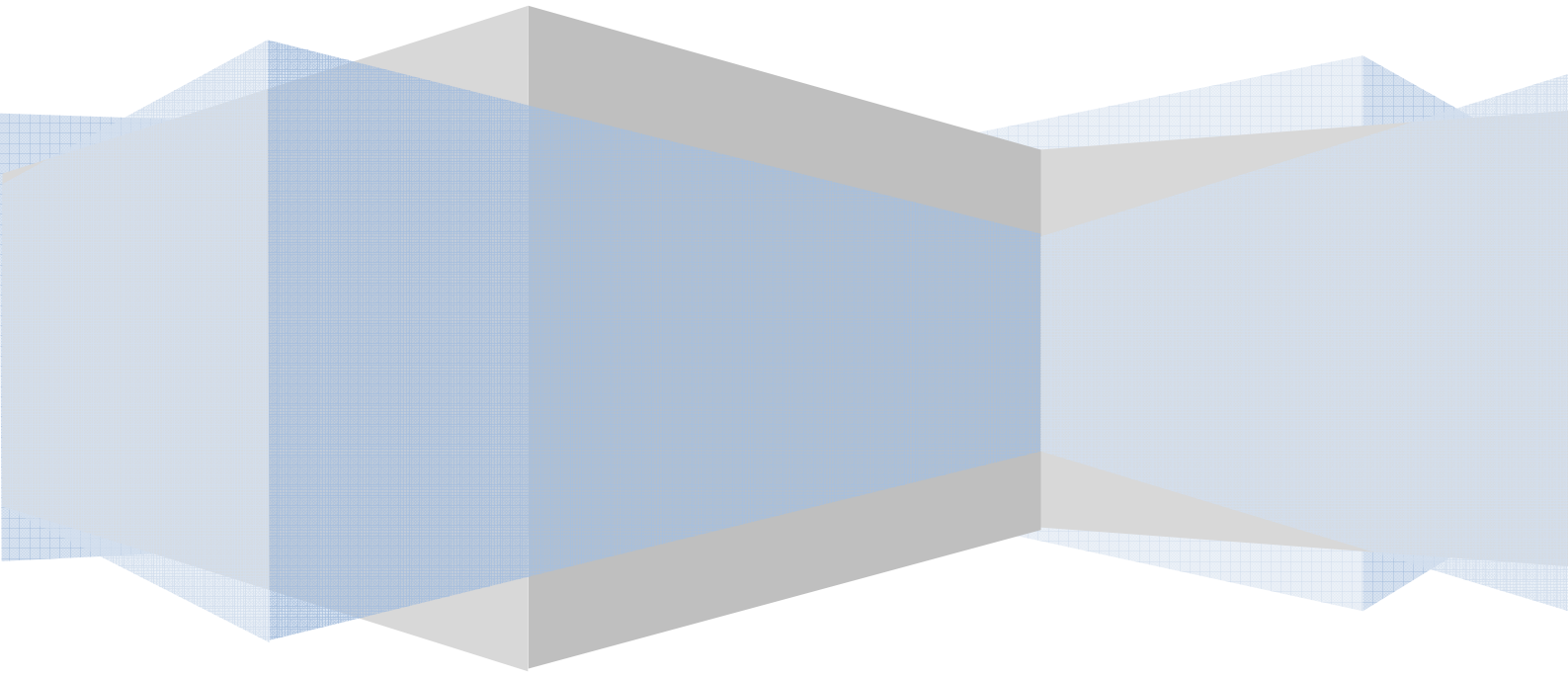
Framework GRIAL

Estándares y Manual de Programación



Versión 18/12/2005

Indice

Sección 1. Estándares

Capítulo I. Nominación de Tablas, Vistas, Objetos

Parte 1. Información General

Es importante comprender perfectamente la normativa de nominación de objetos utilizada en el desarrollo para comprender la distribución funcional de los objetos en la base de datos.

Mediante el conocimiento del nombre cualquier función, módulo u objeto de la base de datos, es posible establecer su ubicación en el esquema general del sistema, su función y su interdependencia con otros objetos de la base.

Parte 2. Nominación de las Tablas

Todas las tablas del sistema se nominan de acuerdo al siguiente esquema:

{sst}_{subg}_{nn}_{tpo}_{descripción}

Utilizaremos como ejemplo la tabla: PRE_EJEC_20_CAB_COMPROMETIDO

conformándose cada grupo como sigue:

{sst}, Sistema, (caracteres 1 a 3): determinan el sistema al que pertenece la tabla, en nuestro ejemplo “PRE” (Presupuesto)

{subg}, Subgrupo (caracteres 5 a 8): determinan el subgrupo funcional dentro el sistema al cual pertenece la tabla, en nuestro ejemplo “EJEC” (Ejecución Presupuestaria)

{nn}, Número secuencial, (caracteres 10 a 11): es un número secuencial dentro de cada sistema y subgrupo, en nuestro ejemplo “01”. De esta manera, la combinación de {sst}_{subg}_{nn} determinan un nombre único para la tabla y su sinónimo público. Todas las referencias a la tabla se realizan mediante su sinónimo público, en nuestro ejemplo: “PRE_EJEC_01”

{tpo}, Tipo (caracteres 13 a 15): determinan el tipo de tabla en cuanto a su importancia y relaciones con otras tablas. El Tipo puede ser:

- TBL: Tabla de códigos, independiente. Contiene información básica del sistema y es referenciada de una o más tablas específicas. Ejemplo: PRE_FORM_01_TBL_MEDIDAS (Tabla de Unidades de Medida)
- CAB: Tabla de cabecera, independiente. Contiene generalmente la cabecera de un documento o transacción. Existirá una o varias tablas de detalle (tipo DET) que la referencien. Ejemplo: PRE_EJEC_20_CAB_COMPROMETIDO (Tabla de Documentos de Compromiso)
- DET: Tabla de detalle, dependiente de una tabla Tipo CAB. Contiene generalmente los detalles de un documento o transacción. Existirá una tabla de cabecera (tipo CAB) referenciada desde esta tabla. Ejemplo: PRE_EJEC_21_DET_COMPROMISOS (Tabla de Detalles de los Documentos de Compromiso)
- MOV: Tabla de movimientos, dependiente de más de una tabla Tipo CAB o DET. Contiene generalmente los detalles que resultan de la interrelación de mas de un documento o transacción. Existirá en esta tabla una referencia a más de una tabla, de la cuales depende. Ejemplo: PRE_ASIE_03_MOV_BANCOS (Relación entre los bancos y las cuentas Contables)
- ARB: Tabla de estructura jerárquica (árbol). Contiene una estructura en forma de árbol, por lo que contendrá una referencia así misma (campo RELA_PADRE) Ejemplo: PRE_FORM_02_ARB_INSTITUCIONES (Tabla del Clasificador Institucional)

{descripción}, Descripción (resto de los caracteres): Descripción libre aclaratoria de la función de la tabla.

Parte 3. Clave primaria por sistema unificado

Todas las tablas del sistema se identifican por una clave primaria normada, la cual es generada automáticamente en el momento de inserción del registro. Dada esta normalización no se permite el establecimiento de claves primarias por metadato (más de un campo o campos e información del usuario). Al ser la clave un único campo numérico se maximiza el tiempo de respuesta en las operaciones de indexado y resolución de vistas en la Base de Datos.

El campo de clave primaria se compone del prefijo "ID" seguido del sinónimo público de la tabla. Para nuestro ejemplo, la clave primaria de la tabla PRE_EJEC_20_CAB_COMPROMETIDO es el campo ID_PREEJEC20.

Parte 4. Campos de relación

Al estar todas las tablas del sistema identificadas por una clave primaria normada, los campos de relación entre tablas responde al mismo formato y son por lo tanto únicos también. (Para relacionar una tabla con cualquier otra es necesario sólo un campo de relación)

Todo campo de relación se compone del prefijo "RELA" seguido del sinónimo público de la tabla referenciada, permitiéndose así un rápido reconocimiento de la estructura relacionada de cualquier tabla, y también así facilitando la creación de vistas y consultas.

Por ejemplo la tabla PRE_EJEC_21_DET_COMPROMETIDO tendrá un campo relacionándola con su tabla cabecera (PRE_EJEC_20_CAB_COMPROMETIDO). El campo de relación se denominará RELA_PREEJEC20.

Parte 5. Ordenamiento

Por convención se colocarán al principio de la tabla todos los campos de relación (prefijo RELA), luego se colocará el campo de identificación (clave primaria, prefijo ID) y luego los campos que representen los atributos específicos de la tabla. Este ordenamiento facilita la identificación de la estructura relacionada de cualquier tabla.

Parte 6. Nominación de Vistas

Las vistas se nominarán de acuerdo a su tabla troncal. La tabla troncal de una vista es aquella que tiene concordancia *uno a uno* de registros con la vista. El ID de la tabla troncal *no se repite* en la vista. Por cada registro de la tabla troncal hay un solo registro en la vista.

Luego del nombre de la tabla troncal se agrega la letra "W" (por vieW) y un underscore '_' y los números correspondientes a las demás tablas involucradas, si existe una tabla con un prefijo diferente a la que la precede en la lista de números, se colocarán las iniciales del prefijo antes del número.

Por ejemplo:

PRE_FORM_17W_15_12_SU02_15

Es una vista con la tabla PRE_FORM_17 como tabla troncal, con joins a las tablas PRE_FORM_15 y PRE_FORM_12, y además joins a las tablas SOC_USUA_02 y SOC_USUA_15.

El orden en el que se listan las tablas joineadas a la tabla troncal con el mismo prefijo deberá adecuarse en lo posible al camino de obtención de los datos. En el caso presentado: "PRE_FORM_17W_15_12_SU02_15" se entiende que desde la PRE_FORM_17 se accede a la PRE_FORM_15, y desde la PRE_FORM_15 se accede a la PRE_FORM_12. Las tablas accesorias de otro esquema (en este caso SU02_15) se colocan al final.

La identificación de la tabla troncal de la vista es muy importante. Por ejemplo, supongamos que la tabla PRE_EJEC_10, es cabecera de la tabla PRE_EJEC_12. (relación uno a muchos)

La tabla PRE_EJEC_12 contiene un campo PREEJEC12_IMPORTE.

Si existiese una vista llamada: **PRE_EJEC_10W_12**, se entiende que la tabla troncal será la PRE_EJEC_10, y que además contiene datos de la PRE_EJEC_12.

Siendo la PRE_EJEC_10 la troncal, existirá en la vista un registro por cada registro de la PRE_EJEC_10 (cabecera) y entonces, los campos correspondientes a la PRE_EJEC_12 (detalle), estarán sumados, o agrupados de alguna manera.

Es de esperar en esta vista de encontrarse con un campo SUM_PREEJEC12_IMPORTE que correspondería a la suma de los importes del detalle.

En cambio, en una vista llamada **PRE_EJEC_12W_10**, se entiende que la tabla troncal es la PRE_EJEC_12, y que además incorpora datos de la PRE_EJEC_10 (cabecera). Se entiende entonces, que existe un registro en la vista por cada registro en la PRE_EJEC_12, y por lo tanto los datos correspondientes a la cabecera (PRE_EJEC_10) se repetirán en todos los registros del detalle.

No se espera ningún campo agrupado en esta vista.

Capítulo II. Nominación extendida

Parte 1. "E", por Extendida, Tablas Implícitas

Cuando las tablas participantes son demasiadas o no es necesario especificarlas por no ser relevantes para la vista, se puede agrega la letra **"E"** al final de la tabla principal del grupo. Se entenderá **"E"** por "Extendido".

Por Ejemplo:

Nombre de la vista: EJE_REND_01_SU02E

Se entiende: la tabla EJE_REND_01 como troncal, con datos extendidos de la SOC_USUA_02. Conociendo el modelo está implícito que los datos extendidos de la SU02 (Puesto de un usuario del sistema) son las tablas SU01, SU15, SU08, PF11 y PF12. (Usuario, Entidad, Documento, U.Orgánica y Cargo)

Parte 2. "O" Outer Join

Cuando una de las tablas participantes este joineada a la tabla troncal mediante un outer join, se agregará la letra "O" DELANTE de la tabla joineada. Es necesario para diferenciar vistas con y sin outer joins.

Por Ejemplo:

PRE_EJEC_21W_22 <- Todos los registros de la 21 con relación en la 22

PRE_EJEC_21W_O22 <- Todos los registros de la 21 tengan o no 22

Parte 3. Campos en las vistas

3.1. Resultantes de Operaciones de agrupación (aggregate functions)

Los campos que resulten de operaciones de agrupación, se nominarán agregándole la operación como prefijo

Por ejemplo

Sum (PREEJEC12_IMPORTE) as SUM_PREEJEC12_IMPORTE

Count (ID_SYSFUNCC03) as COUNT_ID_SYSFUNC03

3.2. Resultantes de Decodificaciones (DECODE)

Si una tabla tiene un campo _COD o _TIPO, codificado con valores 0..n, se creará una vista con el nombre de la tabla + "W" en la cual se incluye un campo con el decode para la descripción del campo codificado. Al campo calculado con decode se le agrega el sufijo _TEXT.

Por Ejemplo, la tabla EJE_ABAS_01 tiene un campo EJEABAS01_TIPO, codificado como: 0=Proveedor, 1=Responsable de Fondo. Se creará una vista EJE_ABAS_01W, de la siguiente forma:

```
CREATE VIEW EJE_ABAS_01W  AS
    Select
    RELA_xxxx…
    ,ID_EJEABAS01
    ,EJEABAS01_DESCRI
    ,EJEABAS01_TIPO
    ,Decode(EJEABAS01_TIPO
      ,0,'Proveedor',
       ,1,'Responsable Fondo'
      ,'Tipo '||EJEABAS01_TIPO) as EJEABAS01_TIPO_TEXT
    From EJE_ABAS_01
```

A partir de aquí se utilizará esta vista cada vez que se necesite tener la descripción del campo codificado. De esta manera, el decode para el campo se encuentra en un solo lugar.

3.3. Campos RELA existentes en más de una tabla

Es posible que se repita un campo RELA entre las tablas que componen la vista. Se nominarán agregando como prefijo el nombre abreviado de la tabla *para las tablas no-troncales.*

Por Ejemplo:

```
Create View PRE_EJEC_10W_11_12_02 as
    Select
    PRE_EJEC_10.RELA_SOCUSUA02 (rela en la tabla troncal, sin cambios)
    ,PRE_EJEC_11.RELA_SOCUSUA02 as PE11_RELA_SOCUSUA02 (tabla secundaria)
    ,PRE_EJEC_02.RELA_SOCUSUA02 as PE02_RELA_SOCUSUA02 (otra tabla secundaria)
```

Capítulo III. Consideraciones Especiales

Parte 1. Campos de la Pre_Form_07w_13

Siendo muy probable que una vista medianamente compleja contenga dos o tres veces esta tabla, se nominarán los campos utilizando una abreviatura del metatipo del clasificador involucrado, por ejemplo:

PF07_PRG_CODIGO -> PREFORM07_CODIGO de un clasificador de programas
PF07_PRG_DESCRI -> PREFORM13_DESCRI de un clasificador de programas

PF07_OG_DESCRI -> PREFORM13_DESCRI de un Objeto del Gasto

Parte 2. Tablas con nombres de campo no normalizados

2.1. Tabla PRE_FORM_15 (Conceptos del Gasto)

Esta tabla contiene campos con nombres iguales a los de la tabla PRE_FORM_07, para evitar confusiones, usar la vista **PRE_FORM_15N**, que posee los campos normalizados.

2.2. Tablas PRE_FORM_01 y PRE_FORM_02 (Períodos e Instituciones)

Estas tablas contienen campos con nombres no normalizados, usar las vistas **PRE_FORM_01N y PRE_FORM_02N** que poseen los campos normalizados.

2.3. Tabla PRE_FORM_03 (Mov Período/Institución)

Esta tabla contiene campos con nombres no normalizados, usar las vistas **PRE_FORM_03N, PRE_FORM_03W_02N y PRE_FORM_03W_02_01** que poseen los campos normalizados.

2.4. Otras Tablas Normalizadas por una Vista "N"

PRE_FORM_07N, PRE_FORM_08N, PRE_FORM_07W_13, PRE_FORM_07WN_05_18, PRE_FORM_11N (Orgánicas), PRE_FORM_12N (Cargos), PRE_FORM_16N, SOC_USUA_01N (Usuarios), SOC_USUA_02N (Identidades de usuarios), SOC_USUA_15N (Entidades), PRE_FORM_05N, PRE_FORM_06N, PRE_BIEN_02N

2.5. Regla General

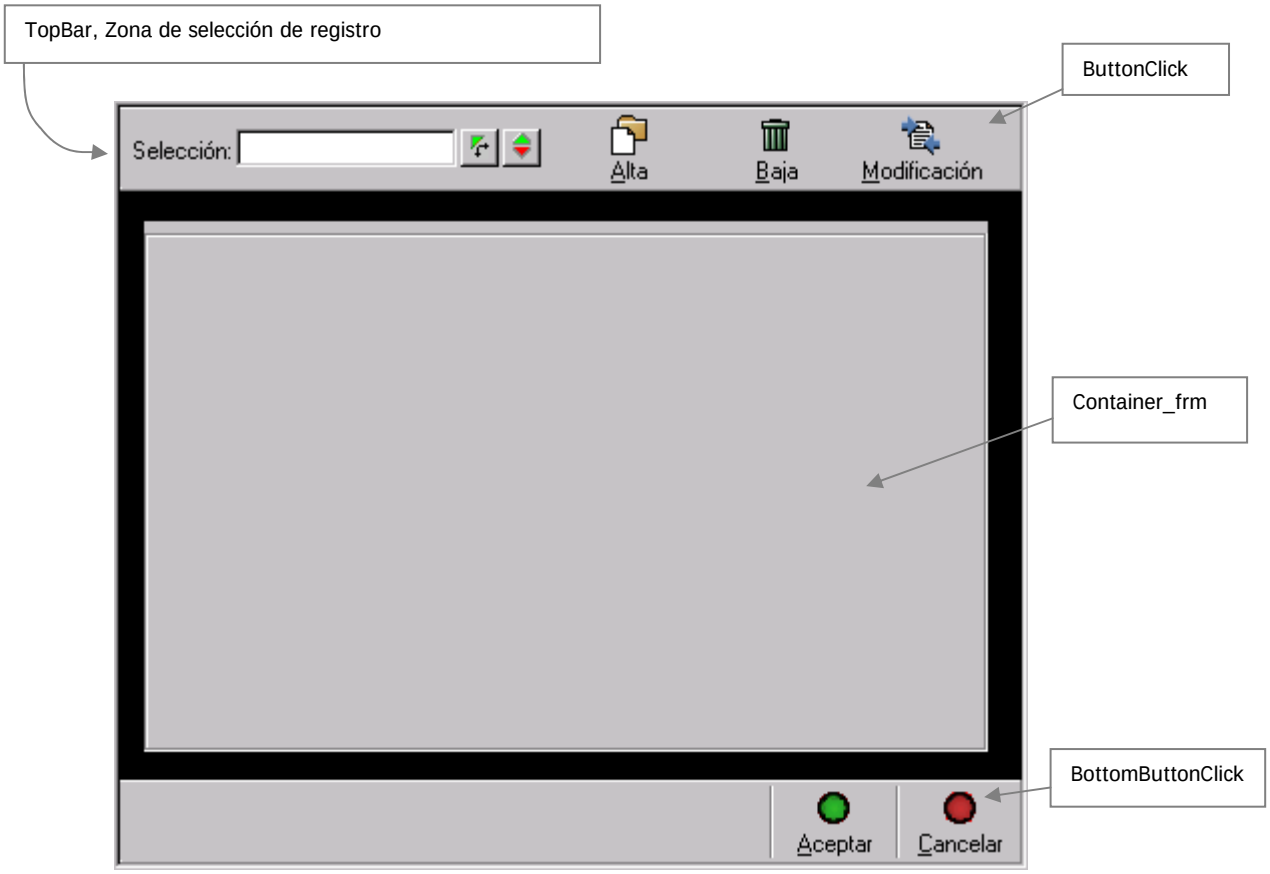
En caso de hallar una tabla con campos no normalizados, utilizar la vista "**N**" correspondiente con los campos normalizados. Usar el nombre de la tabla seguido de la letra "N" cuando se realiza una vista que sólo normalizan los nombres de los campos de una tabla.

Sección 2.
Controles Grial

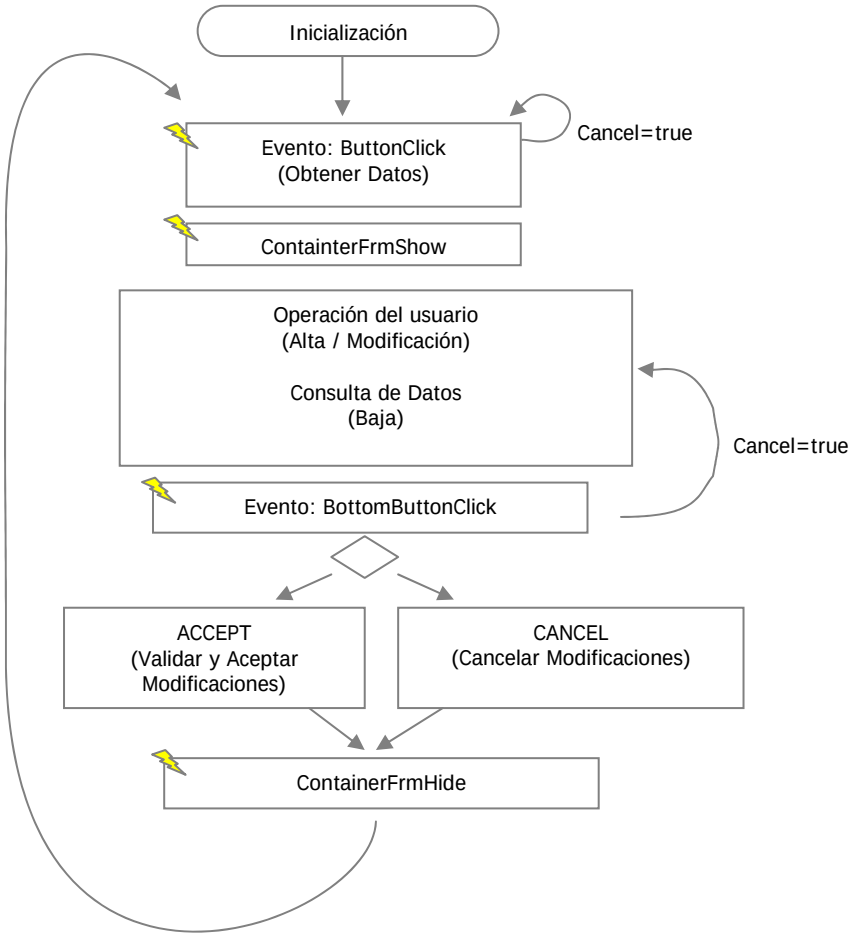
Capítulo IV. Grial Apl, GrialCont

Parte 1. Descripción General

El Contenedor representa *visualmente* una transacción, estandarizando el flujo del programa en todos los módulos del sistema. Adicionalmente brinda el servicio de seguridad, conexión y acceso a los controles registrados.



El flujo de programa determinado por este control responde al siguiente esquema:



Parte 2. Codificación Estándar: Inicialización, Centrado y Finalización

2.1. Componentes Visuales Requeridos

Dentro de cada GrialCont debe colocarse un objeto "Frame" de VB con el nombre "Container_Frm", dicho frame contendrá todos los controles involucrados en la transacción. Al iniciarse una transacción, se hace visible el frame "Container_Frm" (y se dispara el evento ContainerFrmShow), permitiendo la operación del usuario.

Al finalizar la transacción, se oculta dicho frame (y se dispara el evento ContainerFrmHide).

Si al iniciar una transacción, durante el evento ButtonClick, se establece el estado de sólo visualización (NewState=VIEW_MODE), se mostrará el Container_Frm pero deshabilitado.

En caso de tener más de un objeto GrialCont dentro de un mismo módulo, debe crearse un array de frames Container_Frm(0..n), uno dentro de cada uno de los objetos GrialCont.

2.2. Método Initialize

Se debe invocar el método **GrialCont.Initialize** en el evento UserDocument_Show (primer evento una vez mostrada la pantalla del módulo).

Formato:

```
Sub GrialCont.Initialize ( UserDocument_Name As String, Parent_LocationURL As String )
```

Ejemplo:

```
GrialCont1.Initialize UserDocument.Name, Parent.LocationUrl
```

El método Initialize:

- Establece los accesos de seguridad según el usuario
- Prepara la conexión para el acceso a datos
- Prepara la configuración de los controles registrados

2.3. Método Center

Se debe invocar este método en el evento UserDocument_Resize para centrar el objeto que contenga al resto de los objetos dentro de la pantalla.

Formato:

```
Sub GrialCont.Center ( xViewportWidth As Integer, [GeneralContainer As Object] )
```

Ejemplo:

```
GrialCont.Center ViewportWidth, General_GFF
GrialCont.Center ViewportWidth, TabGeneral
```

En el parámetro [GeneralContainer As Object] se indica el objeto que contiene al resto de los objetos de la función, por ejemplo cuando la aplicación presenta inicialmente varios tabs con un contenedor distinto en cada uno de ellos, será el objeto "Tab" el que contenga toda la aplicación.

2.4. Método Finalize

Debe ser invocado en el evento UserDocument_Hide (último evento antes de cerrar el formulario). Realiza descargas de los objetos cargados. Su ausencia puede provocar fallas en la salida del módulo.

Formato:

```
Sub GrialCont.Finalize()
```

Ejemplo:

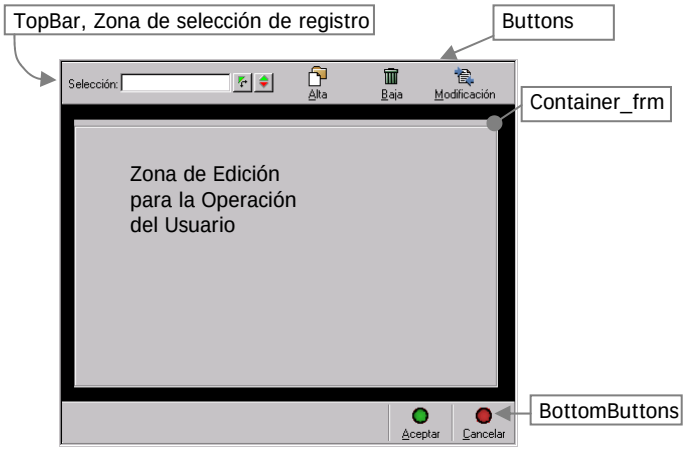
```
Sub UserDocument_Hide
    GrialCont.Finalize
End Sub
```

El método Finalize desconecta los controles registrados para evitar errores de protección general (GPF) en el cierre del documento.

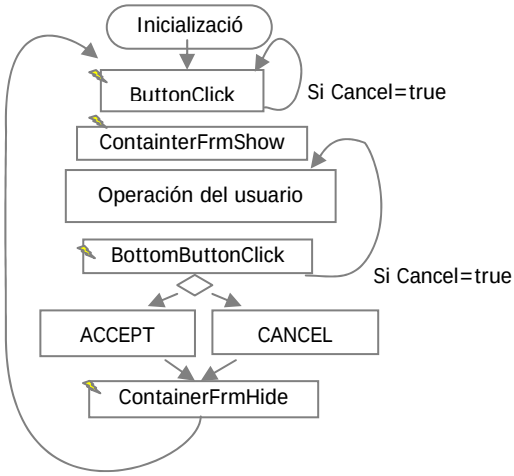
Parte 3. Flujo de Programas, de ButtonClick a BottomButtonClick

El flujo del programa estará determinado por los controles GrialApl incluidos en el módulo.
Se presenta a continuación un ejemplo de construcción y codificación de una transacción modelo dentro de un contenedor (GrialApl).

TRANSACCION (TXN) VISUAL



FLUJO DE EJECUCION



La transacción modelo consiste de un Combo de selección colocado sobre la barra superior de un control GrialApl. Dentro de la zona de edición se colocan los controles necesarios para modificar el conjunto de datos representado por el combo.

EJECUCION DE UNA TXN
DE MODIFICACION

Evento: ButtonClick, MODIFICATION

Sub PreparaTransaccion:
- Tomo el ID del combo de selección
- Cargo los Datos en Memoria desde la BD [1]
- Si es Alta, abro un nuevo registro

Sub MostrarDatos:
- Cargo los Datos en Pantalla desde Memoria [2] (usando AssignTextField)

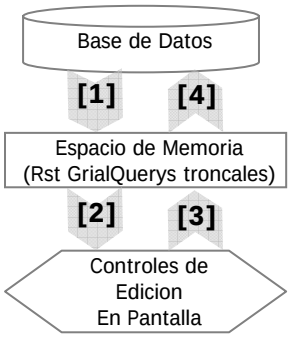
Operación del Usuario en Pantalla

Evento: BottomButtonClick, ACCEPT

Sub TomarDatosDePantalla:
- Cargo los Datos en Memoria desde Pantalla [3] (usando UpdateField y ChangeMainRst)

- Preparo la transacción (método Cmd)
- Envío la TXN a la base de datos [4]

MOVIMIENTO DE DATOS



3.1. Inicio de la Transacción

3.1.1 Evento ButtonClick

El evento ButtonClick determina el inicio de la transacción. En base a la acción elegida (Alta, Baja, Modificación, etc.) se prepararán los datos correspondientes para la transacción.

Formato:

```
Event ButtonClick ( GrialButtonCode As GrialButtons, Cancel As Integer
                  , NewState As Operation_States )
```

Ejemplo:

Se crea una variable global tipo long para almacenar el ID actual en edición (por ejemplo: "Dim IDPreEjec20 as Long", y una variable para llevar los contadores de falsos ID's para las tablas de detalles (Por Ejemplo: "Dim Falso_IDPreEjec21 as Long" si se crean detalles)

```
Dim IDPreEjec20 as long
Dim PreEjec20_grq as New GrialQuery
Dim Falso_IDPreEjec21 as long
Dim PreEjec21_grq as New GrialQuery
```

El usuario selecciona el registro sobre el cual desea operar en el combo, y luego pulsa uno de los botones de la barra superior (Alta, Modificación o Baja) lo que dispara un evento **GrialCont_ButtonClick**. En este evento se debe verificar que el usuario haya seleccionado correctamente un registro (para los casos de Modificación y Baja) y luego invocar un procedimiento para preparar la transacción (**Sub PrepararTransaccion**)

La variable global con el ID en edición se cargará al momento del evento **GrialCont_ButtonClick** con el ID del registro seleccionado en el combo (propiedad **CurrentValue** del combo). En caso de Alta de un nuevo registro, se utilizará el valor -1 como ID.

```
Private Sub GrialCont_ButtonClick(ByVal GrialButtonCode As GrialApp.GrialButtons,
                        Cancel As Integer, NewState As GrialApp.Operation_States)
On Error GoTo ErrH

   Select Case GrialButtonCode

      Case BUTTON_NEW
       IDPreEjec20 = -1
       PrepararTransaccion

      Case BUTTON_MODIFICATION,BUTTON_DELETE
          If NoDataIn(PreEjec20_scb.CurrentValue) Then
                ' Si no se ha elegido ningun registro, se cancela
                Cancel= True
                Exit Sub
          End if
          IDPreEjec20 = ID_PreEjec20_scb.CurrentValue
          PrepararTransaccion

      Case Else 'no es un boton implementado
         Cancel = True

   End Select

Exit Sub

' ------ Manejo de errores
ErrH:
   MsgError GrialCont  ' Le muestro el error al usuario
   Cancel = True ' Cancelo el evento, la pantalla permanecerá igual
                ' y el usuario podrá reintentar la operación
End Sub
```

El procedimiento "**PrepararTransaccion**" debe recuperar de la BD un recordset para cada tabla troncal afectada en la transacción. Se cargará un objeto **GrialQuery** con cada uno de los recordsets troncales necesarios para la transacción. En la condición "where" del recupero de datos se utilizará la variable global con el ID en edición.

Para el caso de alta:

Al ser el valor del ID "-1", se recuperarán automáticamente recordsets vacíos (ya que la condición será: Where ID_PREEJEC20 = -1), por lo tanto el procedimiento PrepararTransaccion se deberá invocar el método PrepareAddNew para el combo superior de selección y para cada GrialQuery de tabla troncal.

El método PrepareAddNew crea un espacio en el recordset donde el módulo puede trabajar. Todos los campos son inicializados con el valor NULL, por lo que al mostrar los datos en pantalla, se mostraran todos los datos en blanco.

Luego de preparar el nuevo registro *deberán cargarse los campos que sólo son cargados en el alta*, es decir, aquellos que permanecen inalterables durante toda la existencia del registro. Por lo general estos campos son el ID de la tabla, la fecha de Alta y los campos RELA a la cabecera para las tablas detalle.

Es recomendable realizar la carga de los campos inalterables en este punto y sólo para el caso de alta en lugar de hacerlo con el resto de los campos en el momento de aceptar la transacción. De esta manera se evita la posibilidad de modificar dichos campos sobre un registro existente por error de programa.

```
Private Sub PrepararTransaccion()
    PreEjec20_grq.Init "select * from sic_docu_01 where ID_PREEJEC20=" & IDPreEjec20
    PreEjec21_grq.Init "select * from sic_docu_02 where RELA_PREEJEC20=" & IDPreEjec20
    PreEjec21_scb.ExtraFilterCondition = "RELA_PREEJEC20=" & IDPreEjec20"  'Grilla Detalle
    Grial_Cont.LoadData PreEjec20_grq, PreEjec21_grq, PreEjec21_scb
    If IDPreEjec20 = -1 Then ' Alta
      PreEjec20_scb.PrepareAddNew    ' Combo superior
      PreEjec20_grq.PrepareAddNew    'Recordset Troncal en memoria
      With PreEjec20_grq.Rst
          !ID_PREEJEC20 = -1          'Datos que solo se asignan en un Alta
          !PREEJEC20_FAPL = Now
      End With
    End If
    PonerEnPantalla 'Pongo troncal y detalle en pantalla
End Sub
```

Al final del procedimiento **PrepararTransaccion** tendremos los espacios necesarios en memoria (en los registros dentro del Combo y de los recordsets troncales) para operar la transacción. Tendremos entonces la información completa del registro recuperado o un registro en blanco para el caso de alta.

En este momento invocaremos un procedimiento para volcar los datos desde el espacio de memoria (registros en los recordsets troncales) hacia los controles de edición en la pantalla. Por norma lo llamaremos **PonerEnPantalla**.

```
Private Sub PonerEnPantalla()
    With PreEjec20_grq.Rst
      AssignTextField PreEjec20_Descri_txt, !PREEJEC20_DESCRI
      AssignTextField PreEjec20_Cod_txt, !PREEJEC20_COD
      SysWflo06_scb.CurrentValue =!RELA_SYSWFLO06
      Fapl_Lb = FormatDate( !PREEJEC20_FAPL )
    End With
End Sub
```

En este procedimiento se usarán la rutinas **AssignTextField** para volcar datos a controles de texto y **FormatDate** para volcar datos de tipo fecha. Por norma, los controles de tipo Textbox que se utilizan para editar campos de la base, deben tener el mismo nombre del campo con el sufijo _txt adicionado, por ejemplo **PreEjec20_Descri_txt.**

Una vez mostrados los datos, el usuario podrá operar con la pantalla de la transacción, hasta que pulse alguno de los botones de la barra inferior (Aceptar o Cancelar) disparándose un evento **GrialCont_BottomButtonClick**

3.1.2 Evento ContainerFrmShow

El evento **ContainerFrmShow** sucede posteriormente al inicio de la transacción (Evento **ButtonClick**) y antes de hacer visibles los controles dentro del GrialCont. Debe utilizarse para realizar ajustes en la pantalla según los datos recuperados de la transacción y habilitar o deshabilitar controles.

El evento **ContainerFrmShow** se ejecuta si y sólo si se ha iniciado satisfactoriamente la transacción, y se está por mostrar la pantalla (es decir, si NO se indicó cancel=true en el evento ButtonClick). El parámetro NewState informa estado elegido durante el evento ButtonClick. Por

default toma el valor CONTROLS_ENABLED para el botón de Alta y Modificación y el valor VIEW_MODE (todos los controles deshabilitados) para el botón de Baja.

Dentro de este procedimiento debe deshabilitarse el combo superior, para que el usuario no pueda cambiar el registro elegido mientras está dentro de la transacción.

Formato:

Event ContainerFrmShow (NewState As Operation_States)

Ejemplo:

```
Private Sub GrialCont_ContainerFrmShow (NewState As Operation_States)
  PreEjec20_scb.Enabled = False
End sub
```

3.2. Operación del Usuario

Mientras se encuentre habilitado el GrialCont, el usuario podrá operar con los controles de la pantalla.

Los botones superiores del GrialCont se deshabilitan automáticamente y se habilitan los botones inferiores, permitiendo al usuario dos opciones: ACPETAR o CANCELAR.

3.3. Cierre de la Transacción

El evento BottomButtonClick determina la finalización de la transacción. Se dispara cuando el usuario hace clic en uno de los botones de la barra inferior (ACEPTAR o CANCELAR). Dependiendo de la acción elegida (BUTTON_ACCEPT o BUTTON_CANCEL) se validarán y aceptarán los datos o se cancelará completamente la transacción.

En caso que el usuario haya pulsado el botón ACEPTAR se procederá a la validación de los datos ingresados y a su transferencia hacia los recordsets troncales en memoria.

La validación y la transferencia de los datos en pantalla hacia los espacios de memoria en el recordset troncal se realizan en un procedimiento que llamaremos **TomarDatosdePantalla**.

3.3.1 Procedimiento TomarDatosdePantalla

Dentro de este procedimiento usaremos:

- Para los campos individuales, la rutina **UpdateField** y **UpdateFieldNum** que permiten actualizar un campo de un recordset y validar por contenido nulo en la misma operación.
- Para recuperar datos de una Grilla editable en pantalla, el procedimiento **Grid_UpdateUserInput** para cerrar el modo edición de la grilla y luego **ChangeMainRst** para actualizar los recordset troncales desde los registros de la grilla en pantalla.

```
Private Sub TomarDatosdePantalla()
    With PreEjec20_grq.Rst
      If ValCur(PreEjec20_Porcentaje_Txt) > 100 then
          Err.Raise 5,,"El Porcentaje debe ser menor que 100"
      End if
      UpdateField !PREEJEC20_PORCENTAJE, valcur(PreEjec20_Porcentaje_Txt)
      UpdateField !PREEJEC20_DESCRI, PreEjec20_Descri_txt,  "una descripción"
      UpdateField !RELA_SYSWFLO06, SysWflo06_scb.CurrentValue
    End With

    ' Aplico los cambios realizados por el usuario en la grilla en el recordset troncal
    PreEjec21_Scb.Grid_UpdateUserInput
    GrialCont.ChangeMainRst PreEjec21_qrq.Rst, PreEjec21_Scb.Rst, "ID_PREEJEC21"
End Sub
```

(Nota: Todas las validaciones deben realizarse dentro este procedimiento y sólo si el usuario pulsa el botón "Aceptar". NO deben realizarse validaciones en eventos del tipo "_LostFocus" o "_Change").

3.3.2 Evento BottomButtonClick

En el evento **GrialCont_BottomButtonClick** creamos un objeto del tipo **GrialTransaction** (o su equivalente **SQLQuery**) para preparar la transacción que enviaremos al servidor.

Ejemplo:

```
Private Sub GrialCont_BottomButtonClick(ByVal GrialButtonCode As GrialApp.GrialButtons, Cancel As
Integer, NewState As GrialApp.Operation_States)

On Error GoTo ErrH
Dim Txn As New GrialTransaction

   If GrialButtonCode = BUTTON_ACCEPT Then

      Select Case GrialCont.LastButton
         Case BUTTON_NEW, BUTTON_MODIFICATION
                  ‘ Validación y toma de datos
                  TomarDatosdePantalla
                  ‘ Preparo la transaccion
                  Txn.Cmd PreEjec20_Grq,”ID_PREEJEC20”      ‘ Cabecera
                  Txn.Cmd PreEjec21_Grq,”;RELA_PREEJEC20”   ‘ Detalle
                  ‘ Aplico la transacción en el servidor
                  GrialCont.Apply Txn

                  If  GrialCont.LastButton = BUTTON_NEW then
                        IDPreEjec20 = PreEjec20_grq.NewIDValue ‘Recupero el nuevo ID creado
                  End If;
                  ‘ Refresco los cambios en el combo superior
                  PreEjec20_scb.LoadOnCurrentRecord IDPreEjec20 ‘ Refresco el registro del combo

      Case BUTTON_DELETE
                  PreEjec20_Grq.DeleteRecord
                  PreEjec21_Grq.DeleteAllRecords
                  ‘ Preparo la transaccion
                  Txn.Cmd PreEjec20_Grq.Rst ‘ Cabecera
                  Txn.Cmd PreEjec21_Grq.Rst ‘ Detalle
                  ‘ Aplico la transacción en el servidor
                  GrialCont.Apply Txn
                  PreEjec20_scb.DeleteRecord  ‘Elimino registro del combo superior
      End Select

End If

Exit Sub

' ------ Manejo de errores
ErrH:
  MsgError GrialCont ' Le muestro el error al usuario
  If Txn.Commited then resume Next ‘ Si ya commiteo, ignoro el error
  Cancel = True  ' Cancelo el evento de cierre de la pantalla
 ' Al cancelar el BottomButtonClick, la pantalla permanecerá igual,
 ' y el usuario podrá reintentar la operación
End Sub
```

3.3.3 Descripción de la operatoria

Luego de recuperar los datos de pantalla, se confecciona la transacción dentro de un objeto **GrianTransaction** mediante el método **Cmd**. Tanto para el Alta de registros como para Modificación, la transacción armada y enviada al servidor es la misma. Los registros deben enviarse en orden, primero los registros de cabecera y luego los registros de detalle y se debe indicar la relación entre los mismos (que campo es ID y qué campo es RELA)

Para el caso de Baja, se eliminarán los registros correspondientes de los recordsets troncales y se envían los recordsets al servidor. Los registros deben enviarse en orden, primero los registros de cabecera y luego los registros de detall, pero no es necesario para la eliminación física de registros especificar la información campos ID y RELA.

La baja física debe utilizarse en caso de excepción. Los controles de FK (Foreign Keys) en la base de Datos evitan que se elimine físicamente un registro si es referenciado desde otra tabla. Este control es realizado automáticamente por la BD.

Si se desea realizar una baja lógica o anulación, debe permitirse al usuario marcar/desmarcar un checkbox de fecha de baja y actualizar el campo de Fecha de Baja Lógica correspondiente en el registro. Un registro dado de baja lógica no podrá ser utilizado para relacionarlo desde otras tablas, pero sí debe aparecer en el combo de selección del módulo principal de la tabla, para permitir su recuperación (desmarcando el checkbox de Fecha de Baja).

En el control de errores del método **GrialCont_BottomButtonClick** debe tomarse en cuenta que el usuario podrá reintentar la transacción, por lo que deben dejarse los recordsets troncales en un estado adecuado como para permitir un reintento de la transacción.

3.4. Evento ContainerFrmHide

El evento **ContainerFrmHide** se dispara una vez cerrada la transacción (Luego del evento BottomButtonClick). Debe utilizarse para rehabilitar los controles deshabilitados el inicio de la edición (como un espejo del evento ContainerFrmShow).

Ejemplo:

```
Private Sub GrialCont_ContainerFrmHide(ByVal ActualState As GrialApp.Operation_States)
  PreEjec20_scb.Enabled = True
End sub
```

Parte 4. Botones Disponibles en la Transacción

En el Inicio del Módulo (evento UserDocument_Show) debe dejarse visibles sólo aquellos botones contemplados en el código

Ejemplo:

```
Private Sub UserDocument_Show ( . . .
. . .
With GrialCont
    .TopBar_AllButtonsVisible False  ' Todos invisibles
    ' Habilito sólo Alta, Baja y Modificación
    .ButtonStd_Visible BUTTON_MODIFICATION, True
    .ButtonStd_Visible BUTTON_NEW, True
    .ButtonStd_Visible BUTTON_DELETE, True
End With
```

También es posible cambiar la descripción de cualquiera de los botones.

Ejemplo:

```
GrialCont.ButtonStd_Caption BUTTON_MODIFICATION, "Seleccionar"
GrialCont.ButtonStd_Caption BUTTON_CANCEL, "Cerrar"
```

Parte 5. Controles Registrados

Los controles registrados son configuraciones de Combos, Grillas y Trees Grial almacenadas en el servidor de datos. La configuración le indica al control que query realizar para cargar los datos y cómo mostrarlos.

Se puede asignar una configuración a un control en un módulo simplemente asignándole la propiedad Name o por la propiedad ConfigurationName. La propiedad ConfigurationName permite tener dos controles (con nombres diferentes) pero que utilicen la misma configuración registrada.

El sufijo del nombre del control registrado determina su tipo:

_scb: Standard Combo/Grid
_ste: Standard Tree Controller
_sgr: Standard Grid Controller

Todos los controles de un documento que posean estos sufijos serán buscados en el repositorio de Controles Registrados para asignarles la configuración correspondiente.

Nota: otros sufijos estándar son:

_grq: Grial Query
_gpr: Grial Print
_ggp: Grial Grid Print
_gct: Grial Cross Tree
_tvw: TreeView
_grx o _grd: Grilla (GridEX de Janus)
Rst: Recordset
Txt: Text Box
Lb : Label
Cmd: Command Button
Opt: Option Button
Chk: CheckBox

Los controles registrados se cargan con datos mediante el método **GrialCont.LoadData**. Este método acepta una lista de controles registrados (y Objetos **GrialQuery**) y los carga según el query de la configuración realizando un único acceso al servidor.

Formato:

Sub GrialCont.LoadData (ParamArray RegistredControls() As Variant)

Ejemplo:

```
PreEjec20_grq.Init "Select * from PRE_EJEC_20 where ID_PREEJEC20=" & IDPreEjec20
GrialCont.LoadData PreEjec20_grq, ID_PreEjec20_Scb, PreEjec21_scb
```

Parte 6. Manejo de Recordsets, Objeto GrialQuery

El Objeto **GrialQuery** representa el Recordset resultante luego de la ejecución de un comando. Puede definirse el comando con lenguaje SQL o cualquiera de las extensiones aceptadas por el **GrialDataServer**.
El objeto **GrialQuery** se inicializa con el comando a ejecutar (método **Init**) y puede ser cargado mediante el método GrialCont,LoadData.

Por ejemplo:

```
Dim PreForm23Grq as New GrialQuery
PreForm23Grq.Init "Select * From PreForm23 where RELA_PREFORM22=" & IDPreform22Elegido
GrialCont.LoadData PreEjec21_Scb, PreForm23Grq
If PreForm23Grq.RecCount = 0 then Err.Raise 5,"No se hallaron datos"
```

El objeto **GrialQuery** posee una serie de métodos y propiedades para el manejo del Rst interno, que simplifican manejo de recordsets ADO.

Propiedades:

CurrentQuery: Propiedad tipo string conteniendo el texto del comando a ejecutar
Rst: Propiedad tipo recordset conteniendo el recordset resultante de la consulta.
RecCount: Cantidad de registros en Rst.

Métodos y Funciones:

Function Find(Criteria As String) As Boolean

Busca el primer registro que coincida con el criterio especificado, retorna False si no halla ningún registro.

Criteria, puede ser: "CAMPO [oper] [Valor]" donde la operación puede ser " =, > , < , >= , <= , LIKE", y pueden combinarse condiciones con los operadores AND y OR

Function Position (ID_Field As String, ID) As Boolean

Se posiciona en el registro que cumpla la condición "ID_FIELD = ID"

Function Sum(FieldName As String)

Retorna la suma los valores del campo indicado para todos los registros actuales

Function ValueList(FieldName As String, [Separ As String = ","])

Retorna un string con los valores del campo indicado, separados por coma. Es útil para conformar un filtro a se utilizado con la cláusula IN.

Por Ejemplo:

```
PreEjec21_scb.ExtraFilterCondition = _ "RELA_EJEREND01 IN ( " _
                                    & ER01Habilitadas_Grq.ValueList("ID_EJEREND01") & ")"
```

Sub MovePreStart() y Function NextRecord() As Boolean

MovePreStart posiciona el recordset en una posición anterior al primer registro para iniciar un recorrido de todos los registros. **NextRecord** avanza un registro y devuelve False en caso que no haya mas registros. Usados en conjunto facilitan un bucle de recorrido de todos los registros del recordset.

Por Ejemplo:

```
With ER01Habilitadas_Grq
    .MovePreStart
    While .Nextrecord
      'Imprimo cada registro...
      GrialPrint_Gpr.WtCol "Cod", .Rst!EJEREDN01_CODIGO
      GrialPrint_Gpr.WtCol "Descri", .Rst!EJEREDN01_DESCRI
      ...
    Wend
End With
```

Function Modified() As Boolean

Retorna true si se ha modificado algún dato dentro del recordset

Function PrepareAddNew() As Boolean

Crea un nuevo registro en blanco dentro del recordset

Sub DeleteRecord()

Elimina el registro actual del recordset

Sub DeleteAllRecords()

Elimina todos los registros del recordset

Todas las funciones de los recordsets ADO, pueden accederse directamente desde el objeto Rst

Por Ejemplo:

```
With ER01Habilitadas_Grq
    .Rst.Filter = "EJEREND01_PRINCIPAL = 1 " 'Solo los registros principales
    .Rst.Sort = "EJEREND01_CODIGO, EJEREND01_FAPL " 'Ordenado por codigo y Fecha
    .MovePreStart
```

```
While .Nextrecord
  'Imprimo cada registro...
  GrialPrint_Gpr.WtCol "Cod", .Rst!EJEREDN01_CODIGO
  GrialPrint_Gpr.WtCol "Descri", .Rst!EJEREDN01_DESCRI
  ...
Wend
End With
```

Parte 7. Recordsets Troncales

Para el manejo y actualización de datos dentro del entorno grial, definiremos los conceptos de "ViewRecordset" o "Recordset de Visualización" y "MainRecordset" o "Recordset Troncal".

- Un "ViewRecordset" surge de una vista definida en la base de datos (Conjunción de varias tablas) y es mostrado al usuario a través de un Control Grial (Combo, Tree o Grilla).
- Un Recordset "Troncal" contiene generalmente datos de una sola tabla, es recuperado por programa mediante un objeto GrialQuery, y es utilizado internamente para actualizar datos y preparar la transacción para el servidor.

Esta diferenciación es necesaria ya que los ViewRecordset por lo general no son "updateables", es decir, al ser conjunciones de varias tablas, no pueden ser modificados para actualizar la base de datos. En cambio, un recordset "troncal" contiene los datos de una única tabla, incluyendo su ID, por lo que las modificaciones puede ser aplicadas a la Base de Datos. Es este recordset "Troncal" es el que es enviado en la transacción.

Como el usuario ve y opera con los ViewRecordset , es necesario al aceptar la transacción transferir los cambios realizados por el usuario hacia el recordset Troncal. Para esto se utiliza un método del GrialCont llamado "ChangeMainRst"

Formato:

Sub ChangeMainRst (MainRecordset As Recordset, ViewRecordset As Recordset, ID_Field As String, [CreateRecords As Boolean = Falso], [PreCancelModifications As Boolean = Verdadero])

El parámetro ID_Field es el nombre del campo que vincula los registros en el Troncal con los registros en el ViewRecordset. En todos los casos, la tabla principal de la vista debe ser la misma tabla del recordset troncal, y también todos los campos que se deseen actualizar deben tener el mismo nombre en la vista y en el troncal.

Todas las modificaciones, altas y bajas realizadas sobre el ViewRecordset por el usuario serán duplicadas en el Troncal, sincronizando los registros que posean el mismo valor del campo ID_Field, y transfiriendo todos los valores entre los campos de igual nombre.

Ejemplo:

```
Dim PreForm23Grq as New GrialQuery
Dim IDPreform19Elegido as Long

Sub PrepararTransaccion…
PreForm20Grq.Init "Select * From PreForm20 " & _
          " where RELA_PREFORM19=" & IDPreform19Elegido
GrialCont.LoadData ID_Periodos_Scb, Instituciones_Ste, PreForm23Grq

Sub BottomButtonClick….
Dim Txn As New GrialTransaction
' Actualizo el recordset troncal
GrialCont.ChangeMainRst PreForm20_grq.Rst, PreForm20_sgr.Rst, "ID_PREFORM20"
Txn.Cmd PreForm20_grq.Rst
Txn.Apply GrialCont
```

Parte 8. Actualización de Datos: Objeto GrialTransaction, Objeto SqlQuery

Las transacciones en el Entorno Grial son preparadas en el cliente, y luego enviadas como un conjunto único al Servidor intermedio. El objeto utilizado para componer una transacción es el GrialTransaction. Los objetos de esta clase funcionan como una "pila" en la cual se acumulan los ítems que conforman la transacción recordsets con modificaciones y/o comandos SQL (con las extensiones aceptadas por el GrialDataServer).

Una vez armada la transacción, mediante el método GrialCont.Apply se envía el conjunto al servidor para su proceso.

Nota: El Objeto **SqlQuery** pertenece a una versión anterior y se mantiene por compatibilidad, posee las mismas funciones del objeto **GrialTransaction**, con la única diferencia del método para aplicar la transacción: El objeto **GrialTransaction** se aplica con el método **Apply** en tanto que en el `SqlQuery` se invoca el método "UpdateRecords"

La transacción se prepara mediante el método "Cmd", que es el que coloca en la pila cada uno de los ítems que la componen. Los ítems deben ser agregados en la transacción en un orden cabecera-detalle, haciendo seguir a cada recordset con registros de cabecera del recordset con registros de detalle dependientes de los anteriores. Por default, al aplicar la transacción, el GrialDataServer asume que los registros de cabecera de un ítem dado se hallan en el ítem inmediato anterior.

Formato:

Sub Cmd (SqlCmd_or_Recordset, [Rela_String As String])
Sub Item(SqlCmd_or_Recordset, [Rela_String As String])

Ejemplo:

```
Dim Txn As New GrialTransaction
Txn.Cmd PreForm20_grq.Rst , "ID_PREFORM20"
Txn.Cmd PreForm21_grq.Rst , "ID_PREFORM21;RELA_PREFORM20"
Txn.Cmd PreForm23_grq.Rst , ";RELA_PREFORM21"
' Aplico la transacción en el Servidor
Txn.Apply GrialCont
```

El primer parámetro del método **Cmd** es un recordset con las modificaciones realizadas o puede ser también un string conteniendo un query o un comando a ejecutar en la BD.

El segundo parámetro establece propiedades para el ítem dentro de la transacción, mediante una sintaxis especial. Cuenta con dos secciones separadas el carácter dos puntos (":"), la sección principal indica hasta tres nombres de campo, indicando qué campo es el ID de la tabla, cuál contiene la relación con el recordset de cabecera y opcionalmente que campo representa el RELA_PADRE para las tablas de tipo ARB (árboles). Estos tres nombres de campo se separan por punto y coma (" ; ").

Dentro de la transacción se asume que si en un **Cmd** se indica un campo RELA, la tabla cabecera es la indicada en el ítem inmediato anterior. Esto es consecuente con la necesidad de ordenar en la transacción primero las tablas cabeceras y luego los detalles.

Por ejemplo:

```
Txn.Cmd PreForm20_grq.Rst , "ID_PREFORM20" ' Cabecera
Txn.Cmd PreForm21_grq.Rst , " ID_PREFORM21 ; RELA_PREFORM20" ' Detalle de la 20
Txn.Cmd PreForm07_grq.Rst , "ID_PREFORM07 ; ; RELA_PADRE" ' Actualizo el arbol
```

Existe una sección de datos opcionales, que debe indicarse antes de la sección principal y se separa de la misma mediante dos puntos (":"). La sección opcional contiene propiedades para el ítem de la transacción y da la posibilidad de modificar el orden cabecera-detalle o de actualizar tablas con múltiples campos RELA.

Por ejemplo:

```
Txn.Cmd PreForm20_grq.Rst , "Name=PF20 : ID_PREFORM20 " ' Cabcera
Txn.Cmd PreForm07_grq.Rst , "ID_PREFORM07 ; ; RELA_PADRE" ' Actualizo el arbol
```

Txn.Cmd PreForm21_grq.Rst , "Header = PF20 : ID_PREFORM21 ; RELA_PREFORM20" 'Detalle

El Formato para el segundo parámetro del método **Cmd** es:

"[Param = valor, Param = valor... :] ID_FieldName ; RELA_FieldName ; RELA_PADRE"

Detalle de cada parte:

ID_FieldName:
Nombre del Campo que contiene el ID de la tabla. Se usará para tomar el valor que debe ser grabado en los campos RELA de los siguientes detalles.

RELA_FieldName:
Nombre del campo que contiene el ID de la cabecera (por default en el recordset anterior). La combinación de ID_FieldName de un ítem y el RELA_FieldName del ítem siguiente establecen una relación cabecera-detalle.

RELA_PADRE:
Cuando un recordset establece relaciones dentro de sí mismo. Por ejemplo una tabla tipo árbol. RELA_PADRE es el nombre del campo que representa la relación padre-hijo dentro de la jerarquía del árbol. La combinación ID_FieldName + RELA_PADRE en un mismo recordset se utiliza para las tablas tipo árbol.

Las posibles combinaciones para la sección opcional "Param = valor " son:

NAME={Identificador XXX} -- Identifica el ítem con un nombre

HEADER={ Identificador XXX } -- Indica en un ítem detalle, cual es el ítem (identificado previamente mediante NAME = XXX) ,que contiene los registros cabecera. Es necesario sólo cuando la cabecera y el detalle no se pueden colocar juntos en la transacción.

RelaField={FIELD_NAME}/{HEADER_NAME} -- Se usa para actualizar un registro que tiene más de un campo RELA. Reemplaza el valor del campo indicado por el valor de ID tomado del header indicado.

SysDateField={FIELD_NAME}[/{FIELD_NAME}...] : Se usa para los campos que deben ser actualizados con la fecha del sistema. Reemplaza los campos indicados por el Sysdate de la máquina donde corre el Servidor intermedio.

Debe indicarse para efectuar un registro preciso de fechas y horas, ya que la máquina cliente donde se prepara la transacción puede tener mal la fecha. En caso de un ALTA, se actualizarán los campos indicados, siempre que contengan valores no-nulos. En caso de MODIFICACION, se actualizarán los campos indicados con la fecha del sistema sólo si fue modificado su valor original

En el módulo cliente, debe indicarse siempre el "SysdateField=field1/field2/..." y se deben cargar/modificar los campos utilizando la función "Now" (sysdate de la terminal, valor no-confiable).

Cuando el servidor actualice el recordset en la base, reemplazará los valores por el sysdate del servidor (valor confiable).

Para los campos con nombre: FAPL, FBAJA, FANULADO, WFFECHACURSO, WFFECHA se realizará este control especial automáticamente.

8.1. Ejemplos de Armado y Aplicación de Transacciones

1.- CABECERA + DETALLE

```
Dim Cabecera_PAREXA02_grq as New GrialQuery
Dim Detalle_PAREXA03_grq as New GrialQuery
...

Dim Txn As New GrialTransaction

Txn.CMD Cabecera_PAREXA02_grq.Rst,"ID_PAREXA02"
Txn.CMD Detalle_PAREXA03_grq.Rst,";RELA_PAREXA02"

GrialCont.Apply Txn
```

2.- CABECERA + DETALLE + Detalle Extra

```
Dim Cabecera_EXACDA01_grq as New GrialQuery
Dim Detalle_EXACDA03_grq as New GrialQuery
Dim Detalle_EXACDA04_grq as New GrialQuery
...

Dim Txn As New GrialTransaction

Txn.CMD Cabecera_EXACDA02_grq.Rst,"Name=CabeceraGeneral:ID_EXACDA01"

' Este detalle toma el header por default
' (recordset en el CMD anterior)
Txn.CMD Detalle_EXACDA03_grq.Rst,";RELA_EXACDA01"

' Este detalle toma el ID del item “CabeceraGeneral”
Txn.CMD Detalle_EXACDA04_grq.Rst,"Header=CabeceraGeneral;RELA_EXACDA01"

Txn.Apply GrialCont
```

3.- CABECERA, DETALLE + Relación

```
Dim Tabla_REACDO12_grq as New GrialQuery
Dim Cabecera_EXACDA01_grq as New GrialQuery
Dim Detalle_EXACDA03_grq as New GrialQuery
...

Dim Txn As New GrialTransaction

SqlQ.CMD Tabla_REACDO12_grq.Rst,"Name=TablaDescripciones:;ID_REACDO12"
SqlQ.CMD Cabecera_EXACDA02_grq.Rst,"ID_EXACDA01"

' Este detalle toma el header por default (recordset en el CMD anterior)
‘y además actualiza un rela al primer recordset de la transacción
SqlQ.CMD Detalle_EXACDA03_grq.Rst, _
   "RelaField=RELA_REACDO12/TablaDescripciones : ; RELA_EXACDA01"

Txn.Apply GrialCont
```

Parte 9. Actualización de Campos en un registro, Método UpdateField

Para estandarizar la asignación de valores desde la pantalla a los campos en un registro, se utiliza el método UpdateField (del Componente GrialUtilP)

Formato:

Sub UpdateField (Dest As Object, Source, [DebeIndicar_CheckMsg As String])

El parámetro “Dest” es un objeto tipo Field (dentro de un Recordset).
El parámetro “Source” es el dato a colocar en el objeto Field
El parámetro Opcional “DebeIndicar_CheckMsg” permite controlar que el valor no sea nulo.

Ejemplo:

```
With PreForm20_grq.Rst
    UpdateField !PREFORM23_DECRI, DescriTxt, "la descripción"
    UpdateField !RELA_PREFORM23, ID_PreForm23_Scb.CurrentValue, "la unidad de medida"
```

El parámetro Opcional "DebeIndicar_CheckMsg" facilita la validación automática para evitar que el campo quede por null o vacío, permitiendo realizar la validación y la actualización en un solo paso.

Si se especifica El parámetro Opcional "DebeIndicar_CheckMsg", se controlará que el valor a actualizar no sea nulo y en caso contrario se generará un error.

En el ejemplo dado, en caso que el TextBox "DescriTxt" este vacío, se generará un error (Run-Time Error 5 de Visual Basic) con el mensaje *"Error: Debe indicar la descripción"*.

Nota: El Método UpdateField no realiza modificaciones en el registro si el valor a actualizar es igual al valor existente en el campo, optimizando así las transacciones al no incluir registros sin modificaciones reales. Debe utilizarse siempre este método en lugar de una asignación directa de un valor al registro.

Se puede modificar el literal "Debe indicar" que se agrega automáticamente el mensaje de error. Para esto debe modificarse la propiedad "DebeIndicarMsg" (Propiedad Pública, as String, del componente GrialUtil). También puede modificarse la propiedad "ValidarTitle" para especificar el título en el mensaje de error (por default es "Validación")

Capítulo V. Grial Combo

Controles con sufijo: _Scb

Parte 1. Descripción General

El GrialCombo tiene tres formas principales de utilización:

1. Como selector de registro para la transacción, en formato combo desplegable.
2. Como selector de registro para un campo RELA, dentro de la transacción.
3. Como Grilla para la edición de Datos, en formato **DisplayFullGrid**

1.1. Selector de Registro principal de la Transacción

Como selector de registro, el GrialCombo representa *visualmente* la elección de un registro dentro de una vista. Se utiliza en la zona de selección de registro (Sobre la TopBar de un GrialCont), para permitir elegir el registro sobre el que va a ser realizada la transacción. También se utiliza para elegir un registro relacionado con un campo RELA dentro de la transacción.

El Combo identifica al registro mediante el valor del campo ID. La propiedad CurrentValue del objeto combo contendrá el ID del registro elegido o null en caso de no existir ninguna selección.

Los controles tipo Combo se nominan con le nombre de la Tabla Troncal representada + "_Scb"; por ejemplo: PreForm08_scb

Visualmente, dentro de un GrialCombo, según su configuración registrada, puede encontrarse una lista de valores (representación directa del recordset de datos) o un árbol (representación de datos mediante un GrialTree interno).

En ambos casos, la propiedad Rst del control, retornará un recordset con los datos del registro actualmente seleccionado por el usuario.

1.1.1 Descripción de la operatoria

Para realizar una selección de registro para una transacción, se coloca un GrialCombo sobre la zona de la TopBar de un GrialCont, a la izquierda de los botones de Alta, Baja y Modificación.

El combo representará los registros sobre los cuales opera la transacción, realizando altas, bajas o modificaciones.

En la inicialización del módulo (UserDocument_Show), se cargan los datos del combo…

```
GrialCont.LoadData  PreForm08_Scb
```

En el momento de inicio de la transacción (ButtonClick), se utiliza para obtener el ID del registro seleccionado por el usuario…

```
Private Sub GrialCont_ButtonClick(ByVal GrialButtonCode As GrialApp.GrialButtons, Cancel As Integer,
NewState As GrialApp.Operation_States)
On Error GoTo ErrH

    Select Case GrialButtonCode
      Case BUTTON_NEW
           PreForm08_Scb.PrepareAddnew
           IDPreform08 = -1
           PrepararTansaccion

      Case BUTTON_MODIFICATION, BUTTON_DELETE
          ' Si no se ha elegido ningún registro, se cancela
          If NoDataIn(ID_PreForm08_Scb.CurrentValue ) Then
                MsgBox "Debe elegir una partida", vbcritical
                Cancel= True
                Exit Sub
          End if
          IDPreform08 = ID_PreForm08_Scb.CurrentValue
          PrepararTansaccion
    . . .
```

Luego, al actualizar los datos (BottomButtonClick, BUTTON_ACCEPT), se recarga el registro modificado para reflejar los últimos cambios …

```
Private Sub GrialCont_BottomButtonClick(ByVal GrialButtonCode As GrialApp.GrialButtons, Cancel As
Integer, NewState As GrialApp.Operation_States)

On Error GoTo ErrH
If GrialButtonCode = BUTTON_ACCEPT Then

    Dim Txn As New GrialTransaction
    Select case GrialCont.LastButtton
    Case BUTTON_MODIFICATION, BUTTON_NEW

      ' Actualizo el recordset troncal
     GrialCont.ChangeMainRst _
          PreForm08_grq.Rst, _ PreForm08GridView_grq.Rst, "ID_PREFORM08"

      ' Preparo la transacción
      Txn.Cmd PreForm08_grq.Rst
      ' Aplico la transacción en el Servidor
      GrialCont.Apply Txn

      If GrialCont.LastButton = BUTTON_NEW then ' Era un Alta
          IDPreForm08 = SqlArea.NewID ' recupero el nuevo ID
      End if
      ' Refresco los datos del combo
      PreForm08_Scb.LoadOnCurrentRecord IDPreForm08

    Case BUTTON_DELETE
      PreForm08_grq.Rst.Delete ' Elimino el registro en el troncal
      Txn.Cmd PreForm08_grq.Rst ' Preparo la transacción
      GrialCont.Apply Txn 'Aplico la transacción en la BD (delete)
      PreForm08_Scb.DeleteRecord 'Elimino el registro del combo. . .

ErrH:
    MsgError GrialCont
    If Txn.Commited then Resume Next 'Continuo si ya esta aplicada la txn
    Cancel = True
End Sub
```

1.2. Uso del GrialCombo como selección de valor para un campo RELA

En este caso, el Combo se utiliza como una parte de los datos de la transacción (dentro del Container_Frm).

La secuencia normal de uso es la siguiente:

Ejemplo:
En el comienzo del módulo, se cargan los datos del control

```
GrialCont.LoadData PreForm23_Scb
```

(Nota: Si el combo muestra siempre un conjunto de datos diferente, dependiendo del registro elegido para cada transacción, debe cargarse cada vez que se inicia una transacción, en PrepararTransaccion)...

En el momento de preparación de la transacción, una vez recuperados los datos principales, posicionamos el combo según el valor del campo RELA asociado...

```
Private Sub PonerEnPantalla()
 ' Posiciono el combo según el valor del campo RELA
With PreForm20_grq.Rst
      PreForm23_Scb.CurrentValue =.Rst!RELA_PREFORM23.Value
```

Al actualizar los datos (BottomButtonClick, BUTTON_ACCEPT), tomamos el valor para el campo RELA desde el Combo...

```
Private Sub TomarDatosDePantalla
    'Actualizo el recordset troncal con los datos
    With PreForm20_grq.Rst
      UpdateField !RELA_PREFORM23, PreForm23_Scb.CurrentValue, "la unidad de medida"
    ...
```

1.3. Uso del GrialCombo como Grilla (DisplayMode = CDM_DisplayFullGrid)

En este caso, el Combo se para mostrar y opcionalmente editar varios registros de detalle dentro de una transacción (dentro del Container_Frm).
Para utilizar un Combo en modo Grilla debe establecerse la propiedad **DisplayMode** con el valor: **CDM_DisplayFullGrid**

La secuencia normal de uso es la siguiente:

Al preparar la transacción, se cargan los datos del detalle:

Ejemplo:

```
Private Sub PrepararTransaccion
' Preparo el registro Troncal
PreForm20Grq.Init _
      "Select * From Pre_Form_20 " & _
      " Where ID_PREFORM20 = " & IDPreForm20

' Preparo el troncal de los detalles
PreForm21Grq.Init _
      "Select * From Pre_Form_21 " & _
      " Where RELA_PREFORM20 = " & IDPreForm20

' Pongo el Filtro para traer los detalles de la cabecera seleccioanda
PreForm21_scb.ExtraFilterCondition = "RELA_PREFORM20 = " & IDPreForm20

' Recupero los datos
GrialCont.LodaData PreForm20grq, PreForm21Grq, PreForm21_Scb
```

Al actualizar los datos (BottomButtonClick, BUTTON_ACCEPT),

tomamos los datos desde el Combo en formato Grilla…

```
Private Sub TomarDatosDePantalla

    'Actualizo el recordset troncal con los datos
    UdpateField PreForm20Grq.Rst!PREFORM20_DESCRI,DescriTxt
    …
    'Actualizo los detalles desde el Combo-Grilla
    PreForm21_scb.Grid_UpdateUserInput
    GrialCont.ChangeMainRst PreForm21_scb.Rst, PreForm21Grq.Rst, "ID_PREEJEC21"

End Sub
```

Para permitir al Usuario modificar, agregar o eliminar registros de un Combo–Grilla, debe establecer las propiedades: **GridAllowEdit, GridAllowAddNew y GridAllowDelete**

1.4. Otras Propiedades del Combo en formato Grilla

Para acceder a los datos de una columna de la fila actualmente seleccionada:
GridValue(FieldName As String) as Variant

Para verificar si el usuario ha seleccionad una fila de datos:
Function Grid_DataRowSelected () as Boolean

Para bloquear la edición de una columna en particular:
ColumnLocked(FieldName As String) As Boolean

Para hacer visible / invisible una columna en particular:
ColumnVisible(FieldName As String) As Boolean

Para establecer manualmente el ancho de una columna:
ColumnWidth(FieldName As String) As integer

Para responder cuando el usuario haga doble-click en la grilla:
Event DblClickFullGrid()

Para responder cuando el usuario se mueva dentro de la grilla:
Event GridRowColChange(LastRow As Long, LastCol As Integer)

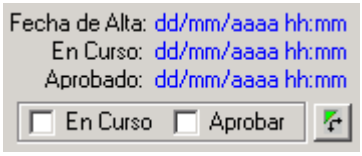
Para verificar los datos al momento que el usuario modifica los datos de una fila:
Event BeforeUpdate(Cancel As Boolean)

Para controlar antes que el usuario elimine una fila:
Event BeforeDelete(Cancel As Boolean)

Para establecer los títulos en caso que el usuario decida imprimir la grilla:
Propiedades ReportTitle, ReportSubTitle y ReportSecondaryTitle

Capítulo VI. Grial WorkFlow

El control Grial WorkFlow facilita el manejo de los comprobantes que participan del sistema de workflow. El control absorbe el manejo de los campos de fecha de alta, de puesta en curso y aprobación del registro troncal cabecera de la transacción. Adicionalmente permite una consulta para ver el histórico de aprobaciones del documento.



Parte 1. Uso del Grial WorkFlow

1.1. Al preparar la Transacción

Al momento de preparar la transacción, una vez que se haya recuperado el registro cabecera o que se haya abierto un espacio en memoria (con `AddNew`), se debe invocar al método **GrialWorkflow.AssignFields**, el cual:

1. Toma los datos del registro y los muestra en pantalla
2. Si el documento esta "En Curso", verifica si el usuario es el próximo en la lista de autorizaciones y habilita la casilla para aprobar
3. Establece las propiedades: `EstabaEnCurso, EstabaAprobado,` de acuerdo a los datos leídos de la base.

Ejemplo:

```
WorkFlow.AssignFields GrialCont, "PRE_EJEC_01", PreEjec01_grq.Rst _
                      , General_GFF.ID_SocUsua02
```

1.2. Al Aceptar la Transacción

Al momento que el usuario acepta las modificaciones se debe invocar al método: **UpdateFields** el cual, de acuerdo a si el usuario aprueba, pone o saca de curso el documento, establece los valores para los campos de la fecha en curso (XXX_WFFECHACURSO) y fecha de Aprobado (XXX_WFFECHA) en el registro troncal.

Ejemplo:

```
WorkFlow.UpdateFields PreEjec01_grq.Rst
```

1.3. Durante la Transacción

En cualquier momento se pueden consultar las propiedades:

EstabaEnCurso (Boolean): cómo se econtraba al momento de iniciar la transaccion
EstabaAprobado (Boolean): cómo se econtraba al momento de iniciar la transaccion

EnCurso (Boolean): si el usuario ha marcado la casilla "En Curso"
Aprobado (Boolean): si el usuario ha marcado la casilla "Aprobado"

FechaAltaText, FechaCursoText, FechaAprobadoText (String):
Tal cual se muestran en pantalla, son útiles para imprimir el docuemnto.

Nota: Para los documentos aprobados, ambas propiedades, **EstabaEnCurso** y **EstabaAprobado** devuelven True, es decir, un documento Aprobado se considera En Curso y Aprobado.

Capítulo VII.
Grial Tree Controller

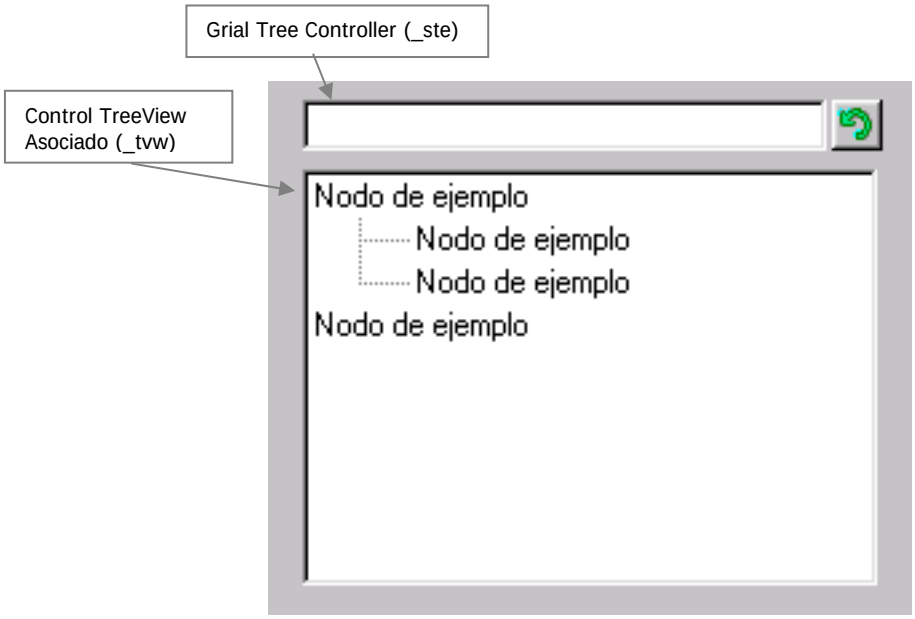
Controles con sufijo: **_ste**

El **GrialTree Controller** maneja la representación *visual* de una tabla tipo árbol (ARB). Generalmente se utiliza para seleccionar un nodo o registro de la tabla. El control requiere de la existencia de un objeto TreeView asociado (Parte de: Windows Common Controls), dentro del cual se muestran y operan los nodos representando los registros de la tabla tipo ARB.

La propiedad **CurrentValue** del objeto GrialTree contendrá el ID del registro elegido o null en caso de no existir ninguna selección. Para acceder a los datos del nodo actualmente elegido se utiliza la propiedad "SelectedNodeRst". Para acceder a los datos de un nodo específico, se utiliza el método "NodeRecord".

Como norma, para elegir el nombre de un control registrado del tipo GrialTree se debe usar el nombre de la tabla o vista cuyos datos se muestran, seguido del sufijo _Ste. Por ejemplo PreForm07_ste, implicara un control que muestra los registros en la tabla PRE_FORM_07.

Para nominar el TreeViewAsociado, se utilizará el mismo nombre que el Grial Tree Controller pero con el prefijo _Tvw (en nuestro ejemplo: PreForm07_Tvw).



Para mayor información ver el Manual de Referencia del GrialTree

Capítulo VIII. Grial Grid Controller y Grial Grid Print

Parte 1. Grial Grid Controller

Controles con sufijo: _sgr

Nota: Debe utilizarse preferentemente un objeto Combo con DisplayMode Grilla (DisplayMode=CDM_DisplayFullGrid) para mostrar una grilla de datos o un reporte. El Grial Grid

Controller se mantiene por compatibilidad y debe utilizarse únicamente cuando es imprescindible acceder a eventos o propiedades muy específicos de la Grilla Janus.

1.1. Descripción General

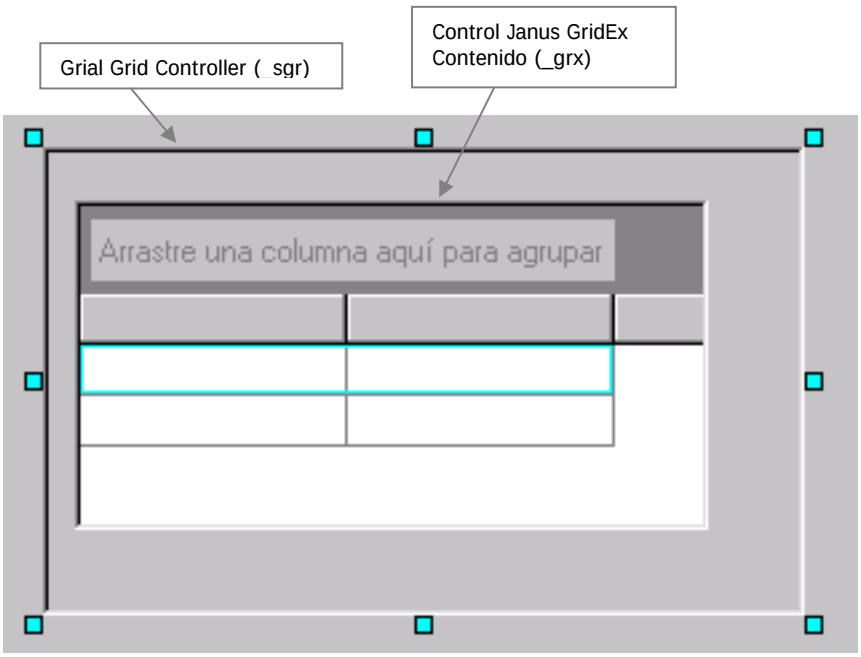
El Grial **Grid Controller** maneja la representación *visual* de una tabla tipo CAB o DET.

El control requiere de la existencia de un objeto **GridEx** colocado dentro del mismo (Parte de: Janus GridEx Controls), dentro del cual se muestran y operan los registros de la tabla.

Mediante la propiedad **ColumnValue** del objeto **GrialGridController** se podrá acceder a todos los campos del registro actualmente seleccionado en la grilla.

Al generar un control registrado de tipo GridController se utilizará el nombre de la tabla o vista cuyos datos se muestran, seguido del sufijo _sgr. Por ejemplo EjeRecu11_sgr, será un control tipo grilla que muestra los registros en la tabla EJE_RECU_11.

Para nominar el control Janus Gridex interno, se utilizará el mismo nombre que el controller pero con el prefijo _grx (en nuestro ejemplo: EjeRecu11_grx).



1.2. Notas especiales para el Grid Controller

1.2.1 Para permitir que el usuario ordene por alguna columna

Debe invocarse al método ColumnHeaderClick del Grid Controller desde el método con similar nombre del objeto Janus Gridex contenido, por ejemplo:

```
Private Sub PreEjec34_grx_ColumnHeaderClick (ByVal Column As GridEX16.JSColumn)
   PreEjec34_sgr.ColumnHeaderClick PreEjec34_grx, Column
End Sub
```

Al invocar este método, cuando el usuario clickee sobre la columna de la grilla, ésta se ordenará por la columna seleccionada.

1.2.2 Control de decimales según la Configuración Regional

Existe un bug en el control de terceros Janus GridEx. Este bug se presenta cuando el usuario tiene configurado el windows para la numeración latina (punto para miles, coma para decimales) en lugar de la numeración sajona (coma para miles, punto para decimales).

Para solucionar este bug de la Janus GridEx, *si se utiliza la grilla para el ingreso por parte del usuario montos o números con decimales*, deberán invocarse los métodos de corrección en DOS eventos de la grilla.

Ejemplo:

```
Private Sub PreEjec21w31_grx_BeforeColUpdate _
    (ByVal Row As Long, ByVal ColIndex As Integer _
    , ByVal OldValue As String, ByVal Cancel As GridEX16.JSRetBoolean)
   PreEjec21w31_Sgr.FixBugJanus_BeforeColUpdate ColIndex
End Sub

Private Sub PreEjec31_grx_BeforeUpdate(ByVal Cancel As GridEX16.JSRetBoolean)
   PreEjec31_Sgr.FixBugJanus_BeforeUpdate
End Sub
```

1.2.3 Método UpdateUserInput

En caso que se permita la edición de la grilla (Propiedad **AllowEdit** del control **JanusGridEx**), al momento de aceptar la transacción es necesario primero cerrar el modo de edición de la grilla, transfiriendo los datos al recordset asociado.

Al invocar al método **UpdateUserInput**, se intentará grabar el dato actualmente en edición en el campo correspondiente del recordset. En caso de falla, (por ejemplo si el campo es numérico y el usuario ingreso texto) se producirá un error de VB (Err.Raise), por lo que se recomienda verificar que exista un control de errores adecuado.

Debe invocarse al método **UpdateUserInput** cuando el usuario acepta la transacción (caso BUTTON_ACCEPT del evento GrialCont_BottomButtonClick).

Al intentar la actualización, se ejecutarán los eventos "**AfterColUpdate**" y luego el evento "**AfterUpdate**", ambos del Control Janus GridEx, si se ha codificado una validación en dichos eventos ésta se ejecutará en este momento, provocándose un error en caso de que los campos no pasen dicha validación.

Parte 2.
Grial Grid Print

Controles con sufijo: _ggp

Nota: Este control es necesario solamente si esta usando un control Grial Grid Controller. **Se recomienda utilizar un control Grial Combo** con **DisplayMode=CDM_DisplayFullGrid**, que maneja automáticamente un control **Grial Grid Print** interno para permitir al usuario imprimir y exportar**.**

2.1. Descripción General

El Control **Grial Grid Print** agrega las funciones de impresión, exportación, ajuste de columnas y cambio de fuente a un control de tipo Janus GridEx. Se recomienda agregar un GrialGridPrint siempre que se utilice una grilla y que sea útil para el usuario imprimir o exportar los datos presentados.

2.2. Uso del Grial Grid Print

2.2.1 Método Prepare_FromGridexObject(aGridex As Object)

Mediante este método se le indica cuál es la grilla (Control Janus GridEx) asociado al GrialGridPrint. Debe invocarse este método luego de haber cargado la grilla con datos.

Ejemplo:
PE11_ggp. Prepare_FromGridexObject (PreEjec11_Grx)

2.2.2 Método SetOperationColumns(ByVal OperationColumnsList As String)

Establece las columnas que serán totalizadas o contadas al momento de realizar la impresión.
El parámetro "OperationColumnsList" es un string con el formato:

[Operation] [DataField] , [Operation] [DataField], ...

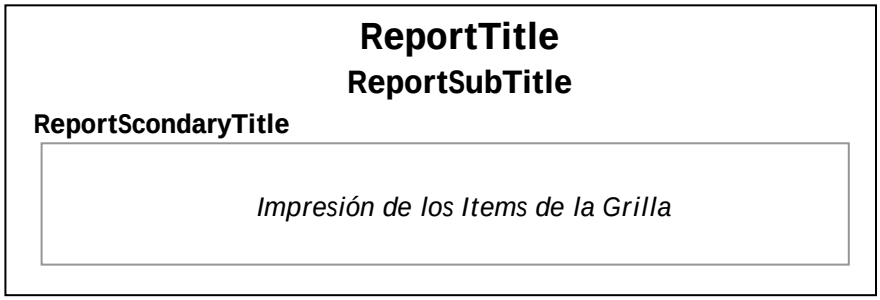
Donde **[Operation]** es: "Sum", "Count" o "Avg"
y **[DataField]** es nombre de un campo del recordset asociado a la grilla

Ejemplo:
PE11_ggp.SetOperationColumns "Count PREEJEC11_DESCRI, Sum PREEJEC11_CANTIDAD"

Al imprimir el contenido de la grilla, se imprimirá el total de registros debajo del campo "PREEJEC11_DESCRI" y el total de cantidad debajo del campo "PREEJEC11_CANTIDAD".

2.2.3 Propiedades ReportTitle, ReportSubTitle y ReportSecondaryTitle as String

Estas propiedades deben cargarse para identificar los datos al momento de la impresión.

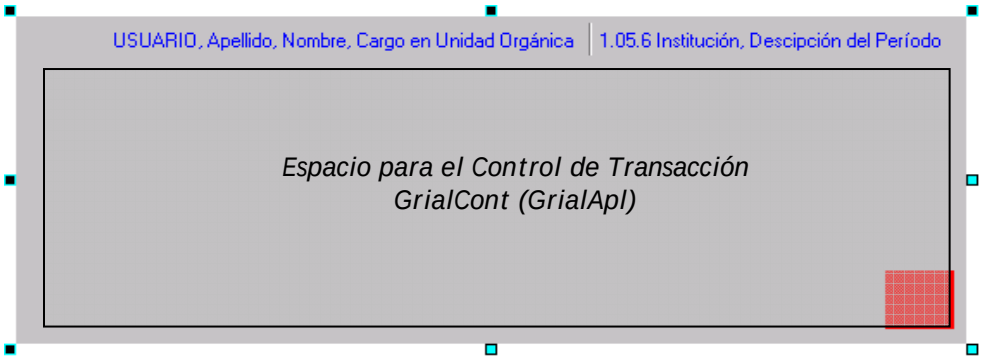


Capítulo IX. Grial Function Frame

Control con sufijo: _GFF

Parte 1. Descripción general

El Control Grial Function Frame simplifica la selección de institución y período de trabajo.



Se utiliza como contenedor general de todo el módulo y le permite al usuario la selección de una institución y período para operar.

Cada vez que el usuario cambia de período o institución, se dispara el evento **SelectionChanged**, para que el módulo pueda cargar los datos que dependan del nuevo período e institución seleccionados.

Es posible establecer permisos para cambiar de períodos e institución dependiendo de cada módulo. Por ejemplo, se puede especificar que un módulo determinado trabaje sólo con instituciones rectoras, o que sólo trabaje con el período en ejecución. Esta configuración se realiza en el módulo "Administrador de Fuentes".

Adicionalmente el control se deshabilita automáticamente mientras el usuario esta en una transacción, no permitiendo el cambio de institución/período hasta se acepte o cancele la transacción en curso.

Por norma, se le asigna el nombre "General_GFF" al control GrialFunctionFrame de cada módulo.

Parte 2. Propiedades

El control General_GFF posee propiedades que informan de todos los datos correspondientes al usuario que esta ejecutando esta aplicación, y de la institución y el período seleccionados.

Es útil para filtrar documentos o datos según la combinación de institución/período seleccionados (ID_PREFORM03) y para obtener e imprimir los datos del usuario en curso.

2.1. Propiedades en relación al Usuario

ID_SocUsua01 As Long
SocUsua01_Nombre As String
SocUsua01_Entidad As String
ID del Usuario, Nombre de Usuario y Nombre de la Entidad

ID_SocUsua02 As Long
ID del puesto activo del Usuario (Cargo + Unidad Orgánica)

ID_PreForm11 As Long
PreForm11_Codigo As String
PreForm11_Descri As String
PreForm11_CodigoyDescri As String
ID, Código y Descripción de la Unidad Orgánica de la identidad activa.

ID_PreForm12 As Long
PreForm12_Codigo As String
PreForm12_Descri As String
PreForm12_CodigoyDescri As String
ID, Código y Descripción del Cargo de la identidad activa.

2.2. Propiedades en relación a la Institución / Perídodo Seleccionados

ID_PreForm03 As Long
PreForm02_Rst As Recordset
ID y Recordset completo de la PRE_FORM_03, que es la combinación de institución+período

2.3. Propiedades en relación a la Institución seleccionada

ID_PreForm02 As Long
PreForm02_Rst As Recordset
ID de la institución seleccionada y registro completo.

EsRectora As Boolean
Si la institución seleccionada es rectora.

PreForm02_Codigo As String
PreForm02_Descri As String
PreForm02_CodigoyDescri As String
Código y Descripción de la institución seleccionada.

2.4. Propiedades en relación al período seleccionado

ID_PreForm01 As Long
ID del Período seleccionado

PreForm01_Descri As String
Descripción del período seleccionado.

PreForm01_Anio As Long
Año del período seleccionado.

PreForm01_Anio_de (RelaPreForm03 As Long) As Long
Año, dado un período+institución particular.

SysForm02_Cod As String
Código correspondiente al tipo de período seleccionado (HIS, FOR, EJE, FUT).

PreForm01_Rst As Recordset
Recordset completo del Período seleccionado

Parte 3. Métodos y Eventos

3.1. Método FirstSelection

Debe invocarse durante el **UserDocument_Show** y su función es validar que el usuario se halle posicionado en una institución permitida para este módulo y luego disparar por primera vez el evento **SelectionChanged**. Siendo que dentro del evento **SelectionChanged** es donde se cargan los datos iniciales, es necesario que se dispare una vez al inicio del módulo para la carga inicial de datos.

Ejemplo:

```
Private Sub UserDocument_Show ( ….

    General_GFF.FirstSelection
```

3.2. Método Set_ID_PreForm01_02

Sub Set_ID_PreForm01_02(aID_PreForm01 As Long, aID_PreForm02 As Long)
Permite seleccionar desde el código en que institución período deseamos posicionar el módulo. Es utilizado cuando se recibe como parámetro un ID de documento y se desea visualizar dicho documento.

3.3. Evento SelectionChanged

Event SelectionChanged(Cancel As Boolean)
Nos permite actualizar y cargar los datos correspondientes a la nueva institución / período.

Por Ejemplo:

```
Private Sub General_GFF_SelectionChanged(Cancel As Boolean)
On Error GoTo ErrH

'Definicion Fitro Origen Pago
ID_EjeRend01_Sel_scb.ExtraFilterCondition = _
```

“RELA_PREFORM02=" & General_GFF.ID_PreForm02

...

Sección 3.
Consideraciones Generales de Programación

Capítulo X. Control de Errores y Validación

Parte 1. Control de Errores

Debe agregarse un control de Errores (On Error Goto ErrH…) en todos los Procedimientos del tipo EVENTO, es decir, Subrutinas (Sub) que sean disparados por acciones del usuario. No es necesario para las subrutinas y funciones específicas definidas en el módulo y que no responden a ningún evento, siendo por lo tanto invocables sólo desde otro punto del código.

Se debe colocar control de errores en todos los procedimientos del tipo:

xxx_Click
xxx_DoubleClick
xxx_KeyPress
xxx_MouseMove

y también:

UserDocument_Show
UserDocument_Resize (Si se coloca código extra al standard)

GrialCont_ButtonClick (GrialApl)
GrialCont_ContainerFrmShow (GrialApl)
GrialCont_BottomButtonClick (GrialApl)
GrialCont_ContainerFrmHide (GrialApl)

CurrentValueChanged (GrialCombo)

Ejemplo de un Control de errores típico:

```
Sub UnBotonCmd_Click
On Error goto ErrH
' Código del procedimiento…
 …
Exit Sub
 ' Al Final del Sub…
ErrH:
    MsgError GrialCont
End Sub
```

Parte 2. Validaciones

En lo posible todas las validaciones de datos deben hacerse en conjunto, dentro de la rutina para tomar datos de la pantalla que es invocada una vez aceptada la transacción.

No se deben realizar validaciones en eventos como: GotFocus o LostFocus o RowColChange, ya que si el usuario elige cancelar la transacción, estos eventos se dispararán de todas formas, tal vez impidiéndole CANCELAR la transacción.

Las validaciones de datos se realizan en conjunto, una vez aceptada la transacción. No se deben realizar validaciones en eventos como LostFocus, GotFocus o RowColChange.

Capítulo XI. Colores y Tamaño de las pantallas

Parte 1. Selección de Colores

En todos los casos debe utilizarse la paleta de colores por tipo del sistema.(ButtonFace, ButtonText, WindowText, etc.)

En caso de mostrar un objeto tipo label con información referente a la transacción, para diferenciarlo de un label de uso común (usado como etiqueta de otro control), se elegirá el color azul puro (Código &H00FF0000&) para la propiedad ForeColor.

Parte 2. Tamaño de la Pantalla

Establecer el tamaño del "UserDocument" en:

Width: 12.000
Height: 8.100

Este tamaño es el máximo para visualizar en modo pantalla completa para una resolución de 800x600 en Windows XP. Si se requiere mayor espacio, se recomienda extenderse hacia la derecha (en el Width), en este caso, poner como Height 7.700, para tomar en cuenta la barra de desplazamiento horizontal que agregará Windows automáticamente.

Capítulo XII. Como poner en producción un módulo.

Parte 1. Poner en producción modificaciones a un módulo existente

Deben verificarse los siguientes puntos antes de poner en producción cualquier módulo.

Paso 1: Verificar el código.

- Asegurarse que se hayan seguido todas las recomendaciones de programación previamente enumeradas en este documento (Control de Errores, Acceso a los objetos Field, etc.)
- Asegurarse que exista la línea "Option Explicit" al principio de cada módulo. (se recomienda marcar la casilla "Requerir declaración de Variables" en el menú Herramientas"/"Opciones"de Visual Basic).
- Verificar el modo de Compatibilidad Binaria:

Acceder en VB al Menú "Proyecto" / Propiedades de … / Tab "Componente".

Si el proyecto NO SE ENCUENTRA *en modo de compatibilidad binaria* primero *agregue la cláusula* **Private** *a todos los Subs y Functions del módulo,* luego coloque el proyecto en modo **Compatibilidad Binaria** y establezca como **DLL de compatibilidad** (caja de texto en la parte inferior del frame), *la DLL que se encuentra en el directorio "Package/Support".*

Al no indicar la cláusula Private, los Sub y Functions de un User Document son **Public**, esto implica que pueden ser llamados desde el exterior y que poseen un identificador único de método, por lo que son tomados en cuenta para mantener la compatibilidad binaria.

Una vez que un módulo esta en modo compatibilidad binaria, no se puede modificar ni el nombre ni los parámetros de cualquier procedimiento que no sea **Private.**

Debe usar la cláusula **Private** para cualquier nuevo procedimiento que se agregue al módulo.

Paso 2: Verificar Propiedades del Proyecto.

Acceder al Menú "Proyecto" / Propiedades de … y Verificar:

- Tab "Generar", Que se encuentre marcada la casilla de "Incremento Automático", Verificar el Título de la Aplicación.
- Tab "Compilar", Analizar la conveniencia de compilar a P-Code (Ver discusión de P-Code más adelante en este documento).
- Tab "Componente", Debe estar en modo "compatibilidad binaria". (excepto para nuevos proyectos, ver Nuevos Proyectos en este mismo documento)
- Tab "Depurar", desmarcar la casilla "usar Explorador existente".

Paso 3: Compilar el Ejecutable (DLL)

Una vez verificado el proyecto, regenerar el ejecutable. Guarde el código fuente del proyecto: Menú Archivo / **Guardar Proyecto.** Elija luego la opción del menú "Archivo / Generar xxxx.DLL". Deberá generar la DLL *en el mismo directorio del proyecto*.

Paso 4: Transferir el package a "Producción"

Una vez generado el DLL, acceda al módulo de Grial "Fuentes Locales", ingrese en la asignación utilizada para modificar este módulo y seleccione el botón de status "Producción" y luego haga clic en "Ejecutar cambio de Status"

El sistema le pedirá confirmación para acceder al utilitario de Visual Basic "Package & Deployment Wizard", el cual nos servirá para crear el package (archivo .CAB) para poder poner en producción el módulo. Ver Cómo usar el "Package&Deployment Wizard"

Luego de generar correctamente el Package, aparecerá nuevamente el módulo Grial de Fuentes Locales. Confirme el pase a producción. El sistema controlará que el package se halla generado correctamente, y de ser así, enviará los archivos al servidor Web de producción.

Paso 5: Opcional: Finalizar la asignación

Si se considera que ha finalizado con los cambios en el contexto de la asignación, debe hacerse un Check-In de los fuentes del módulo.

Vuelva al módulo Grial Fuentes Locales, a la asignación utilizada para crear este nuevo módulo, y seleccione "Check-In" en la barra de Status. Luego pulse "Ejecutar Cambio de Status"

El sistema le pedirá confirmación para generar un archivo .ZIP conteniendo los fuentes, y luego confirmación para transmitir los fuentes comprimidos al servidor de control de versiones, cerrándose la asignación y dejando disponibles los fuentes para otros usuarios.

Parte 2. Cómo Poner en producción un Módulo Nuevo

Paso 1: Desarrollo Inicial

Al inicio de un nuevo proyecto, y mientras se trabaja en modo diseño, no se genera ningún archivo ejecutable para el proyecto (archivo .DLL). Durante este período el modo de compatibilidad estará en "Sin Compatibilidad", y se ejecutará y probará el módulo mediante la tecla F5 de Visual Basic.

Paso 2: Verificar el código.

- Controle que todos los Eventos de tipo **xxx_Clik** tengan un control de Errores (**On Error Goto ErrH**) y que se hayan seguido todas las recomendaciones de programación previamente enumeradas en este documento.
- Controle que TODOS los Sub y Functions del User Document tengan la cláusula **Private**.
- Controle que exista la línea "**Option Explicit**" al principio de cada módulo. (se recomienda marcar la casilla "Requerir declaración de Variables" en el menú Herramientas"/"Opciones"de Visual Basic).

Paso 3: Verificar Propiedades del Proyecto.

Acceda al Menú "Proyecto" / Propiedades de … y controle:

- Tab "Generar", Que se encuentre marcada la casilla de "Incremento Automático", controle el Título de la Aplicación.
- Tab "Compilar", Marcar la casilla compilar a P-Code (Ver discusión de P-Code más adelante en este documento).
- Tab "Componente", Debe estar en modo "sin compatibilidad", ya que aún no se ha generado ninguna DLL para este proyecto.
- Tab "Depurar", desmarque la casilla "usar Explorador existente".

Paso 4: Generar el primer DLL

Guardar el código fuente del proyecto: Menú Archivo / **Guardar Proyecto.**
Elija luego la opción del menú "Archivo / Generar xxxx.DLL". Deberá generar la DLL *en el mismo directorio del proyecto*.

Paso 5: Primer transferencia a "Producción"

Una vez generada la primer DLL, vuelva al módulo de Grial "Fuentes Locales", ingrese en la asignación utilizada para crear este nuevo módulo y seleccione el botón de status "Producción" y luego haga clic en "Ejecutar cambio de Status"
El sistema le pedirá confirmación para acceder al utilitario de Visual Basic "Package & Deployment Wizard", el cual nos servirá para crear el package (archivo .CAB) para poder poner en producción el nuevo módulo. Ver Cómo usar el "Package&Deployment Wizard"
Luego de generar correctamente el Package, aparecerá nuevamente el módulo Grial de Fuentes Locales. Confirme el pase a producción. El sistema controlará que el package se halla generado correctamente, y de ser así, enviará los archivos al servidor Web de producción.

Paso 6: IMPORTANTE: Colocar el proyecto en modo "Compatibilidad Binaria"

Una vez generado el primer package y puesto en producción, acceder nuevamente al proyecto Visual Basic para *cambiar el proyecto a modo "Compatibilidad Binaria"*. Para esto acceder al Menú "Proyecto" / Propiedades de … / Tab "Componente", especificar **Compatibilidad Binaria** y establecer como **DLL de compatibilidad** (caja de texto en la parte inferior del frame), *la DLL que se encuentra en el directorio "Package/Support".*

IMPORTANTE: Para la compatibilidad BINARIA debe usarse la DLL ubicada en la carpeta …/PACKAGE/SUPPORT.

Una vez cambiado el proyecto a "Compatibilidad Binaria", guardar el proyecto (Menú Archivo / Guardar Proyecto), y *Cerrar el Visual Basic.*

Paso 7: Opcional: Finalizar la asignación

Si se considera que se ha finalizado el módulo y no se realizarán más cambios en el contexto de la asignación, debe hacerse un Check-In de los fuentes del módulo.

Vuelva al módulo Grial Fuentes Locales, a la asignación utilizada para crear este nuevo módulo, y seleccione "Check-In" en la barra de Status. Luego pulse "Ejecutar Cambio de Status"

El sistema le pedirá confirmación para generar un archivo .ZIP conteniendo los fuentes, y luego confirmación para transmitir los fuentes comprimidos al servidor de control de versiones, cerrándose la asignación y dejando disponibles los fuentes para otros usuarios.

Capítulo XIII. Cómo usar el "Package & Deployment Wizard"

Una vez dentro del "Package&Deployment Wizard"…

- Elegir el botón "Empaquetar"
- El P&D.Wizard siempre preguntará si se desea recompilar la aplicación, conteste *NO*. Por lo general aparecerá esta pregunta por que el archivo de proyecto (.vbp) es posterior al último .DLL generado, ya que la opción de auto-incremento de versión modifica el .VBP luego de cada compilación.
- Elija la secuencia de creado de un "Internet Package". Si no existen secuencias, seleccione "(none)", y luego en el tipo de secuencia seleccione "Internet Package" o "Paquete de Internet"
- Seleccione la carpeta del paquete. Es *IMPORTANTE* normalizar el nombre para esta carpeta: debe ser: "Grial Development/Sources/*Proyecto/Sistema/módulo*/**Package".** Dependiendo del lenguaje en que instaló el VB, el wizard propondrá distintos nombres. *La Carpeta para el package debe especificarse en el path:* "Grial Development/Sources/*Proyecto/Sistema/módulo*/**Package".**
- Luego de elegir la carpeta pueden aparecer algunos diálogos con listas de archivos sin información de dependencia, marque todas las casillas y acepte los diálogos o simplemente ignórelos). Llegará al punto de elegir los "Archivos Incluidos". Debe *desmarcar* **todos los archivos** *excepto la DLL del proyecto.* Luego pulse "Siguiente/Next".
- Pantalla de origen del archivo: Si desmarcó correctamente los archivos en la pantalla anterior, deberá aparece aquí sólo el archivo DLL del proyecto. Pulse "Siguiente/Next".
- Pantalla de configuración de seguridad: Coloque "si/yes" en todas las casillas de la grilla. Pulse "Siguiente/Next".
- Fin del recorrido. Pulse "Finalizar/Finish" para generar el package.

Capítulo XIV. Detalles de Programación

Parte 1. Rutinas obsoletas en GrialUtil.DLL vs GrialUtilP.OCX

El componente DLL GrialUtil.DLL (DLL, incorporado a algunos proyectos como referencia) es obsoleto, debe reemplazarse por el componenente GrialUtilP.OCX (OCX, como componente)

Las rutinas en GrialUtilP.OCX han cambiado de nombre para homogeneizar el lenguaje:

Obsoleto: *Rutinas en GrialUtil DLL*	*Nuevo:* *Componente GrialUtilP OCX*
NoNuloStr()	NVLStr()

EstaVacío()	NoDataIn()
SiVacioNulo()	IfEmptyNull()

Parte 2. Compilación a P-Code vs. Código Nativo

Se recomiendo compilar los módulos en P-Code, pero en caso que la velocidad de ejecución LOCAL del módulo es crítica (si el módulo posee procesos locales en el cliente, sean bucles for-next extensos o bucles recorriendo una gran cantidad de registros en un recordset) es posible obtener una mejora en tiempos de ejecución compilando a "código nativo" en lugar de **P-Code**, mediante dicha opción, el código ejecutable resultante posee en promedio **el doble** de tamaño que compilando en P-Code.

La opción del modo de compilación se encuentra en:

Menú "Proyecto" / Propiedades de … / Tab "Compilar"

Parte 3. Diferencias entre el objeto SQLQuery y el objeto GrialTransaction

El Objeto **SqlQuery** es obsoleto y se mantiene por compatibilidad, posee las mismas funciones del objeto **GrialTransaction**, con la única diferencia en el método para aplicar la transacción:

El objeto **GrialTransaction** se aplica con el método **Apply** en tanto que en el `SqlQuery` se utiliza el método "**UpdateRecords**"

Por Ejemplo:

SQLArea.UpdateRecords es equivalente a **Txn.Apply GrialCont**

Ambos comandos envían la transacción al servidor.

Se recomienda en los nuevos módulos utilizar objetos **GrialTransaction** que permiten el uso de conexiones a más de una base de datos simultáneamente.