

```
Long    ll_depto, nRet
String  ls_appat, ls_apmat, ls_nombres
Int     li_procesados
Double  dSalario, dTotSal, dAvgSal
```

```
ll_depto = 100
li_procesados = 0
```

```
DECLARE c_deptoinfo PROCEDURE FOR @nRet = sp_deptoinfo
@depto  = :ll_depto,
@totals = :dTotSal OUT,
@avgsal = :dAvgSal OUT
USING Sqlca;
```

```
EXECUTE c_deptoinfo;
```

```
FETCH c_deptoinfo INTO :ls_appat, :ls_apmat, :ls_nombres;
```

```
DO WHILE sqlca.sqlcode = 0
    // procesar datos
    li_procesados++
    .
    .
    .
    // siguiente conjunto de datos
    FETCH c_deptoinfo INTO :ls_appat, :ls_apmat, :ls_nombres;
LOOP
```

```
// Si se efectúa un FETCH adicional, se obtiene el valor que retorna el
procedimiento,
// (el número de empleados ) y los valores de output (suma y promedio de
salarios).
```

```
FETCH c_deptoinfo INTO :nRet, :dTotSal, :dAvgSal;
```

```
CLOSE c_deptoinfo;
```

```
//... tratamiento de los valores obtenidos con el último FETCH.
```

```
.....
```

## Uso de la sentencia RAISERROR en procedimientos almacenados

Dentro de un procedimiento, se usa RAISERROR para generar un error similar a los generados por el sistema. Raiserror setea un flag de sistema para grabar el hecho de haber ocurrido un error.

La sintaxis básica es la siguiente:

```
RAISERROR error_number [{format_string | @local_variable}]
```

El número *error\_number* queda en la variable global @@error, la cual almacena el número de error más recientemente generado por Sybase SQL Server. Los números de error para mensajes de error definidos por el usuario deben ser mayores que 20.000.

Para indicar un mensaje de error definido por el usuario, se puede emplear un string puesto directamente entre comillas, o bien usar el contenido de una variable local. La longitud de *format\_string* está limitada a 255 caracteres. Si se usan variables locales, éstas deben ser de tipo *char* o *varchar*. El *format\_string* o @*local\_variable*, es opcional. Si no se incluye, SQL Server usa el mensaje correspondiente al número *error\_number* desde sysusermessage, en el lenguaje por defecto.

En el cliente PowerBuilder, el parámetro *error\_number*, es capturado desde el atributo SQLDBCode del objeto transacción que se esté usando. El mensaje de error generado, es capturado desde el atributo SQLErrText del objeto transacción.

Ej.

### Procedimiento almacenado

```
.
If @@error <> 0
Begin
    RaisError 20001 "Error al eliminar agencia."
    Rollback tran
    Return -100
End
```

### PowerBuilder

```
.
execute del_agencia;
If sqlca.sqlcode = -1 Then
    MessageBox( "Error", "Sqlldbcod = " + String(sqlca.sqlldbcod) + "~n~r" + &
        "SqlErrText = " + sqlca.sqlerrtext )
End If
```



Se describen a continuación los distintos elementos presentes en el uso de llamadas a procedimientos almacenados.

## DECLARE PROCEDURE

Sintaxis

```
DECLARE ProcedureName PROCEDURE FOR StoredProcedureName  
@Param1=Value1, @Param2=Value2,...  
{USING TransactionObject} ;
```

Parámetro	Descripción
<i>ProcedureName</i>	Cualquier nombre válido PowerBuilder.
<i>StoredProcedureName</i>	El nombre de un proceso almacenado.
<i>me</i>	
@ <i>paramX</i> <i>valueX</i>	= El nombre de un parámetro (argumento) definido en el procedimiento almacenado, y una expresión válida PowerBuilder. X representa el número del parámetro y el valor en la sintaxis.
<i>TransactionObject</i>	El nombre del objeto transacción para el que se desea declarar el procedimiento. Esta cláusula se requiere sólo si el objeto transacción no es el por defecto, SQLCA.

Descripción

Declara un procedimiento para el objeto transacción especificado. DECLARE PROCEDURE es un comando no ejecutable. Es análogo a declarar una variable.

En PowerBuilder, se puede usar la palabra reservada opcional OUT para indicar una variable tipo *output* de un procedimiento almacenado en Sybase SQL Server.

@param = :Valor **OUT**

Para declarar un procedimiento global, shared, o instance, seleccione Declare – Global Variables, Declare – Shared Variables, o Declare – Instance Variables, en la ventana, objeto de usuario, menú, o script.

Para declarar un procedimiento global, shared, o instance, seleccione Declare - Global Variables, Declare - Shared Variables, o Declare - Instance Variables en la Ventana, Objeto de usuario, Menú, o Script PowerBuilder. Para declarar un procedimiento local puede ayudarse dando click al botón Paint SQL disponible al escribir un script.

### Ej. 1

Esta sentencia declara el procedimiento Sybase SQL server Emp\_proc para la base de datos especificada en el objeto transacción por defecto. Referencia las variables Emp\_name y Emp\_sal, las que deben setearse a valores apropiados antes de ejecutar el comando EXECUTE Emp\_proc.

```
DECLARE Emp_proc procedure for GetName  
@emp_name = :Emp_name_var,  
@emp_salary = :Emp_sal_var ;
```

### Ej. 2

Esta sentencia declara el procedimiento almacenado ORACLE Emp\_proc para la base de datos especificada en el objeto transacción tran\_oracle. Referencia las variables Emp\_name y Emp\_sal, las que deben setearse a valores apropiados antes de ejecutar el comando EXECUTE Emp\_proc.

```
DECLARE Emp_proc procedure for GetName (:Emp_name_var, :Emp_sal_var)  
USING tran_oracle;
```

## EXECUTE

### Sintaxis

```
EXECUTE ProcedureName ;
```

Parámetro	Descripción
<i>ProcedureName</i>	El nombre asignado en la sentencia DECLARE del procedimiento almacenado que se desea ejecutar. El procedimiento debe haber sido declarado previamente. <i>ProcedureName</i> no es necesariamente el nombre del procedimiento almacenado en la base de datos.

### Descripción

Ejecuta el procedimiento previamente declarado identificado por *ProcedureName*. La cláusula USING TransactionObject no se permite con EXECUTE; el objeto transacción fue especificado en la sentencia que declara el procedimiento.

### Nota

---

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia EXECUTE.

---

Ej. Esta sentencia ejecuta el procedimiento almacenado identificado por Emp\_proc.

```
EXECUTE Emp_proc ;
```

## CLOSE PROCEDURE

Sintaxis

CLOSE *ProcedureName* ;

Parámetro	Descripción
<i>ProcedureName</i>	El nombre asignado en la sentencia DECLARE del procedimiento almacenado que se desea cerrar.

### Descripción

Cierra el procedimiento *ProcedureName*; finaliza el procesamiento de *ProcedureName*. Esta sentencia debe ser precedida por una sentencia EXECUTE para el mismo procedimiento. La cláusula USING *TransactionObject* no se permite con CLOSE; el objeto transacción fue especificado en la sentencia que declara el procedimiento.

Sólo se necesita usar CLOSE para cerrar procedimientos que retornan conjuntos resultados o filas. PowerBuilder cierra automáticamente procedimientos que no retornan conjuntos resultados (y setea el código de retorno a 100).

CLOSE aparece a menudo en los script que se ejecutan cuando sqlca.sqlcode después de un fetch es igual a 100 (no hay más filas).

### Nota

Es buena práctica testear el éxito / falla del código después de ejecutar la sentencia CLOSE.

## Tips



## Aplicación

1.- Al dar click derecho a la barra de menú de una aplicación, los textos de opciones del menú aparecen en inglés. Para corregir esto, en el evento Open de la aplicación codifique:

```
// Toolbar  
this.toolbarframetitle = "Barra de Menú"  
this.ToolBarPopupMenuText = "Izquierda, Arriba, Derecha, Abajo, Flotante, Ver  
Texto, Ver Tips"
```

## 2.- Terminar una aplicación

En el evento Close de la aplicación:

- a) eliminar cualquier instancia de objeto creada con CREATE.
- b) desconectar cualquier conexión a la base de datos

```
destroy userobj1;  
destroy userobj2;  
disconnect;
```

disconnect ejecuta un COMMIT al objeto transacción específico y luego desconecta la aplicación de la base de datos.

Desde el script del menú-salir poner HALT CLOSE.

HALT, termina una aplicación inmediatamente, en cambio HALT CLOSE, ejecuta primero el evento Close de la aplicación.

---

## Controles

### 1.- Basura en DropDownListBox.

En un evento externo al DropDownListBox

OJO. Si no se ha seleccionado nada en un dropdownlistbox, al obtener el texto a veces pasa que tiene basura. Lo mejor es buscar el texto que trae para asegurar que sí está seleccionado.

```
ori_emb = ddlb_emb.text           // Puede traer basura
nRowSel = ddlb_emb.FindItem(ori_emb,0)    // Buscarlo
If nRowSel = 0 Then
    ori_emb = ""
End If
```

// Otros script en un DropDownListBox

Constructor

-----

```
this.text = ""
```

SelectionChanged

-----

```
this.PostEvent( 'ue_postSelec' )
```

ue\_postSelec

-----

Asegurar que se hizo un cambio efectivo. Esto, pues si no está seleccionado algo y luego se selecciona algo y se da el foco a otro objeto no seleccionando nada, el texto queda con basura, generalmente un string largo. Así, si se conoce el largo de cada selección, se propone el siguiente código.

String cThis

```
cThis = this.text
```

```
If LEN(cThis) > 3 Then
```

```
    this.text = ""
```

```
Else
```

```
    -- un cambio real
```

```
    -- otros seteos
```

```
End If
```

### 2.- Index del ddlb

## Tips

---

```
// Retorna el Index del ddlb.  
ddlb_tipo_consulta.FindItem( ddlb_tipo_consulta.text, 0 ) -> Int
```

### 3.- ListBox

En evento selectionChanged

```
Int    nSel  
String cSel
```

```
nSel = this.SelectedIndex()  
cSel = this.text( nSel )
```

### 4.- Seleccionar un ítem de un ListBox

```
listboxname.SelectItem ( itemnumber )  
Selecciona el ítem itemnumber. 0 deselecciona.
```

---

## General

1.- Enviar nombre de ventana a una función desde un objeto

```
f_funcion( Parent.ClassName() )
```

2.- Objeto PowerBuilder cpmeter.

Para usar el objeto de clase "cpmeter" , de tal forma que indique el % de filas recuperadas mientras se ejecuta Retrieve() en un datawindow control.

- Copiar el objeto de usuario uo\_3d\_meter desde los ejemplos PowerBuilder a una librería de su aplicación.
- Copiar cpallette.dll al directorio de su aplicación.
- El dataWindow está en una ventana distinta a la ventana del "cpmeter".
- Crear las variables Instance siguientes para la ventana del dataWindow control:

```
long nCuenta
int  nCompleto
```

- Crear una nueva ventana, ( w\_meter ) y poner el objeto de usuario "cpmeter" en ella. Dele las dimensiones deseadas. Setear ventana tipo Main, sin "Control Menu", no Resizable, no Maximize Box, no Minimize Box.

No hay código en esta ventana.

- En el script del evento RetrieveStart del datawindow control poner lo siguiente :

```
open( w_meter )
```

- En el script del evento RetrieveRow del dataWindow control poner lo siguiente :

```
nCuenta = nCuenta + 1
nCompleto = INT( 100 * nCuenta / Total de filas )

if ncompleto >= 100 then
    w_test_meter.uo_1.uf_set_position( 100 )
else
    w_test_meter.uo_1.uf_set_position(ncompleto)
end if
```

Tips

---

- En el script del evento RetrieveEnd del dataWindow control poner lo siguiente:

```
close ( w_meter )
```

### 3.- Para llamar a un evento de usuario de una ventana desde una función.

La función es una función de ventana.

dentro de la función:

```
.
.
this.TriggerEvent( "ue_mievento" )
.
.
```

### 4.- Para seleccionar un directorio

```
String PathFull, File, path
Integer value, nPos

value = GetFileOpenName("Seleccionar Directorio", &
    PathFull, File )
If value = 1 Then
    nPos = Pos( PathFull, File ) - 1
    sle_directorio.text = LEFT( docname , nPos )
End If
```

### 5.- Pasar un string como parámetro a un evento con TriggerEvent o PostEvent.

Paso:

```
this.PostEvent( 'ue_xxx', 0, 'algo' )
```

Recuperación:

```
String cCol
cCol = String(Message.LongParm, "address")
```

### 6.- Sheet a la izquierda

- apertura desde el menú

```
OpenSheetWithParm( w_tabs_contrato, parámetro, w_main, 0, Original! )
```

- en Open de la ventana

---

```
this.ToolBarAlignment = AlignatLeft!  
this.ParentWindow().ArrangeSheets( Layer! )
```

La ventana que contenga Tabs debe ser Tipo Main con Menú.



---

## Ventana

### 1.- Capturar el botón derecho del mouse en una MDI frame.

Se desea capturar el botón derecho del mouse en cualquier lugar de una MDI frame.

Para capturarlo en la barra de título y área de menú, mapear un evento de usuario a `pbm_ncrbuttondown` (NC significa no-cliente. Hay otros eventos `pbm_nc...` que podrían ser útiles en otras situaciones)

Para capturarlo en el área de MicroHelp, mapear un evento de usuario a `pbm_rbuttondown`.

Para capturarlo en el área cliente (MDI\_1), mapear un evento de usuario a `pbm_parentnotify` (área cliente es una child window en el frame).

En los script, codificar:

```
IF Message.wordparm=256 THEN
...
END IF
```

El número 256 es el número de mensaje `wm_rbuttondown` (ver `WINDOWS.H`).

#### NOTA

Se puede capturar otros eventos en el área cliente sustituyendo el número correspondiente en `Message.wordparm`.

### 2.- Cerrar ventana con ESC

Evento key de la ventana

```
IF key = KeyEscape! THEN cb_cancelar.TriggerEvent(Clicked!)
```

### 3.- Obtener el handle (puntero) de MDI\_1

Se desea obtener el handle de `mdi_1`, pero la sentencia `Handle(mdi_1)` no trabaja bien y retorna error.

Todos los controles pueden opcionalmente tener un ID asociado. El ID para mdi\_1

es siempre 2000. Se puede obtener el handle usando:

```
function uint GetDlgItem (uint h, uint i) library "user.exe"
```

Pasando el handle de la ventana MDI como primer argumento, y 2000 como el segundo.

---

## DataWindow

1.- Anular tecla ENTER en un datawindow. (útil en freeform).

- a) Crear un evento ('ue\_tecla') con pbm\_dnkey como mensaje
- b) codificar :

```
If KeyDown( KeyEnter! ) Then
    this.SetActionCode( 1 )
End If
```

2.- Cambiar fila si da click en un campo no editable.

En Pb5.0

Clicked

-----

```
this.SetRow( row )
```

En Pb4.0

-----

```
this.SetRow( this.GetClickedrow() )
```

3.- Colorear filas.

En el constructor:

```
this.SetTransObject( Sqlca )
this.Modify( "cod_person.background.mode = '0' "
```

En RowFocusChanged!

// Cambio color fondo y texto

```
this.Modify( "cod_person.background.color = '0~tif( getrow() = " + string(
getrow() ) + " , 16711680, 12632256 ) ' " )
this.Modify("cod_person.Color= '0~tif( getrow() = " + string( getrow() ) + " ,
16777215, 0 ) ' " )
```

// Cambio weight ( simple/bold )

```
this.Modify("pregunta.Font.Weight='400~tif(getrow() = " + string( getrow() ) + "
, 700, 400 )' " )
```

## Tips

---

1) Para colorear una fila dada nRow

```
dw_1.Modify( "cod_derecho.background.color = '0~tif( getrow() = " + string( nRow ) + " , 16711680, 12632256 ) ' " )
```

```
dw_1.Modify("cod_derecho.Color= '0~tif( getrow() = " + string( nRow ) + " , 16777215, 0 ) ' " )
```

2) BORDER Para cambiar volumen de filas

```
this.Modify( "fecha.border = '6~tif( getrow() = " + string( getrow() ) + " , 5, 6 ) ' " )
```

#### 4.- Columna clickeada.

Obtener el nombre de la columna a la cuál se le dió click o dobleclick. Usar en evento Clicked o DoubleClicked de un dw.

```
Int    nColumn
String cColumna
```

```
nColumn = dw_doctos.GetClickedColumn()
cColumna = dw_doctos.Describe( "#" + STRING(nColumn) + ".Name")
```

#### 5.- Cómo detectar un campo calculado.

Para detectar el nombre de un campo calculado texto en un dw, al dar click.

Evento Clicked

-----

```
Int    nCol, nPos
String cColumna
```

```
nCol    = this.GetClickedColumn()
cColumna = this.Describe( "#" + STRING(nCol) + ".Name")
```

```
If cColumna = "!" Then
    cColumna = this.GetObjectAtPointer()
    nPos    = POS( cColumna , "~t" )
    cColumna = LEFT( cColumna , nPos - 1 )
End If
```

```
MessageBox( "Columna" , cColumna )
```

NOTA: El nombre del campo calculado puede ser usado en SetSort().

#### 6.- Copiar / Eliminar todas las filas de un DataWindow.

Para eliminar

```
dw_1.RowsMove(1, dw_1.RowCount(), Primary!, dw_1, 1, delete!)
```

Para copiar desde dw\_1 a dw\_2:

```
dw_1.RowsCopy(1, dw_1.RowCount(), Primary!, dw_2, 1, primary!)
```

7.- Agregar una columna 'virtual' a un DataWindow para importar los datos a otro DataWindow que sí tiene la columna.

Agregar un valor de columna a un set de datos para hacer importstring a otro que tiene el campo.

```
// Agrego campo selección a cada registro y luego importo.
cDatos = dw.describe("DataWindow.Data")
cDatos = f_global_replace( cDatos, "~n~r", ",12345~n~r" )
dw_1.ImportString( cDatos )
```

En este caso, dw no contiene un campo que sí tiene dw\_1 al final.  
Se inicializa con cero.

8.- Devolver valor en DropDownDataWindow.

Supongamos que tenemos el campo 'Campo' (numérico) el cuál tiene un dropDataWindow que muestra una descripción (texto) del campo. Luego de una selección, se valida en alguna parte el valor seleccionado. Si falla, se pone en la variable Message.StringParm = 'Error', y se devuelve la selección a la original.

```
Evento ItemChanged
*-----
String cCampo
cCampo = this.Describe( "Evaluate( 'LookupDisplay(Campo) ', 1 ) " )

If NOT ISNULL( GetText() ) Then
    nCampo = INTEGER( GetText() )
    this.TriggerEvent( "ue_validar" )
    If Message.StringParm = "Error" Then
        this.SetText( cCampo )
        this.SetActionCode(2)
    End If
End If
```

9.- DropDownDataWindow.

Cuando un dw tiene dropdowndatawindow que tiene argumentos de retrieve, si en el evento constructor del dw se ha especificado SetTransObject(...), y luego se

hace un `insertRow`, se pedirán los argumentos de `retrieve`. Sin embargo, si no se ha puesto `SetTransObject(...)` en el constructor, no se pedirán.

Por esto, si el `dw` tiene `dropdowndatawindow` con argumentos de `retrieve`, poner el `SetTransObject(...)` posterior al `insertRow`.

#### 10.- Ordenar por lo que muestra un DropDownDataWindow.

Para ordenar un datawindow por los valores mostrados en un campo que tiene dropdowndatawindow

(ej. cod\_comuna : se ve el nombre de la comuna, pero el campo es numérico)

usar:

Event Retrieve

-----

this.Retrieve()

this.SetSort( "LookupDisplay(cod\_comuna)" )

this.Sort()

#### 11.- Reset a valor original en DropDownDataWindow.

Resetear un dddw a su valor original si no se cumple alguna condición.

Itemchanged event

-----

String cTexto

cText = This.GetItemString(1,1)

If NOT condicion Then

    this.SetItem(1,1,cTexto )

    this.SetActionCode(1)

End If

Itemerror event

-----

this.SetActionCode(1)

#### 12.- Retrieve a dwChild antes del padre.

a)

Al hacer import string a un datawindow que contiene datawindowChild, es necesario recuperar los registros de cada uno de ellos antes de hacer el ImportString.



Ej.:

Evento ue\_retrieve

-----

DataWindowChild dwcTipoDoc, dwcEmisor

this.GetChild( "Cod\_Tipo\_Doc" , dwcTipoDoc )

this.GetChild( "Cod\_Emisor" , dwcEmisor )

dwcTipoDoc.SetTransObject( sqlca )

dwcEmisor.SetTransObject( sqlca )

dwcTipoDoc.Retrieve()

dwcEmisor.Retrieve()

this.ImportString( cDataDocToVer )

b)

Los retrieve para los dddw codigo\_region y codigo\_rae ( sin parámetros ) se ejecutan automáticamente al hacer retrieve del dw que los contiene.

Sin embargo, si los campos tienen dddw con parámetros y se ejecuta el retrieve del que los contiene, intenta hacer retrieve a los dddw con parámetros y aparece la ventana pidiéndolos.

Al insertar un registro en blanco en un datawindowchild con parámetros, evito que pida sus parámetros de retrieve al hacer retrieve del datawindow que lo contiene.

El tratamiento se hace en el rowfocuschanged del dw principal.

### 13.- Obtener lo que muestra un DropDownDataWindow.

cRow = STRING( dw\_doctos.GetRow() )

cTipoDoc = dw\_doctos.Describe( "Evaluate( 'LookupDisplay(Cod\_Tipo\_Doc)' , " + cRow + " ) "

)

---

#### 14.- Datetime.

Cuando utilice un campo DateTime ( TimeStamp al crear la tabla ) en un DataWindow, para que efectivamente se actualice con la fecha y hora actual al hacer un InsertRow(), en Column Specifications - Initial Value, ponga today. En la definición de la tabla debe ser no-nulo,y como valor inicial Set Today en los atributos extendidos.

---

### 15.- Destacar filas con SHIFT + CTRL.

#### INSTANCES

-----

```
boolean ib_action_on_buttonup = false
long   il_LastClickedRow
```

#### CLICKED

-----

```
string  ls_KeyDownType
```

```
If row = 0 then Return
```

```
If Keydown(KeyShift!) then
    wf_Shift_Highlight(row)
ElseIf this.IsSelected(row) Then
    il_LastClickedRow = row
    ib_action_on_buttonup = true
```

```
ElseIf Keydown(KeyControl!) then
    il_LastClickedRow = row
    this.SelectRow(row,TRUE)
```

```
Else
    il_LastClickedRow = row
    this.SelectRow(0,FALSE)
    this.SelectRow(row,TRUE)
```

```
End If //selected row
```

```
ue_lbuttonup ( usuario- pbm_dwnlbuttonup )
```

-----

```
long   ll_ClickedRow
string  ls_KeyDownType
```

```
If ib_action_on_buttonup Then
    ib_action_on_buttonup = false
```

```
If Keydown(KeyControl!) then
    this.selectrow(il_lastclickedrow,FALSE)
```

---

```
Else
    this.SelectRow(0,FALSE)
    this.SelectRow(il_lastclickedrow,TRUE)
End If

il_lastclickedrow = 0
End If
```

---

```

wf_shift_highlight ( public --> integer )
-----
integer li_Idx

dw_highlight.setredraw(false)
dw_highlight.selectrow(0,false)

If il_lastclickedrow = 0 then
//   dw_highlight.SelectRow(al_aclickedrow,TRUE)
   dw_highlight.setredraw(true)
   Return 1
end if

if il_lastclickedrow > al_aclickedrow then
   For li_Idx = il_lastclickedrow to al_aclickedrow STEP -1
      DW_highlight.selectrow(li_Idx,TRUE)
   end for
else
   For li_Idx = il_lastclickedrow to al_aclickedrow
      DW_highlight.selectrow(li_Idx,TRUE)
   next
end if

dw_highlight.setredraw(true)
Return 1

```

16.- Distintos elementos de una columna de un dw.

```

Int    nCol
String cColName , cReq, cEdit, cTipo

nCol      = this.GetColumn()
cColName = this.Describe( "#" + STRING(nCol) + ".Name")
cReq      = this.Describe( cColName + ".Edit.Required")
cEdit     = this.GetText()
cTipo     = this.Describe(cColName + ".ColType")

```

17.- Drag & Drop de filas de un dw a otro.

Desde

---

```

-----
// Elegir un .bmp para el drag

Clicked Event de dw_1
-----
Long nRow
nRow = this.GetClickedRow()
this.SetRow( nRow )
this.selectRow( o ,False )
this.selectRow( nRow, TRUE )
this.drag( begin! )

Hasta
-----
DragDrop Event
-----
DataWindow dw
Long nRow, nNewRow
dw = DraggedObject()
nRow = dw.GetRow()
nNewRow = dw_2.InsertRow(0)
dw_2.SetItem(nNewRow,'col_drag',dw.GetItemString(nRow,'col_drag'))
dw_2.ScrollToRow( nNewrow )
dw_2.SelectRow(0, FALSE )
dw_2.SelectRow( nNewRow, TRUE )

```

## 18.- dwo

```

// Columna
If dwo.type = 'column'
    = dwo.name
    = dwo.id
    = dwo.border
    = dwo.background.color
    = dwo.color
    = dwo.visible
    = dwo.tag
    = dwo.protect
End If

// Texto

```

---

```

If dwo.type = 'text'
    = dwo.name
    = dwo.border
    = dwo.background.color
    = dwo.color
    = dwo.visible
    = dwo.tag
    = dwo.text
End If

```

#### 19.- Ejemplo de edición si dio click.

Se tiene un campo 'seleccion', numérico con edit 'CheckBox' en un dw.  
Además de este campo, existen otros, que serán o no editables en función de si seleccion = 1.

Estos campos tendrán fondo blanco cuando sean editables (y protect = 0), y fondo gris ( y protect = 1 ) cuando no.

Además, se desea que al poner seleccion = 1, quede una de las columnas seteadas para edición.

a) campos que cambiarán de editables a no editables.

En las propiedades de cada campo, en 'expressions' poner:

```

background.color
    if(seleccion=1,rgb(255,255,255),rgb(192,192,192))
protect
    if(seleccion=1,0,1)

```

b) evento clicked del dw.

```

Date dFechaNula
String cRefNula

SetNull( dFechaNula )
SetNull( cRefNula )

If dwo.Type = "column" THEN
    If dwo.name = "seleccion" Then

```

---

```

If this.GetItemNumber( row, "seleccion" ) = 1 Then
    // En este caso se dejan nulos los campos.
    this.SetItem( row, "fecha_inicial", dFechaNula )
    this.SetItem( row, "fecha_termino", dFechaNula )
    this.SetItem( row, "referencia", cRefNula )
Else
    this.PostEvent( 'ue_setcolumn' )
End If
End If
End If

```

c) evento ue\_setcolumn

```
this.SetColumn( 'fecha_inicial' )
```

La razón que este evento sea gatillado con POSTEVENT, es que el valor del campo 'seleccion' ( CheckBox ) no cambia hasta que el evento ha terminado, y por lo tanto si estaba en cero el campo está protegido y no puede ser seteado con SetColumn.

## 20.- Ejemplos uso de la función modify.

Los siguientes ejemplos de uso de la función modify, cambian el *color* y el atributo *visible* de un datawindow usando métodos diferentes. Si se va a cambiar condicionalmente un atributo de una columna de un DataWindow, recordar que esto puede ser hecho usando 'expresiones' al construir el DataWindow.

**<dw>.modify("Datawindow.Color=<long>")**

a.- Este ejemplo cambia el color background a rojo.

String cRet

```
cRet = dw_1.Modify("Datawindow.Color=255")
```

```
If cRet <> "" Then
```

```
    MessageBox( "Modify error", cRet )
```

```
End If
```

b.- Este ejemplo cambia el color background a verde, usando una variable.

String cRet

Long nGreen



## Tips

```
// RGB para el verde = 65280
nGreen = 65280
```

```
cRet = dw_1.Modify("DataWindow.Color=" + String(nGreen) )
If cRet <> "" Then
    MessageBox( "Modify error", cRet )
End If
```

c.- Este ejemplo cambia el color background a azul, usando la función RGB.

```
String cRet
cRet = dw_1.modify("DataWindow.Color="+String(RGB(0,0,255)))
If cRet <> "" Then
    MessageBox( "Modify error", cRet )
End If
```

**<dw>.modify(" <nombrecolumna>.Color=~" <long> ~" ")**

a.- Este ejemplo cambia el color de columna a rojo.

```
// valores de colores:
//RED = 255
// GREEN = 65280
// BLUE = 16711680
```

```
dw_1.modify("mycompute.Color =~"255~"")
```

b.- Este ejemplo cambia el color de columna a verde usando la función RGB.

```
dw_1.modify("dept_name.color=~" + String(RGB(0,255,0)) + "~"")
```

c.- Este ejemplo cambia condicionalmente el color de una columna llamada "mycompute" a rojo o verde, dependiendo de si su valor es menor que 1000.

```
String cRet
```

```
cRet = dw_1.modify("mycompute.color='0~tif(mycompute<1000,255,65280)'" )
If cRet <> "" then
    MessageBox( "", cRet)
End If
```

d.- Este ejemplo cambia condicionalmente el color de una columna llamada "mycompute" a rojo o verde dependiendo de si su valor es igual a 500 o igual a 300.

```
dw_1.modify("mycompute.Color=~"0~tif(dept_id=300,255,if(dept_id=500,16711680,65280))~")
```

e.- Este ejemplo cambia condicionalmente el color de columna a rojo, usando una variable long.

```
String cRet
Long nRed
```

```
nRed = 255
cRet = dw_1.modify("dept_name.Color=~" + String(nRed) + "~")
If cRet <> "" Then
    MsgBox( "", cRet )
End If
```

```
<dw>.modify("<nombrecolumna>.Visible=~"<0-False, 1-True>~")
```

a.- Este ejemplo cambia el atributo visible de la columna dept\_id a 0, para hacer la columna invisible.

```
String cRet
```

```
cRet = dw_1.modify("dept_id.visible='0'")
If cRet <> "" Then
    MsgBox( "Error", cRet )
End If
```

b.- Este ejemplo cambia el atributo visible de la columna dept\_id a 1, para hacerla visible.

```
String cRet
```

```
cRet = dw_1.modify("dept_id.visible='1'")
If cRet <> "" Then
    MsgBox( "Error", cRet )
End If
```

c.- Este ejemplo cambia condicionalmente el atributo visible de la columna dept\_id, dependiendo de su valor.

String cRet

```
cRet = dw_1.modify("dept_id.visible=~"0~tif(dept_id=500,1,0)~"")
If cRet <> "" Then
    MessageBox( "Error", cRet )
End If
```

d.- Este ejemplo cambia condicionalmente el atributo visible de la columna dept\_id anidando condiciones if...

String cRet

```
cRet = dw_1.modify("dept_id.visible=~"0~tif(dept_id<200,0,if(dept_id>400,0,1))~"")
If cRet <> "" Then
    MessageBox( "Error", cRet )
End If
```

e.- Este ejemplo cambia el atributo visible de la columna dept\_id, usando una variable entera.

```
Int i_can_see_it
I_can_see_it = 0
```

```
cRet = dw_1.Modify("dept_id.visible=" + String(I_can_see_it) + "")
If cRet <> "" Then
    MessageBox( "Error", cRet )
End If
```

## 21.- El mayor más uno.

Si al agregar un nuevo registro en una fila de un dw que contiene un campo numérico ascendente, debe encontrar el mayor de ellos en los buffers primary! y delete!, y luego sumar 1 al máximo entre los dos para el nuevo valor agregado...

Para obtener el mayor en el buffer primary!, ponga un campo calculado en la banda summary. Ej. nElMayor: MAX( código for all ).

---

Ejemplo de Script al agregar:

```
-----
Int nNewValor, i, nDelValor

nNewValor = dw_1.GetItemNumber( 1, "nElMayor" )

If ISNULL( nNewValor ) Then // Cuando no hay filas.
    nNewValor = 0
End If

For i = 1 To dw_1.deletedCount()
    nDelValor = dw_1.GetItemNumber( i, "codigo", delete!, false )
    If nDelValor > nNewValor Then
        nNewValor = nDelValor
    End If
Next

nNewValor++

Insertar y poner nNewValor en el campo 'codigo' de la fila nueva.
.
.
```

22.- En qué fila de cuántas está el DataWindow.

Para indicar en qué fila de cuántas está ahora el DataWindow, cree un campo calculado con la siguiente expresión:

```
"Fila " + GetRow() + " de " + RowCount()
```

y póngalo en el Footer del DataWindow object.

23.- Exportar los datos contenidos en un DataWindow a un string.

```
String cDatos
cDatos = dw_1.Describe("DataWindow.Data")
```

Para retornarlos use: dw\_1.ImportString( cDatos )

NOTA: Esta utilización de la función Describe es de gran utilidad y se ejecuta rápidamente.

#### 24.- Filas modificadas.

```

long nRows, row = 0, count = 0

dw_status.AcceptText()
nRows = dw_status.RowCount( )
DO WHILE row <= nRows
    row = dw_status.GetNextModified(row, Primary!)
    IF row > 0 THEN
        count = count + 1
    ELSE
        row = row + 1
    END IF
LOOP

MessageBox(String(count) + " filas fueron modificadas.")

```

Ver también: ModifiedCount() , GetItemStatus(), SetItemStatus()

#### 25.- Find con fechas.

```

String dFormat, cSearch, cClave
Long  nRowFind

dFormat  = "yyyy/mm/dd"           // u otro válido
cSearch = STRING( FechaBuscar , dFormat )

cClave = "STRING(CampoFecha,'" + dFormat + "') = '" + cSearch + "'"

nRowFind = dw.Find( cClave, 1 , dw_RowCount() )
.
.

```

Obs:

- el formato de fecha "dFormat", puede ser otro válido.
- la variable "FechaBuscar" debe ser un valor de fecha válido.
- el signo igual puede cambiarse a >, >= , < , <= .

También puede ponerse:

```
cClave = "STRING(CampoFecha,'yyyymmdd') = " + "" + STRING(
FechaBuscar,'yyyymmdd') + ""
```

## 26.- FindGroupChange.

```
boolean bFound
long nBreakrow

bFound = FALSE
nBreakrow = 0

DO WHILE NOT (bFound)
    nBreakrow = dw_1.FindGroupChange(nBreakrow,1)
    If nBreakrow <= 0 THEN EXIT

    If (expresión) THEN
        bFound = TRUE
        EXIT
    End If

    If nBreakrow = dw_1.RowCount( ) THEN EXIT

    nBreakrow = nBreakrow + 1
LOOP

If bFound THEN
    // Proceso
End If
```

## 27.- Find sin pasar de número de filas del dw.

```
nFind = 0
nFind = dw_1.Find( cFind , nFind, nAuxResp )
DO WHILE nFind > 0
    //Proceso
    If nFind < nAuxResp Then
        nFind = nFind + 1
```

---

```

Else
  EXIT
End If
nFind = dw_1.Find( cFind , nFind, dw_1.RowCount() )
LOOP

```

28.- Hacer que la tecla ENTER funcione como TAB en un dw.

- a) Crear evento a pbm\_dwnprocessenter en el dw.
- b) codificar :

```
Send(Handle(this), 256,9, Long(0,0))
```

29.- Header largo en DataWindow Grid.

Al poner un header en un DataWindow Grid, éste pertenece a una columna, de modo que no puede crecer para traslapar varias columnas.

Para hacer un header libre de movimiento:

- Haga un text object en una columna, tipée el texto que desea.
- Haga click derecho para ir a propiedades del texto
- En 'Position-Layer', seleccione Foreground.

Ahora, el texto puede crecer y moverse para cubrir varias columnas.

Si no desea que el usuario pueda redimensionar las columnas, en propiedades del DataWindow object, desmarque 'Column Moving'.

---

### 30.- Indicador de fila actual 3-D.

El DataWindow object tiene una función llamada CurrentRow. Retorna el número de la fila que tiene el foco. Por otro lado, otra función del DataWindow Object, llamada GetRow, retorna el número de la fila en la cuál ocurre una expresión.

No estamos hablando de funciones que pueden ser llamadas desde script, sino de las funciones que pueden llamarse desde expresiones en el propio DataWindow object.

Tenemos aquí una pequeña aplicación de CurrentRow.

Agregue un text object al DataWindow object. Quita la palabra 'text'. Vaya a sus propiedades y haga su borde 3-D Raised. Dé click al tab Expressions (en Propiedades) y ponga una expresión para el atributo Visible, como esto:

```
if(currentrow()=getrow(),1,0)
```

Esta expresión se evalúa para cada fila en el DataWindow. De tal forma que el text object 3-D Raised sólo es visible para la fila actual. Dimensione el text object para que cubra todas las columnas del DataWindow. Ahora se tiene un indicador 3-D para la fila actual.

Quizás se tenga que poner algún atributo especial para el texto, de modo que no cubra dejando invisibles las columnas. Eso se lo dejamos al lector.

### 31.- Marcar varios registros con el mouse. Evento Clicked.

Evento Clicked

-----

Long nRowSel, nRow

nRow = this.GetClickedRow()

If nRow < 1 Then Return

nRowSel = this.GetSelectedRow( nRow - 1 )

If nRowSel = nRow Then

// Está seleccionado

this.SelectRow( nRow, False )

Else

// No está seleccionado

this.SelectRow( nRow,True )

End If



## Tips

---

```
// Si se desea llevar una cuenta de los registros que están seleccionados,
// crear un campo calculado invisible con lo siguiente:
If (isSelected(), 1, 0 )

// Además, crear un campo suma oculto de este campo y compararlo con rowcount() para realizar
// alguna acción.
totsel = sum(If (isSelected(), 1, 0 ) for all)
```

... siguiendo el script, podría ponerse:

```
If this.GetItemNumber( 1, 'totsel' ) = this.RowCount() Then
    cbx_ope_all.Checked = True
Else
    cbx_ope_all.Checked = False
End If
```

### 32.- MicroHelp en campos de un dw.

En la lista de Tag del datawindow, poner instrucciones por cada campo editable.

En el evento itemfocusChanged poner:

```
String cTag = ""
Int    nCol

nCol = this.GetColumn()

If nCol > 0 Then
    cTag = this.Describe( "#" + STRING( nCol ) + ".tag" )
End If

w_main_mdi.SetMicroHelp( cTag )
```

### 33.- ModifiedCount() , DeleteCount().

Al momento de actualizar, use dw.ModifiedCount() para obtener el número de filas modificadas y no actualizadas.

Ej.

```
If dw_nombre.ModifiedCount() > 0 Then
    dw_nombre.Update()
```

---

End If

Si su dw permite borrar filas, debe usar además, la función DeletedCount().

Ej.

```
If dw_nombre.ModifiedCount() > 0 OR &  
  dw_nombre.DeletedCount() > 0 Then  
  dw_nombre.Update()  
End If
```

NOTA :

- Tanto ModifiedCount() como DeletedCount() trabajan para los registros 'provenientes de un retrieve' ( no nuevos ) y cuando la tabla relacionada con el dw, es actualizable.

- Para usar ModifiedCount() con registros nuevos, en el evento 'losefocus' del dw , agregue lo siguiente:

```
this.AcceptText()
```

### 34.- Mover una columna en un DataWindow Grid.

Para mover o cambiar columnas de lugar en un DataWindow grid, haga un preview, entonces, mueva la columna con drag & drop a la nueva posición. Cuando salga del modo Preview, la columna estará en el lugar que la dejó.

En propiedades del grid (botón derecho) el checkbox 'Column Moving' debe estar chequeado.

### 35.- No cambiar fila en dw freeform.

En el evento ue\_tecla, para evitar movimientos indeseados en los datawindows free\_form, usar este evento con este script

- a) Crear un evento ('ue\_tecla') con pbm\_dnkey como mensaje
- b) codificar :

```
If KeyDown( KeyEnter! ) OR &
  KeyDown( KeyPageUp! ) OR &
  KeyDown( KeyUpArrow! ) OR &
  KeyDown( KeyPageDown! ) OR &
  KeyDown( KeyDownArrow! ) OR &
  KeyDown( KeyControl! ) Then
  this.SetActionCode( 1 )
End If
```

### 36.- No mover DataWindow.

Si se pone una barra de título a un DataWindow, el usuario podrá moverlo. Si se necesita evitar esto:

en un evento de pbm\_syscommand en el dw, codificar lo siguiente:

```
uint a, b
```

```
a = message.wordparm
If a = 61456 or a = 61458 Then
  message.processed = True
  message.returnvalue = 0
End If
```

37.- Ordenar dando click al header de una columna.

```
// POWER 5.0
Clicked
-----
String cHeader, cColumna
If dwo.type = "text" Then
    cHeader = dwo.Name
    cColumna = Left( cHeader, Len(cHeader) - 2 )
    this.Modify( cHeader + ".border='5'" )
    this.SetSort( cColumna + " A" )
    this.Sort()
    this.Modify( cHeader + ".border='6'" )
    // Luego del Sort, queda en el 1er registro.
    this.PostEvent( RowFocusChanged! )
End If
```

```
// POWER 4.0

Clicked
-----
String cObjeto, cColumna, cHeader
Int    nCol, nPos
nCol   = this.GetClickedColumn()
cObjeto = this.Describe( "#" + STRING(nCol) + ".Name" )

If cObjeto = "!" Then
    cObjeto = this.GetObjectAtPointer()
    nPos    = POS( cObjeto , "_t" )
    cHeader = LEFT( cObjeto, nPos + 1 )
    cColumna = LEFT( cObjeto , nPos - 1 )
    this.Modify( cHeader + ".border='5'" )
    this.SetSort( cColumna + " A" )
    this.Sort()
    this.Modify( cHeader + ".border='6'" )
    // Luego del Sort, queda en el 1er registro.
    this.PostEvent( RowFocusChanged! )
End If
```

NOTA

## Tips

---

Para ordenar en sentido inverso, use el evento rbuttondown y en lugar de poner :  
`this.SetSort( cColumna + " A" )`  
 ponga  
`this.SetSort( cColumna + " D" )`

38.- Para cambiar un report anidado.

```
dw_x.Modify("<Reportname>.DataObject='<dataobject name>' ")
```

39.- Para destacar filas. En evento RowFocusChanged.

---- POWER 4.0

```
Long nRow, nRowSel
```

```
nRow = This.GetRow()
```

```
If nRow < 1 Then Return
```

```
nRowSel = this.GetSelectedRow(0)
```

```
If nRowSel > 0 Then
```

```
    this.SelectRow( nRowSel , FALSE )
```

```
End If
```

```
this.SelectRow( nRow , TRUE )
```

---- POWER 5.0

```
Long nRowSel
```

```
If currentrow < 1 Then Return
```

```
nRowSel = this.GetSelectedRow(0)
```

```
If nRowSel > 0 Then
```

```
    this.SelectRow( nRowSel , FALSE )
```

```
End If
```

```
this.SelectRow( currentrow , TRUE )
```

40.- Para manejar pulsaciones de teclas en un dw.

- Cree un evento de usuario usando como Event ID, pbm\_dwnKey
- En el evento creado use la función KeyDown() para el control de las teclas pulsadas por el usuario.

Ej :

```
If KeyDown( KeyEnter! ) Then
    this.TriggerEvent( RowFocusChanged! )
End If
```

Ej : Para usar Ctrl+C para poner el texto de una columna de un dw en el portapapeles use en el evento creado:

```
If KeyDown( 3 ) Then
    this.Copy()
End If
```

...quizás en otro dw se desea pegar este contenido (Ctrl+V).

Use en el evento creado para ese dw:

```
If KeyDown( 22 ) Then
    this.Paste()
End If
```

Ej : Para evitar cambio de fila en un dw (freeform). En el evento creado:

```
If KeyDown( KeyEnter! ) OR &
    KeyDown( KeyPageUp! ) OR &
    KeyDown( KeyUpArrow! ) OR &
    KeyDown( KeyPageDown! ) OR &
    KeyDown( KeyDownArrow! ) OR &
    KeyDown( KeyControl! ) Then

    this.SetActionCode( 1 )

End If
```

41.- Proceso almacenado. Cambiar proceso almacenado de un dw.

```
String cSyntax
cSyntax = dw_1.Describe( "Datawindow.Syntax" )
cSyntax = f_global_replace( cSyntax, 'pa_proceso1', 'pa_proceso2')
dw_1.Create( cSyntax )
```

42.- Referencia a otras filas de un dw.

```
id_producto se refiere a la fila actual
id_producto[-1] se refiere a la fila anterior a la actual
id_producto[1] se refiere a la fila siguiente a la actual
```

En un DataWindow ordenado por id\_producto, podría ir en una expresión para el color:

```
if(id_producto=id_producto[-1], rgb(0,0,0),rgb(255,0,0))
```

43.- RUT en DataWindow.

a. El campo debe ser char(12).

b. Evento editchanged

```
Int nLen
String cDato
```

```
If this.GetColumnName() <> "rut" Then Return
```

```
cDato = this.GetText()
```

```
nLen = Len( cDato )
```

```
If MID(cDato, nLen -1 , 1) = "K" Then
```

```
    cDato = MID( cDato, 1, Len( cDato ) -1 )
```

```
Else
```

```
    cDato = f_formato_rut( cDato )
```

```
End If
```

```
this.SetText( cDato )
```

```
this.SelectText( Len( cDato ) + 1 , 0 )
```

## c. Evento Itemchanged

```
String cCol, cDato
```

```
cCol = this.GetColumnName()
```

```
cDato = this.GetText()
```

```
If cCol = "rut" Then
```

```
    If NOT f_valrut( cDato ) Then
```

```
        MessageBox( "RUT", "Rut no válido" )
```

```
        this.SetActionCode(1)
```

```
    End If
```

```
End If
```

## d. Evento itemerror

```
this.SetActionCode(1)
```

## 44.- ScrollVertical no cambia fila.

Se tiene un datawindow que puede tener varias filas, pero solo se muestra una. Para ver otra fila se tiene la barra vertical, Sin embargo al dar click sobre ella, no se produce un cambio en la fila, sino que solo se ve la nueva fila. Para que realmente se posicione en la fila que está mostrando, escribir el sgte. código en el evento scrollvertical.

```
Evento ScrollVertical
```

```
-----
```

```
String ls_primer
```

```
ls_primer = this.object.datawindow.FirstRowOnPage
```

```
this.SetRow( LONG(ls_primer) )
```

## 45.- Seleccionar filas con click.

```
Evento Clicked del dw
```

```
-----
```

```
nRow = this.GetClickedRow()
```

```
nRowSel = this.GetSelectedRow(0) // fila actual seleccionada.
```

```
If nRowSel > 0 Then
```

```
    this.SelectRow( nRowSel , FALSE )
```



---

```
End If
this.SelectRow( nRow , TRUE )
```

46.- Seleccionar/Deseleccionar registros con un click.

```
Long nRowSel

If row < 1 Then Return

nRowSel = this.GetSelectedRow( row - 1 )

If nRowSel = row Then
    this.SelectRow( row, False )
Else
    this.SelectRow( row, True )
End If
```

47.- Seleccionar/Deseleccionar un campo checkbox en un dw.

Seleccionar/Deseleccionar un campo checkbox en un dw, dando un click en cualquier columna. En el ejemplo, el campo 'seleccion' es de tipo checkbox.

```
Evento Clicked
-----

Int nSeleccion

If row < 1 Then Return

nSeleccion = this.GetItemNumber( row, "seleccion" )
If nSeleccion = 1 Then
    this.SetItem( row, "seleccion", 0 )
Else
    this.SetItem( row, "seleccion", 1 )
End If
```

48.- SetItem cambia a NewModified!.

Se necesita setear columnas a registros nuevos, pero si el usuario deja la fila sin modificar nada, se espera que la fila no sea insertada al hacer Update().

## Tips

---

```
// insertar la fila nueva
nRow = dw_radios.InsertRow(0)

// SeiItem causa el cambio de estado a NewModified!
dw_radios.SetItem( nRow, "cod_empresa" , nCodEmp )
dw_radios.SetItem( nRow, "cod_radio" , nNewCodRadio )

// cambiar el estado de la fila. Debe quedar NEW!
dw_radios.SetItemStatus( nRow, 0, Primary!, NotModified! )
```

49.- Si el dw no avanza/retrocede en filas con teclas de flechas arriba/abajo.

```
Evento tecla ( pbm_dwnKey )
*-----
Long nRow, nRows

nRow = this.GetRow()
nRows = this.RowCount()

If nRow < 1 Then Return

If KeyDown( KeyDownArrow! ) Then
    If nRow < nRows Then
        nRow = nRow + 1
        this.ScrollToRow( nRow )
        this.SetRow( nRow )
    End If
    this.SetActionCode(1)

ElseIf KeyDown( KeyUpArrow! ) Then
    If nRow > 1 Then
        nRow = nRow - 1
        this.ScrollToRow( nRow )
        this.SetRow( nRow )
    End If
    this.SetActionCode(1)
End If
```

50.- Simular AutoSelection en campos con estilo EditMask.

```
En evento Itemfocuschanged
*-----
String cTexto
```

---

```
cTexto = TextLine()
this.SelectText( 1, LEN( cTexto ) )
```

#### 51.- Simular botón dentro de un dw.

Tenemos un texto dentro de un Dw que esperamos se vea y funcione como un botón.

Clicked Event

-----

String cCol

```
If dwo.type = "text" Then
    Choose case dwo.name
    If dwo.name = 'busca_empleador'
        this.SetColumn( 'codigo_empleador' )
        this.Modify("busca_empleador.Border='5'")
        Open( w_buscar_empresa )
        this.Modify("busca_empleador.Border='6'")
    End If
End If
```

#### 52.-Sincronizar movimiento Horizontal y vertical entre dw's.

PB4

---

evento scrollhorizontal

-----

String scrollposition

```
scrollposition = this.describe( "Datawindow.horizontalscrollPosition" )
dw_2.modify( "Datawindow.horizontalscrollposition= " + scrollposition )
```

PB5

---

Supongamos que estamos en el dw\_1

evento scrollhorizontal

-----

```
dw_2.object.datawindow.horizontalscrollposition = scrollpos
```

---

evento scrollvertical

-----  
dw\_2.object.datawindow.verticalscrollposition = scrollpos

53.- Suprimir argumentos de retrieve en DataWindow con DropDownDataWindow.

Cuando se envía ejecuta una función InsertRow o Retrieve en un DataWindow (dw\_1) que tiene una columna DropDownDataWindow (dddw\_1), se ejecuta automáticamente un retrieve en la columna dddw\_1. Si dddw\_1 tiene argumentos de retrieve, aparecerá una caja de diálogo solicitándolos.

Para evitar esto, tome los siguientes pasos para poblar ese DropDownDataWindow antes de poblar el datawindow primario.

Supongamos que la tabla usada tiene 4 columnas: nombre, cod\_pais, cod\_ciudad. El objeto transacción es SQLCA.

// declare una variable de tipo DataWindowChild como puntero al dw child.

DataWindowChild dwc\_child

// obtenga el puntero al child

dw\_1.GetChild( 'cod\_ciudad', dwc\_child )

// haga SetTransObject para el child

dwc\_child.SetTransObject( Sqlca )

// inserte una fila en blanco en el child para prevenir que el padre ejecute un

// retrieve en el child y aparezca la caja de diálogo solicitando argumentos.

dwc\_child.InsertRow(0)

// haga SetTransObject para el padre

dw\_1.SetTransObject(Sqlca)

// haga retrieve en el padre

dw\_1.Retrieve()

Los datos han sido recuperados para el padre, pero todo lo que existe en el dddw\_1 es una fila en blanco. Cómo el dddw se puebla, depende de la información que requiera. Un ejemplo: el usuario da click a un país particular y la información que requiere el child son los códigos de ciudad para ese país solamente.

## Tips

---

Se podría escribir algo como:

```
Long   nRow
String nCodPais

nRow    = dw_1.GetClickedRow()
nCodPais = dw_1.GetItemString( nRow, 'cod_pais')
dwc_child.Retrieve(nCodPais)
```

El script podría ser puesto en el evento Clicked, RowFocusChanged, GetFocus u otro del padre, dependiendo de nuestros requerimientos.

Nota: Otra opción, en lugar de insertar una fila en el child a este nivel, podría ser crear antes esta fila en el child. Para hacer esto, en el DataWindow painter y con el DataWindow que será DropDownDataWindow, ir a 'Rows' – 'Data' y añadir una fila en blanco.

### Después de un Modify.

Otra aplicación puede ser, si se desea en algún momento cambiar el DataWindow object de un DropDownDataWindow y ejecutar retrieve en éste con un valor por defecto.

Antes, en el DataWindow que será DropDownDataWindow inserte una fila en blanco según la nota anterior.

```
DataWindowChild dwc_child
String error
```

```
Error = dw_1.modify("cod_depto.dddw.name=d_depto2")
```

```
dw_1.GetChild( "cod_depto", dwc_child )
dwc_child.SetTransObject(Sqlca)
```

```
// retrieve con un valor por defecto
dwc_child.Retrieve(100)
```

Nota: Cuando se sse ejecuta una función modify, el acto de modificar realmente destruye y recrea el objeto. Cuando el objeto se recrea, se ejecuta automáticamente un retrieve.

---

54.- Ver error.

```
dberror
```

```
-----
```

```
MessageBox ("Database Error", "Database Error Code: " + String (sqldbcode) +  
&
```

```
                "~r~nDatabase Error Message:  " + sqlerrtext,  
exclamation!)
```

```
sqldbcode = -193. Es clave duplicada.
```

```
ver cuando el campo no admite nulo y se le manda uno.
```

## **Funciones de usuario**

**1.- f\_alltrim.** Elimina vacíos a izquierda y derecha de un string.

```
string f_alltrim(string ctexto)
```

```
Return( LeftTrim( RightTrim( cTexto ) ) )
```

**2.- f\_conectar.** Conecta a una base de datos.

```
integer f_conectar(transaction otran)
```

```
f_desconectar( otran )
```

```
connect using otran;
```

```
return otran.sqlcode
```

**3.- f\_copyfile.** Copia un archivo en otro.

```
None f_copyfile(string fromfile, string tofile)
```

```
//
```

```
// Sintaxis: f_copyFile( FromFile, ToFile )
```

```
//
```

```
// Copia el archivo <FromFile> a <ToFile>
```

```
//
```

```
// Si <ToFile> existe, lo sobrescribe.
```

```
//
```

```
Integer fHandleInput , fHandleOutput, Loops = 1, i
```

```
Long   fLenInput
```

```
Blob   cBloque
```

```
pointer oldpointer
```

```
oldpointer = SetPointer( HourGlass! )
```

```
// Tamaño de archivo Input
```

```
fLenInput = FileLength( FromFile )
```

```
// Obtengo puntero a Input
```

```
fHandleInput = FileOpen( FromFile, StreamMode!, Read!, LockRead! )
```

```
// Obtengo puntero a Output
```

```
fHandleOutPut = FileOpen( ToFile, StreamMode!, Write!, Shared!, Replace! )
```



```
// Determino cuantas lecturas de input se deben ejecutar
```

```
If fLenInput > 32765 Then
```

```
    If MOD( fLenInput, 32765 ) = 0 Then
```

```
        Loops = fLenInput / 32765
```

```
    Else
```

```
        Loops = ( fLenInput / 32765 ) + 1
```

```
    End If
```

```
End If
```

```
// Leo archivo Input y escribo a OutPut
```

```
For i = 1 To Loops
```

```
    FileRead( fHandleInput , cBloque )
```

```
    FileWrite( fHandleOutPut, cBloque )
```

```
Next
```

```
// Cierro los archivos
```

```
FileClose( fHandleInput )
```

```
FileClose( fHandleOutPut )
```

```
SetPointer( oldPointer )
```

**4.- f\_creadwsp.** Crea un DataWindow con llamada a proceso almacenado.

integer f\_creadwsp(datawindow dw, string columnas, integer ancho, string headers, string proceso, string argumentos)

// Construye la sintaxis de un datawindow grid con llamada a proceso almacenado con o sin argumentos. Luego crea el datawindow dw enviado a partir de la sintaxis. Así el dw, queda listo para efectuarle un retrieve, lo cual ejecutará el proceso que debe estar creado previamente en el servidor.

```
// dw      : Datawindow control que se armará con la sintaxis.
```

```
// columnas : String con las columnas/tipo a crear.
```

```
// ancho    : Ancho de las columnas. ( 600 es buen número ). Todas mismo ancho.
```

```
// headers  : String con lo que se desea aparezca como título de cada columna.
```

```
// proceso  : String con el nombre del proceso almacenado.
```

```
// argumentos : Argumentos del proceso almacenado.
```

```
//
```

```
// Ej:
```

```
// nRet = f_creadwsp( dw_1, &
```

```
// "cd_escenario,decimal(2),cd_unid_constru,char(8),servid,long,costo_total,real", &
```

```
// 600, &
```

```
// "ESCENA,COD_UC,SERVIDUMBRE,TOTAL", &
```

```
//      "dbo.spa_lee_detalle_uc", &
//      "arg1,number,arg2,string,argumento3,number" )
//
//If nRet = 1 Then
//  dw_1.Retrieve( arg1, arg2, arg3 )
// End If

// OBS:
//
// 1. Los tipos permitidos para las columnas son:
//   char(n), number, decimal(n), date, datetime, time, long, ulong, real
//
// 2. Los nombres para columnas y header deben cumplir con las normas
//   para nombres de columnas títulos en un dw,
//   por ejemplo, no llevar espacios en blanco (usar underscore), etc...
//
// 3. Los tipos permitidos para argumentos son:
//   number, string, date, time, datetime, numberarray, stringarray,
//   datearray, timearray, datetimearray
//
// 4. Si no hay argumentos para el proceso almacenado, enviar "".
//
// 5. Si headers = "". Se tomarán los nombres de las columnas.
//
// 6. Si headers <> "", el número de headers debiera ser igual al de columnas.
//   Si alguno es "", pone el nombre de la columna.
//
//
// La necesidad que creó esta función.
//
// Se requiere crear gran cantidad de consultas ( cientos ) a una base de datos,
// para exportar los resultados a archivos ( excel, dbase,...).
// En este caso, la apariencia del dw no es relevante ( puede estar invisible incluso ),
// pero sí lo es el tamaño de la librería que contendría los datawindows que habría
// que hacer para cada consulta.
// Con la función sólo se tienen que hacer los procesos almacenados, y los dw son
// creados virtualmente.
//
```

```
Int  X, i, delta
String cSql, cColumnas, cHeaders, cProceso, cArgumentos, cDetalle
String cNomCol , cTipo, cTit
```

```
String cError
```

```

Int  nError

X = 10
i = 0
delta = 0

If ancho = 0 Then
    ancho = 600
End If

// Columnas

cNomCol = TRIM(f_get_token( columnas, "," ))
cTipo  = TRIM(f_get_token( columnas, "," ))
cTit   = TRIM(f_get_token( headers, "," ))

Do while cNomCol <> ""
    X = 10 + ancho * i + delta
    If cTit = "" Then
        cTit = cNomCol
    End If
    cColumnas = cColumnas + 'column=(type=' + cTipo + ' updatewhereclause=yes name=' + cNomCol +
'dbname=' + cNomCol + ' ) '
    cHeaders = cHeaders + 'text(band=header alignment="2" text=' + cTit + '"border="6" color="0"
x=' + String(X) + ' y="12" height="57" width=' + String(ancho) + ' name=' + cTit + '_t
font.face="Arial" font.height="-8" font.weight="400" font.family="2" font.pitch="2"
font.charset="0" background.mode="2" background.color="12632256" ) '
    cDetalle = cDetalle + 'column(band=detail id=' + String( i + 1 ) + ' alignment="0"
tabsequence=32766 border="0" color="0" x=' + String(X) + ' y="4" height="65"
width=' + String(ancho) + ' format="[general]" name=' + cNomCol + ' edit.limit=60
edit.case=any edit.focusrectangle=no edit.autoselect=yes edit.autohscroll=yes
font.face="Arial" font.height="-8" font.weight="400" font.family="2" font.pitch="2"
font.charset="0" background.mode="1" background.color="536870912" ) '
    cNomCol = TRIM(f_get_token( columnas, "," ))
    cTipo  = TRIM(f_get_token( columnas, "," ))
    cTit   = TRIM(f_get_token( headers, "," ))
    i++
    delta = 8
Loop

cProceso = 'procedure="1 execute ' + proceso + ';1 '

// Argumentos

cNomCol = TRIM(f_get_token( argumentos, "," ))

```

```
cTipo = TRIM(f_get_token( argumentos, "," ))
```

```
Do while cNomCol <> ""
```

```
    cArgumentos = cArgumentos + '(' + cNomCol + ',' + cTipo + ',' +  
        cProceso = cProceso + '@' + cNomCol + '=' + cNomCol + ',' +  
        cNomCol = TRIM(f_get_token( argumentos, "," ))  
        cTipo = TRIM(f_get_token( argumentos, "," ))
```

```
Loop
```

```
If Trim(cArgumentos) <> "" Then
```

```
    cArgumentos = MID( cArgumentos , 1 , Len(cArgumentos) - 1 )  
    cArgumentos = "arguments=(" + cArgumentos + ")"  
    cProceso = MID( cProceso, 1, Len( cProceso ) - 2 )
```

```
End If
```

```
cProceso = cProceso + ""
```

```
// Sintaxis
```

```
cSql = &
```

```
'release 5; ' + &
```

```
'datawindow(units=0 timer_interval=0 color=16777215 processing=1
```

```
print.documentname="" print.orientation = 0 print.margin.left = 110 print.margin.right =
```

```
110 print.margin.top = 97 print.margin.bottom = 97 print.paper.source = 0
```

```
print.paper.size = 0 print.prompt=no grid.lines=0 grid.columnmove=no
```

```
selected.mouse=no ) ' + &
```

```
'header(height=85 color="536870912" ) ' + &
```

```
'summary(height=1 color="536870912" ) ' + &
```

```
'footer(height=0 color="536870912" ) ' + &
```

```
'detail(height=81 color="536870912" ) ' + &
```

```
'table(' + &
```

```
    RightTrim(cColumnas) + '' + &
```

```
    RightTrim(cProceso) + '' + &
```

```
    RightTrim(cArgumentos) + '' + &
```

```
) ' + &
```

```
    RightTrim(cHeaders) + '' + &
```

```
    RightTrim(cDetalle)
```

```
Clipboard( cSql )
```

```
nError = dw.create( cSql, cError )
```

```
If nError <> 1 Then
```

```
    beep(1)
```

```
    MessageBox( "Error al crear datawindow", cError )
```

```
End If
```

Return nError

**5.- f\_creatatabla.** Crea tabla en una base de datos.

integer f\_creatatabla(transaction tran, string nombre\_tabla, string campos)

```
//
// Esta función verifica si la tabla a crear existe.
// Si existe, la borra.
//
// Se supone que la función sólo intentará crear tablas auxiliares.
```

String cSql

Int nRet

```
cSql = "if exists (select 1 " + &
        "from sysobjects " + &
        "where name = '" + nombre_tabla + "' " + &
        "and type = 'U') " + &
        "drop table " + nombre_tabla
```

// ELIMINACION SI EXISTE

f\_conectar( tran )

nRet = f\_ejecutar( tran, cSql )

If nRet < 0 Then

beep(1)

```
    MessageBox( "Atención", "No se pudo eliminar la tabla '" + nombre_tabla +
        "'.~n~r~n~r" + &
        "Mensaje del Servidor: " + "~n~r~n~r" + &
        Tran.SqlErrText )
```

End If

If nRet < 0 Then

f\_desconectar( tran )

Return nRet

End If

```
cSql = "CREATE TABLE " + nombre_tabla + &
        " (" + campos + " )"
```

nRet = f\_ejecutar( tran, cSql )

```

If nRet < 0 Then
    beep(1)
    MsgBox( "Atención", "No se pudo crear la tabla " + nombre_tabla +
".~n~r~n~r" + &
        "Mensaje del Servidor: " + "~n~r~n~r" + &
        Tran.SqlErrText )
End If
f_desconectar( tran )

Return nRet

```

**6.- f\_dberror.** Verifica si la instrucción SQL tuvo éxito.

integer f\_dberror(transaction tran, string instruccion, string tabla)

String mensaje, accion, ENTER

ENTER = CHAR(13) + CHAR(10)

instruccion = lower( instruccion )

tabla = upper( tabla )

mensaje = ""

If tran.sqlcode = -1 Then

choose case instruccion

case "select"

accion = "LEER"

case "update"

accion = "ACTUALIZAR"

case "insert"

accion = "INSERTAR"

case "delete"

accion = "ELIMINAR"

case else

accion = instruccion

end Choose

If Trim(tabla) = "" Then

mensaje = "Error al " + accion + ENTER + ENTER

Else

mensaje = "Error al " + accion + " en tabla " + tabla + ENTER + ENTER

End If

```

        beep(1)

    If LEFT(tran.sqlerrtext,3) <> 'um_' Then
        mensaje = mensaje + "Sqlcode: " + String( tran.sqlcode ) + ", "
        mensaje = mensaje + "Sqlldbcde: " + String( tran.sqlldbcde ) + ENTER +
ENTER
        mensaje = mensaje + "Mensaje del servidor: " + ENTER + ENTER +
tran.sqlerrtext
        OpenWithParm( w_error_sp, mensaje )
    Else
        mensaje = MID(tran.sqlerrtext,4)
        MsgBox( "Atención", mensaje, Exclamation! )
    End If

End If

Return ( tran.sqlcode )

```

Ejemplo de uso.

-----  
f\_conectar( Sqlca )

```

declare insertar_inout procedure for @nRet = sp_i_edit_gate_control
    @cd_depot      = :gnCodDepot,
    @cor_cnt       = :nCorCnt,
    @in_out        = :cInOut,
    @sigla         = :cSigla,
    @numero        = :nNumero,
    @digito        = :cDig,
    @cd_tp_cnt     = :cTipo,
    @st_hc         = :cSTHC,
    @condicion     = :cCond,
    @tatc          = :nTatc,
    @tamano        = :nSize,
    @cd_puerto_ori = :cPtoOri,
    @cd_puerto_dst = :cPtoDes,
    @rut_cliente   = :cRutClie,
    @cd_linea      = :cLinea,
    @cd_transportista = :nTrans,
    @rut_chofer    = :cRutChof,
    @nm_chofer     = :cNomChof,
    @patente_camion = :cPatente
Using Sqlca;

```

```
bError = False
```

```
execute insertar_inout;
```

```
nRetProc = f_dberror( Sqlca, 'insert', 'Movimiento In/Out' )
```

```
If nRetProc = -1 Then
```

```
    bError = True
```

```
    Close insertar_inout;
```

```
    f_desconectar( Sqlca )
```

```
    Return
```

```
End If
```

```
Close insertar_inout;
```

```
f_desconectar( Sqlca )
```

**7.- f\_delchar.** Elimina caracteres de un string.

```
string f_delchar(string cEliminar, string cEnEsteString)
```

```
//
```

```
//     Elimina caracteres de un string
```

```
//
```

```
//     Uso: cStr = f_delchar( cEliminar, cEnEsteString )
```

```
//
```

```
//     <cEliminar>      : String conteniendo los caracteres a eliminar...
```

```
//     <cEnEsteString> : ...en este string.
```

```
//
```

```
//     Retorna string con los caracteres eliminados.
```

```
//
```

```
//     Ej : cTest = "Ayer fué 13 de Abril de 1995"
```

```
//           cTest = f_delchar( "0123456789", Test )
```

```
//
```

```
//           cTest se transforma en : "Ayer fué de Abril de"
```

```
//
```

```
Integer i
```

```
String cNewStr = ""
```

```
For i = 1 To Len( cEnEsteString )
```

```
    If Pos( cEliminar, MID( cEnEsteString, i, 1 )) = 0 Then
```

```
        cNewStr = cNewStr + MID( cEnEsteString, i, 1 )
```

```
    End If
```

```
Next
```



```
Return( cNewStr )
```

8.- **f\_delta\_minutos.** Retorna los minutos entre dos datetime.

```
long f_delta_minutos( datetime fec_ini, datetime fec_fin )
```

```
int nDias, nHrs, nMin
```

```
long nTiempo
```

```
nDias = DaysAfter ( date(fec_ini), date(fec_fin) )
```

```
nTiempo = nDias * 1440
```

```
nHrs = hour(time(fec_fin)) - hour(time(fec_ini))
```

```
nTiempo = nTiempo + (nHrs * 60)
```

```
nMin = Minute(time(fec_fin)) - Minute(time(fec_ini))
```

```
nTiempo = nTiempo + nMin
```

```
return nTiempo
```

8.- **f\_desconectar.** Desconecta de una base de datos.

```
integer f_desconectar(transaction tran)
```

```
disconnect using otran;
```

```
return otran.sqlcode
```

9.- **f\_diachar.** Retorna el día en palabras.

```
string f_diachar(integer nNumDia )
```

```
String cNombre
```

```
CHOOSE CASE nNumDia
```

```
    CASE 1
```

```
        cNombre = "Domingo"
```

```
    CASE 2
```

```
        cNombre = "Lunes"
```

```
    CASE 3
```

```
        cNombre = "Martes"
```

```
    CASE 4
```

```

        cNombre = "Miércoles"
CASE 5
        cNombre = "Jueves"
CASE 6
        cNombre = "Viernes"
CASE 7
        cNombre = "Sábado"
END CHOOSE
Return( cNombre )

```

NOTA. El día de una fecha: DayNumber( fecha )

**10.- f\_digvercnt.** Retorna dígito verificador de un container.

```
string f_digvercnt(string cvar1)
```

```
String cVar2, cVar3, cRet
Long  nVar1, nVar2, i, j, z
```

```
cRet = '?'
```

```

If Len( cVar1 ) <> 10 Or IsNumber( Left(cVar1,4)) Or &
  LONG(MID( cVar1, 5, 6 )) < 0 Or LONG(MID( cVar1, 5, 6 )) > 999999 Then
  Return cRet
End If

```

```

cVar2 = 'ABCDEFGHJKLMNOPQRSTUVWXYZ'
cVar3 = '1012131415161718192021232425262728293031323435363738'

```

```
nVar1 = 0
```

```

For i = 1 To 4
  For j = 1 to 26
    If Upper(MID(cVar1,i,1)) = MID(cVar2,j,1) Then
      nVar1 = nVar1 + ( LONG(MID(cVar3,(j*2)-1,2)) * ( 2^ ( i - 1)))
      EXIT
    End If
  Next
Next

```

```

For i = 5 to 10
  nVar1 = nVar1 + ( LONG(MID(cVar1,i,1)) * ( 2^ ( i - 1)))

```

Next

nVar2 = MOD(nVar1,11)

If nVar2 = 10 Then

nVar2 = 0

End If

cRet = Trim( String(nVar2) )

Return cRet

**11.- f\_ejecutar.** Ejecuta un comando SQL.

integer f\_ejecutar(transaction tran, string que\_accion)

If tran.DBHandle() <> 0 Then

EXECUTE IMMEDIATE :que\_accion USING tran;

End If

Return tran.SQLCode

**12.- f\_elimina tabla.** Elimina una tabla de la base de datos.

integer f\_elimina tabla(transaction tran, string nombre\_tabla)

//

// Esta función verifica si la tabla a crear existe.

// Si existe, la borra.

//

String cSql

Int nRet

cSql = "if exists (select 1 " + &

"from sysobjects " + &

"where name = " + nombre\_tabla + " " + &

"and type = 'U') " + &

" drop table " + nombre\_tabla

// ELIMINACION SI EXISTE

f\_conectar( tran )

nRet = f\_ejecutar( tran, cSql )

If nRet < 0 Then

256

```
        beep(1)
        MsgBox( "Atención", "No se pudo eliminar la tabla '" + nombre_tabla +
".~n~r~n~r" + &
        "Mensaje del Servidor: " + "~n~r~n~r" + &
        Tran.SqlErrText )
End If

f_desconectar( tran )

Return nRet
```

**13.- f\_es\_par.** Determina si un número es par.

```
Boolean f_es_par( long numero )

Boolean bRet

bRet = False

If LONG(numero/2)*2 = numero Then
    bRet = True
End If

Return bRet
```

**14.- f\_fecha\_server.** Trae la fecha del servidor.

```
string f_fecha_server()

String cFecha

f_conectar( Sqlca)

declare x procedure for @cFecha = sp_getdateserver
    using Sqlca;
execute x;
fetch x into :cFecha;
close x;

f_desconectar(Sqlca)

Return cFecha
```

El procedimiento sp\_getdateserver:

```
create procedure sp_getdateserver
as
select getdate()
Return 0
```

**15.- f\_file2string.** Lee un archivo a un string.

```
integer f_file2string(string cFile, REF string cLinea)

//
// Lee un archivo hasta 60000 bytes, en una variable string.
//
//      f_file2string( cFile , cString ) --> nRet
//
//      Retorna el número de bytes leídos, o -1 si hubo error.
//

Int    nHIn, nLeidos = 0

nHIn   = FileOpen( cFile, StreamMode!, Read! )
If nHIn <> -1 Then
    nLeidos = FileRead ( nHIn, cLinea )
Else
    nLeidos = -1
End If

FileClose( nHIn )

Return ( nLeidos )
```

**16.- f\_filetear\_archivo.** Crea archivo desde otro con separadores de línea.

Lee un archivo un archivo de texto y crea un archivo de salida cuyo contenido son líneas obtenidas del archivo fuente las cuales vienen separadas por el caracter especificado. Dicho de otro modo, cambia las ocurrencias del caracter especificado, por el caracter especificado + ENTER, escribiendo el resultado en un archivo de salida.

Ej. Suponga que tiene un archivo continuo donde las entidades van separadas por el signo ~.

ALFA ROMEO~FIAT~MERCEDES BENZ~PEUGEOT~BMW

y se desea obtener un archivo como el siguiente:

```
ALFA ROMEO
FIAT
MERCEDES BENZ
PEUGEOT
BMW
```

```
integer f_filetear_archivo( string cInFile, string cOutFile, cSep, bIncluirsepInOut,
bEliminarEnter )
```

```
// cInFile. Archivo origen.
// cOutFile. Archivo destino.
// cSep. Caracter separador.
// bIncluirsepInOut. Incluir separador en el archivo de salida.
// bEliminarEnter. Si elimina ENTER (CHAR(13)+CHAR(10)) antes de escribir.
```

```
// f_filetaear_archivo lee el archivo cInFile y escribe el archivo cOutFile
// cuyas filas son los textos separados por cSep en cInFile.
```

```
// Ej. f_filetear_archivo( "c:\kk.mbp", "c:\pp.sal", "~", TRUE, TRUE)
```

```
Int nRet
String slTemp, cReemp
Long llFileHnd, llBytesRead, llLargo, llPosl
Long llTotalBytes
Blob tlBuffer, tlTextoBruto
```

```
SetPointer(HourGlass!)
```

```
//Lectura
llLargo = FileLength(cInFile)
llFileHnd = FileOpen(cInFile, StreamMode!, Read!, LockRead!)
llBytesRead = FileRead(llFileHnd, tlBuffer)
llTotalBytes = llBytesRead
```

```
Do While llBytesRead > 0
    tlTextoBruto = tlTextoBruto + tlBuffer
    llBytesRead = FileRead(llFileHnd,tlBuffer)
    llTotalBytes = llTotalBytes + llBytesRead
Loop
FileClose(llFileHnd)
```

```
// Escritura
```

```

If bIncluirSepInOut Then
    cReemp = cSep + CHAR(13) + CHAR(10)
Else
    cReemp = CHAR(13) + CHAR(10)
End If

llFileHnd = FileOpen(cOutFile, StreamMode!, Write!, LockWrite!, Replace!)
slTemp = String( tlTextoBruto)
slTemp = LEFT( slTemp, 5000)
llLargo = Len(slTemp)

Do While llLargo > 0
    If bEliminarEnter Then
        slTemp = f_global_replace(slTemp, CHAR(10), "")
        slTemp = f_global_replace( slTemp, CHAR(13), "" )
    End If

    slTemp = f_global_replace( slTemp, cSep, cReemp )
    FileWrite( llFileHnd, slTemp )

    tlTextoBruto = BlobMid( tlTextoBruto, llLargo + 1)
    slTemp = String( tlTextoBruto )
    slTemp = LEFT( slTemp, 5000 )
    llLargo = Len( slTemp )
Loop

FileClose( llFileHnd )

SetPointer( Arrow! )

Return nRet

```

**17.- f\_formato\_rut.** Formatea un RUT con puntos y guión.

```
string f_formato_rut(string cRut)
```

```
String cRet = "", cSep
Int  nLen, i
```

```
cRut = f_delchar( ".-", cRut )
nLen = Len( cRut )
```

```
For i = nLen To 1 STEP -1
```

```
    CHOOSE CASE i
```

```

CASE nLen - 1
    cSep = "-"
CASE nLen - 4, nLen - 7
    cSep = "."
CASE ELSE
    cSep = ""
END CHOOSE

cRet = MID( cRut, i , 1 ) + cSep + cRet

```

Next

Return cRet

### 18.- f\_get\_token. Extrae tokens delimitados.

string f\_get\_token(REF string source, string separator)

```

// The function f_Get-Token receive, as arguments, the string from which
// the token is to be stripped off, from the left, and the separator
// character. If the separator character does not appear in the string,
// it returns the entire string. Otherwise, it returns the token, not
// including the separator character. In either case, the source string
// is truncated on the left, by the length of the token and separator
// character, if any.

```

```

int          p
string ret

```

p = Pos(source, separator) // Get the position of the separator

```

if p = 0 then                                     // if no separator,
    ret = source                                   // return the whole source string and
    source = ""                                    // make the original source of zero
length
else
    ret = Mid(source, 1, p - 1) // otherwise, return just the token and
    source = Right(source, Len(source) - p) // strip it & the separator
end if

```

return ret

### 19.- f\_gettextfrompath. Obtiene la extensión de un archivo, desde un path.



```

string f_gettextfrompath(string cPath)

Boolean bOk = FALSE
String cExt , cChar
Int nLenPath, i

cExt = ""
nLenPath = LEN( cPath )

For i = nLenPath To 1 STEP -1
    cChar = MID( cPath , i , 1 )
    If cChar = "\" OR cChar = ":" Then
        EXIT
    ElseIf cChar = "." Then
        bOk = TRUE
        EXIT
    Else
        cExt = cChar + cExt
    End If
Next

If NOT bOk Then
    cExt = ""
End If

Return ( cExt )

```

**20.- f\_getfilefrompath.** Obtiene nombre de archivo desde un path.

```

string f_getfilefrompath(string cPath)

String cFile , cChar
Int nLenPath, i

cFile = ""
nLenPath = LEN( cPath )

For i = nLenPath To 1 STEP -1
    cChar = MID( cPath , i , 1 )
    If cChar = ":" OR cChar = "\" Then
        EXIT
    Else
        cFile = cChar + cFile
    End If

```

Next

Return ( cFile )

**21.- f\_global\_replace.** Hace un reemplazo en todo el string.

```
string f_global_replace(string source, string look_for, string replace_with)
```

```
/*
```

```
A String Occurrence Search and Replace Routine
```

The following code demonstrates a string occurrence search and replace routine.

This routine works generically for any string. For example, if old\_str = "red" and new\_str = "green", all occurrences of "red" inside of mystring will be replaced with "green".

```
*/
```

```
int start_pos=1,len_look_for
```

```
len_look_for = len(look_for)
```

```
//find the first occurrence of look_for ...
```

```
start_pos = Pos(source,look_for,start_pos)
```

```
//only enter the loop if you find whats in look_for
```

```
DO WHILE start_pos > 0
```

```
    //replace look_for with replace_with ...
```

```
    source = Replace(source,start_pos,Len_look_for,replace_with)
```

```
    //find the next occurrence of look_for
```

```
    start_pos = Pos(source,look_for,start_pos+Len(replace_with))
```

```
LOOP
```

```
return source
```

**22.- f\_llena\_ddlb.** Llena un dropdownlistbox con una columna de tabla.

```
integer f_llena_ddlb( string nombre_columna, string nombre_tabla, string
condicion_where, dropdownlistbox nombre_ddlb)
```

```
string    descrip_corta, sentencia_sql
```

```
IF condicion_where = "" THEN
```

```
    sentencia_sql = "SELECT " + nombre_columna + " FROM " + nombre_tabla &
    + " ORDER BY " + nombre_columna
```

```

ELSE
    sentencia_sql = "SELECT " + nombre_columna + " FROM " + nombre_tabla &
        + " WHERE " + condicion_where + " ORDER BY " + nombre_columna
END IF

```

```

PREPARE sqlsa FROM :sentencia_sql;

```

```

DECLARE un_cursor DYNAMIC CURSOR FOR sqlsa;

```

```

OPEN DYNAMIC un_cursor;
IF Sqlca.Sqlcode < 0 THEN
    MessageBox("Error en Base de datos!", sqlca.sqlerrtext)
    CLOSE un_cursor;
    Return sqlca.sqlcode
END IF
nombre_ddlb. Reset ( )

```

```

DO WHILE SQLCA.SQLcode = 0
    FETCH un_cursor INTO :descrip_corta;
    IF SQLCA.SQLcode = 0 THEN
        nombre_ddlb. AddItem ( descrip_corta )
    ELSEIF Sqlca. Sqlcode < 0 THEN
        MessageBox("Error en base de datos", sqlca.sqlerrtext)
        CLOSE un_cursor;
        Return sqlca.sqlcode
    ELSE
        Exit
    END IF
LOOP

```

```

CLOSE un_cursor;
Return ( 0 )

```

**23.- f\_mes\_char.** Retorna mes en palabras.

```

string f_mes_char(integer nMes)

```

```

String cRet

```

```

cRet = ""

```

```

Choose Case nMes

```

```

Case 1

```

```

    cRet = "Enero"

```

```

Case 2

```

```

        cRet = "Febrero"
Case 3
        cRet = "Marzo"
Case 4
        cRet = "Abril"
Case 5
        cRet = "Mayo"
Case 6
        cRet = "Junio"
Case 7
        cRet = "Julio"
Case 8
        cRet = "Agosto"
Case 9
        cRet = "Septiembre"
Case 10
        cRet = "Octubre"
Case 11
        cRet = "Noviembre"
Case 12
        cRet = "Diciembre"
End Choose

Return cRet

```

#### **24.- f\_padr.** Rellena a la derecha de un string.

```

string f_padr(string cTexto, integer nLen, string cCharFill)

//Rellena a la derecha <cTexto> con <nLen> caracteres <cCharFill>.

Return( cTexto + FILL( cCharFill, nLen - Len( cTexto ) ) )

```

#### **25.-f\_string2file.** Graba un string en un archivo.

```

integer f_string2file(string cString, string cFile)

Int nH, nBytes

nH = FileOpen( cFile, StreamMode!, Write!, LockReadWrite!, Replace! )
nBytes = FileWrite( nH, cString )
FileClose( nH )

Return 0

```

**26.-f\_valrut.** Valida un RUT.

Boolean f\_valrut(string cRut)

```
//
//      Valida un RUT.
//
//      Uso:   bOkRut = f_valrut( cRut )
//
// Si cRut es nulo o vacío, retorna FALSE.
// El Rut puede venir en cualquiera de los formatos del Ej.
//      Se Asume que último caracter es el dígito verificador.
//
// Ejs:   bOkRut = f_valRut( 8.991.087-0 )
//        bOkRut = f_valRut( 8991087-0 )
//        bOkRut = f_valRut( 89910870 )
//
// Waldo Gómez A.
```

String cDig, xNum, cResto

Integer nSuma = 0, nFactor = 2 , nResto, i

Boolean lOk = True, lCont = True

cRut = f\_delchar( "-.", RightTrim(LeftTrim( cRut ) ) )

cDig = Right(cRut,1)

cRut = MID( cRut, 1, Len(cRut) - 1 )

For i = Len(cRut) To 1 STEP -1

    xNum = MID(cRut, i, 1 )

    If Pos( "0123456789", xNum ) = 0 Then

        lOk = FALSE

        Exit

    End If

    nSuma = nSuma + INTEGER(xNum) \* nFactor

    If nFactor = 7 Then

        nFactor = 2

    Else

        nFactor = nFactor + 1

    End If

Next

If lOk Then

```
    nResto = MOD(nSuma,11)
    If nResto = 0 Then
        cResto = "0"
    ElseIf nResto = 1 Then
        cResto = "K"
    Else
        cResto = STRING( 11 - nResto, "0" )
    End If

    If cResto <> UPPER( cDig ) Then
        lOk = FALSE
    End If
```

End If

Return ( lOk )