Creación de un componente de datos (Data-Aware Component)

En este documento vamos a explicar, sin entrar en mucha profundidad, como crear un componente de datos (Data-Aware Component) a partir de un componente existente.

Para profundizar sobre el tema recomiendo el libro (que ya sólo se puede conseguir en Internet): **Developing Custom Delphi 3 Components** de *Ray Konopka.*

Un componente de datos no es más que un componente '*normal*' al que se le han añadido características de acceso a datos. Éste se realiza a través de la clase TDataLink que es la que se encarga de hacer la parte compleja de este acceso. Alguna de las propiedades y métodos de la clase son :

Propiedades	
Active	Indica cuando el dataset del DataLink está activo
ActiveRecord	Índice del registro activo dentro del buffer del Dataset
Bof	Índica si es el primer registro
BufferCount	Número de registros en el buffer del dataset
DataSet	Conjunto de datos
DataSource	Fuente de los datos
DataSourceFixed	Indica si la propiedad DataSource puede cambiar
Editing	Índica si el DataSource esta en modo edición
Eof	Índica si es el último registro
ReadOnly	Índica se los datos pueden ser modificados
RecordCount	N° de registros en el buffer de registro interno. Puede ser menor que BufferCount
VisualControl	Índica si un control visual utiliza el DataLink

Métodos	
ActiveChanged	Responde a un cambio en la propiedad Active
CheckBrowseMode	Permite responder a un cambio de estado en el dataset antes de que se produzca
DataEvent	Responde a los eventos que se produzcan durante el trabajo con los datos
DataSetChanged	Responde a los cambios en el Dataset
DataSetScrolled	Se llama cada vez que nos movemos a un nuevo registro.
EditingChanged	Se produce cada vez que hay un cambio en la propiedad Editing.
DataSourceFixed	Indica si la propiedad DataSource puede cambiar
RecordChanged	Este método se llama cada vez que se produce una modificación en un campo del registro actual
UpdateData	Se llama cada vez que se llama a UpdateRecord.
UpdateRecord	Es llamado por el componente para que el registro actual refleje el contenido del componente sin hacer un post del registro.

Para conectar el componente a los datos no usaremos este tipo sino que lo haremos a través de un descendiente : TFieldDataLink (también tenemos la posibilidad de hacerlo a través de TGridDataLink pero con es obvio esto servirá para el caso de un control que muestre un conjunto de datos a la vez, lo cual no es nuestro caso).

Las propiedades y eventos de esta clase son:

Propiedades	
CanModify	Indica cuando el control puede modificar el campo asociado del conjunto de datos
Control	El componente de datos que maneja el DataLink (debe ponerse a Self)
Editing	Indica cuando el dataset este en modo edición
Field	Campo con el que opera el componente
FieldName	Nombre del campo asociado del conjunto de datos

Eventos	
OnActiveChange	Se produce cuando la propiedad Active del TFieldDataLink cambia
OnDataChange	Se produce cuando cambia el contenido del dataset
OnEditingChange	Se produce cada vez que se produce un cambio en el estado de edición (cuando se entra o se sale del estado de edición)
OnUpdateData	Se produce cuando el datasource necesita actualizar los datos desde el componente para actualizar la base de datos.

Bueno, hasta aquí la teoría ahora pondremos manos a la obra para crear nuestro componente, para ello vamos a partir del componente ImageClick que podéis encontrar en esta página web (http://www.i-griegavcl.com/imageclick.asp). Este en un componente gráfico que responde a la pulsación del ratón cambiando la imagen mostrada (seleccionada de una lista de imágenes en el form).

Para conectar a un conjunto de datos nuestro componente tendremos que añadirle un componente DataLink, en nuestro caso un TFieldDataLink, puesto que sólo mostraremos un registro cada vez.

Veamos el código completo de nuestro componente y después lo explicaremos paso a paso.

```
unit DBImageclick;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  ExtCtrls,imglist,DB,DBTables,DBCtrls,ImageClick;

type
  TDBimageClick=Class(TImageClick)
  private
     FDataLink : TFieldDataLink;
     FReadOnly : Boolean;
     Function  GetDataField:string;
     Procedure SetDataField(const Value:string);
     Function  GetDataSource:TDataSource;
     procedure SetDataSource(Value:TDataSource);
     Procedure UpdateData(Sender:TObject);
     Procedure DataChange(Sender:TObject);
  protected
     Procedure Notification(AComponent:TComponent;Operation:TOperation); override;
     procedure SetSelected(Value:integer); override;
     Procedure Click; override;
  public
     Constructor Create(AOwner : TComponent); override;
     Destructor Destroy; override;
```

```
  published
     Property ReadOnly : Boolean read FReadOnly write FReadOnly default False;
     Property DataField : String read GetDataField write SetDataField;
     Property DataSource : TDataSource read GetDataSource write SetDataSource;
  end;

Implementation

constructor TDBImageClick.Create(AOwner: TComponent);
begin
   inherited Create(AOwner);
   ControlStyle:=controlStyle-[csReplicatable];
   FReadOnly:=False;
   FDataLink:=TFieldDataLink.Create;
   FDataLink.OnDataChange:=DataChange;
   FDataLink.OnUpdateData:=UpdateData;
   //FdataLink.Control:=self; {En nuestro caso no es necesario}
end;

destructor TDBImageClick.destroy;
begin
   FDataLink.Free;
   FDataLink:=nil;
   inherited Destroy;
end;

function TDBImageClick.GetDataField:String;
begin
   Result:=FDataLink.FieldName;
end;

procedure TDBImageClick.SetDataField(const value:string);
begin
   FDataLink.FieldName:=Value;
end;

function TDBImageClick.GetDataSource:TDataSource;
begin
   Result:=FDataLink.DataSource;
end;

Procedure TDBImageClick.SetDataSource(value:tdatasource);
begin
   if FDataLink.DataSource<>Value then
   begin
      FDataLink.DataSource:=Value;
      If Value<>nil then
         Value.FreeNotification(self);
   end;
end;

Procedure TDBImageClick.Notification(AComponent:TComponent;Operation:TOperation);
begin
   inherited Notification(Acomponent,Operation);
   if (Operation=opRemove) AND
      (FDataLink<>nil) and
      (AComponent=DataSource) then
           DataSource:=nil;
end;

Procedure TDBImageClick.DataChange(Sender : TObject);
begin
   if FDataLink.Field=nil then
      Selected:=0
   else
      Selected:=FDataLink.Field.AsInteger;
end;
```

```
Procedure TDBImageClick.UpdateData(Sender : TObject);
begin
   FdataLink.Field.AsInteger:=Selected;
end;

Procedure TDBImageClick.SetSelected(Value:integer);
begin
   if (csDesigning in ComponentState) or (FDataLink=nil)  then
      exit
   else
      begin
         inherited;
         If FReadOnly then
            FDataLink.Modified;
      end;
end;

Procedure TDBImageClick.Click;
begin
   if (csDesigning in ComponentState) or
      FReadOnly or
      (FDataLink=nil) or (not fdatalink.Edit) then
         exit
   else
   begin
      inherited;
      FDataLink.Modified;
   end;
end;
end.
```

Este sería el código de nuestro componente :

- Heredamos de la clase TImageClick que va a servir como base.
- Añadimos un objeto TFieldDataLink (lo asignamos a la variable privada FDataLink).
- Publicamos tres propiedades que son :

  - *`ReadOnly`* : Indicará si el componente puede modificar los datos o no. En nuestro caso va a trabajar de manera distinta a como lo haría normalmente en otro componente de datos (vea la definición del componente TDBEdit para comprobarlo).
  - *`DataField`* : Nombre del campo
  - *`DataSource`* : Nombre de la fuente de datos

- También sobreescribimos una serie de métodos que explicaremos más adelante.

Veamos los códigos de los distintos procedimientos y funciones:

```
constructor TDBImageClick.Create(AOwner: TComponent);
begin
   inherited Create(AOwner);                          (1)
   ControlStyle:=controlStyle-[csReplicatable];       (2)
   FReadOnly:=False;                                  (3)
   FDataLink:=TFieldDataLink.Create;                  (4)
   FDataLink.OnDataChange:=DataChange;                (5)
   FDataLink.OnUpdateData:=UpdateData;                (5)
   //FdataLink.Control:=self; {En nuestro caso no es necesario} (6)
end;
```

(1) Llamamos al método create del antecesor

(2) Nos aseguramos que el estilo del control no se puede replicar (caso de un Grid), para ello 'restamos' csReplicatable al estilo.

(3) Ponemos por defecto a false la propiedad ReadOnly del componente (damos la posibilidad de que el control pueda modificar los datos de la B.D.).

(4) Acto seguido creamos una instancia de la Clase TFieldDataLink que será la que nos controle la parte engorrosa del acceso a los datos.

(5) Le asignamos los procedimientos DataChange y UpdateData a los eventos adecuados que escribiremos más adelante y que nos permitirán ver y modificar los valores de los datos.

(6) En nuestro caso como el componente no tiene la posibilidad de capturar el foco no nos hace falta asignar la propiedad *Control* de DataLink.

Con esto se crea el control DBImageClick que aparecerá en el form.

```
destructor TDBImageClick.destroy;
begin
   FDataLink.Free;
   FDataLink:=nil;
   inherited Destroy;
end;
```

Liberamos el objeto DataLink y asignamos nil a la variable protegida.

Llamamos al procedimiento Destroy del antecesor.

```
function TDBImageClick.GetDataField:String;
begin
   Result:=FDataLink.FieldName;
end;
```

Con esta función leemos el valor de la propiedad DataField de nuesto componente (el campo del datasource al que hace referencia nuestro componente), para lograrlo lo único que hacemos es leer el nombre del campo del DataLink.

```
procedure TDBImageClick.SetDataField(const value:string);
begin
   FDataLink.FieldName:=Value;
end;
```

Este procedimiento hace lo contrario que la función anterior, en este caso asignamos el campo al que queremos que nuestro componente tenga acceso, para ello asignamos el valor de campo al nombre de campo del DataLink.

```
function TDBImageClick.GetDataSource:TDataSource;
begin
   Result:=FDataLink.DataSource;
end;
```

Esta función la utilizaremos para leer el nombre de la propiedad DataSource de nuestro componente. Para ello sólo deberemos ver el valor DataSource del DataLink.

```
Procedure TDBImageClick.SetDataSource(value:tdatasource);
begin
   if FDataLink.DataSource<>Value then
   begin
      FDataLink.DataSource:=Value;
      If Value<>nil then
         Value.FreeNotification(self);
   end;
end;
```

Este es el procedimiento que nos permitirá asignar la propiedad DataSource de nuestro componente. Para ello comprobamos que el DataSource que pretendemos asignar no es el que ya podemos tener asignado, asignamos el valor y comprobamos que este no es nulo para hacer que el Datasource al que hace referencia nos avise antes de que se libere (*Value.FreeNotification(self)*), esto es necesario siempre que un componente hace referencia a otro distinto y que él no controla de manera explícita (es decir, que no lo ha creado, si mira el código de TImageClick, el componente antecesor de este que estamos creando, verá que también utiliza esto para que el componente se entere si se destruye la lista de imágenes). Si no se pusiera esta línea de código lo que ocurriría es que nuestro componente haría referencia a un objeto que no existe con las consecuencias que de ello se derivan (código de error, posible 'cuelgue de delphi', etc).

```
Procedure TDBImageClick.Notification(AComponent:TComponent;Operation:TOperation);
begin
   inherited Notification(Acomponent,Operation);
   if (Operation=opRemove) AND
      (FDataLink<>nil) and
      (AComponent=DataSource) then
           DataSource:=nil;
end;
```

Muy bien, hemos hecho que el DataSource nos avise cuando vaya ha hacer algo que nos pueda afectar (como es el borrado del mismo), pero que tenemos ¿qué hacer entonces nosotros en nuestro componente? Pues deberemos comprobar la operación que de la que se nos avisa y obrar en consecuencia. En nuestro caso deberemos llamar antes al procedimiento Notificación del antecesor, por si este tuviera que tomas también alguna medida y después comprobaremos varias cosas, a saber :

1. Que la operación de la que se nos avisa es la de liberación del objeto (borrado).
2. Que tenemos asignado un DataLink en nuestro componente.
3. Que el componente que nos está avisando es un DataSource.

Si se cumplen todas estas condiciones nosotros sólo debemos romper el enlace a los datos.

```
Procedure TDBImageClick.DataChange(Sender : TObject);
begin
   if FDataLink.Field=nil then
      Selected:=0
   else
      Selected:=FDataLink.Field.AsInteger;
end;
```

Este procedimiento es el que va a responder al evento OnDataChange provocado por el DataLink. Este evento nos avisa de que el registro en ha cambiado y que debemos actualizar nuestro componente (por ejemplo cuando hacemos un scroll en el conjunto de datos seleccionado).

Entonces comprobamos que el DataLink tiene asociado un campo, si no fuera así seleccionamos la imagen 0 (que siempre va a ser la de defecto), en caso contrario asignamos aquella cuyo valor (propiedad selected) sea el del campo del DataLink (sólo se admiten enteros).

```
Procedure TDBImageClick.UpdateData(Sender : TObject);
begin
   FdataLink.Field.AsInteger:=Selected;
end;
```

Este procedimiento responderá al evento OnUpdateData provocado por el DataLink cuando necesite actualizar el DataSet. Y lo que hace es asignar al campo Field del DataLink el valor del campo del componente (el número asociado a la imagen que muestra el componente en ese momento).

```
Procedure TDBImageClick.SetSelected(Value:integer);
begin
   if (csDesigning in ComponentState) or (FDataLink=nil)  then
      exit
   else
      begin
         inherited;
         If FReadOnly then
            FDataLink.Modified;
      end;
end;
```

Este es un procedimiento que ya existía en la clase antecesora y que en esta nueva clase sobreescribimos (override) para dotarlo de nueva funcionalidad. Este procedimiento asignaba un valor a la propiedad *Selected* de TimageClick.

Aquí lo que haremos es comprobar que no estamos en fase de diseño (csDesigning) y que el DataLink está unido a datos, si no fuera así, salimos del procedimiento sin hacer nada., en caso contrario llamamos al procedimiento heredado que se encarga de asignar un nuevo valor y después comprobamos si estamos en modo ReadOnly para avisar al DataLink de que hemos modificado datos o no.

```
Procedure TDBImageClick.Click;
begin
   if (csDesigning in ComponentState) or
      FReadOnly or
      (FDataLink=nil) or (not fdatalink.Edit) then
         exit
   else
   begin
      inherited;
      FDataLink.Modified;
   end;
end;
```

Este es otro procedimiento sobreescrito ya que también existe en la clase antecesora. El la clase antecesora, este procedimiento se encarga de controlar la pulsación del ratón sobre el control (mirar funcionamiento en código fuente).

Para que este procedimiento reaccione a la pulsación del ratón tienen que darse una serie de circunstancias :

1. Que no estemos en modo diseño.
2. Que el componente no sea ReadOnly.

3. Que el DataLink tenga acceso a datos.
4. Que pueda ponerse en modo de edición. Por ejemplo no podría ponerse en modo edición si el DataSet sobre el que se basa el DataLink fuera de solo lectura.

Si se dan todas estas condiciones llamaremos al procedimiento Click del antecesor y avisaremos al DataLink de que hemos hecho una modificación sobre el valor del campo, para que llegado el momento actualice el campo correspondiente del conjunto de datos.

Y aquí termina todo. Hay otro evento que se debe tener en cuenta la mayoría de las veces, el evento OnEditingChange, pero que en este componente no lo he considerado necesario. Para ver su uso recomiendo la siguiente dirección : http://personal.redestb.es/revueltaroche/ccu12.htm .

Espero que este documento os sirva para tener una visión, aunque sea somera, de la creación de componentes con acceso a datos.