

JUGANDO CON EL PORTAPAPELES

Neftalí - Germán Estévez – neftali@mx3.redestb.es

Este artículo intenta ser un breve repaso sobre las posibilidades que Delphi nos brinda de interactuar con el portapapeles de Windows (“Clipboard para los amigos”).

Presentación

La idea de éste artículo es hacer un breve repaso a la clase *TClipboard* que se implementa en Delphi y las operaciones más básicas que podemos utilizar en nuestros programas relacionadas con el portapapeles de Windows. Espero que las explicaciones y los ejemplos aclaren (un poco al menos) el funcionamiento básico sobre éste tema.

Introducción

La clase *TClipboard* nos permite acceder al portapapeles de Windows; Copiar, cortar y pegar objetos y registrar cambios que otras aplicaciones pueden realizar en él.

Delphi implementa las funciones básicas sobre los controles estándar (los más utilizados). No tenemos problemas para copiar textos, imágenes,... al portapapeles desde los controles normales. Los “problemas” nos los podemos encontrar cuando deseamos trabajar con el portapapeles desde controles no considerados como “estándar” o cuando deseamos manejar datos que no sean los más comunes comentados anteriormente.

Funciones estándar sobre el Portapapeles

Para acceder a las funciones estándar del *portapepeles* desde una unit de nuestro programa basta con añadir la Unit *<Clipbrd.pas>* al USES. Además podremos utilizar la variable *portapapeles* que crea Delphi (al añadir la unit al uses ya tendremos acceso a ella), así que no debemos crear nuestro propio objeto.

Soporte de Delphi

Delphi tiene “lo suficiente” para realizar las operaciones estándar, pero la verdad es que cuando queremos realizar algo que se sale de lo más básico, en ésta clase tenemos “poca cancha”.

Las clases *TCustomEdit* (de la cual derivan los controles de edición como *TEdit*, *TMemo*, *TMaskEdit*,...) y *TDBImage* poseen lo métodos **CopyToClipboard**, **CutToClipboard** y **PasteFromClipboard** que permiten manejar las operaciones básicas sobre el portapapeles en formato *CF_TEXT* (texto normal) y *CF_BITMAP* (formato para mapas de bits) respectivamente.

Podemos obtener más información e incluso algunos ejemplos si accedemos a la ayuda de *Delphi* sobre estos métodos.

El resto de clases gráficas como *TBitmap*, *TGraphics*, *TIcon*, *TJPEGImage*, *TMetafile*, *TOLEGraphic* y *TPicture* poseen los métodos **LoadFromClipboardFormat** y **SaveToClipboardFormat** para tratar las imágenes en los diferentes formatos.

```
var
  MyFormat : Word;
  AData: Cardinal;
  APalette : HPalette;
begin
  Self.Imagel.Picture.Bitmap.SaveToClipboardFormat(MyFormat, AData, APalette);
  Clipboard.SetAsHandle(MyFormat, AData);
```

Utilizando la clase *TClipboard* también podemos copiar Texto y un *PChar*, con lo que podemos copiar el contenido de un *ListBox*, de un *Memo*, o de un *StringGrid* por ejemplo utilizando:

```
Clipboard.AsText := Self.ListBox1.Items.Text;
Clipboard.AsText := Self.Memo1.Text;
```

NOTA: El ejemplo del StringGrid está desarrollado completamente en uno de los ejemplos que se adjuntan con éste artículo.

Operaciones con otros elementos; Utilizar Streams

Hasta ahora hemos visto las operaciones “directas” que nos ofrece Delphi para interactuar con el portapapeles y los objetos en los que podemos utilizarlas, pero ¿qué pasa cuando debo utilizar un objeto que no está entre los que hemos mencionado anteriormente? ¿Qué pasa si yo deseo copiar un icono? ¿O un RTF?

La forma más práctica (a mi entender ;-) claro...) es utilizar *Streams* como paso intermedio para almacenar el objeto a copiar (e igual para pegar).

Para los que os atrevéis a “chafardear” en las VCL, podéis ver que éste es el sistema que usa Borland por ejemplo, para los métodos *GetComponent* y *SetComponent* de la clase *TClipboard* (unit **clipbrd.pas**) o para el método *Assign* de la clase *TTreeNode* (unit **Comctrls.pas**) por comentar alguno.

¿Y qué es eso de los “Streams”?

Bueno, Delphi nos dice que los Streams proveen un método común para escribir/leer datos en diferentes formatos. Existe la clase generica *TStream* (que es abstracta y por lo tanto no deberían crearse instancias/objetos de ella) y unas derivadas de ésta especializadas para determinados tipos de datos. Así existen clases para trabajar con Memoria (*TMemoryStream*), Ficheros (*TFileStream*), Strings (*TStringStream*), campos BLOB de Base de Datos (*TBlobStream*), sockets (*TWinSocketStream*) y objetos OLE (*TOleStream*).

Cada una de éstas clase nos posee métodos para acceder a datos, así *TStringStream* posee

métodos como *ReadString* o *WriteString* o la clase *TBlobStream* implementa un constructor donde se le pasa como parámetro un campo de tipo *TBlobField* a partir del cual obtiene los datos.

Además de esto, otras muchas de las clases de Delphi implementan métodos para interactuar con Streams, así *SaveToStream* y *LoadFromStream* se encuentran en clases como *TBitmap*, *TBlobField*, *TCustomClientDataSet*, *TCustomTreeView*, *TDBGridColumns*, *TGraphic*, *TIcon*, *TJPEGImage*, *TMetafile*, *TStrings*,...

Así, podemos copiar un *Bitmap* a otro (además de utilizando un *Assign*, por supuesto) utilizando el siguiente código:

```
var
  Str: TMemoryStream;
begin
  Str := TMemoryStream.Create();
  Image1.Picture.Bitmap.SaveToStream(Str);
  Str.Position := 0;
  Image2.Picture.Bitmap.LoadFromStream(Str);
  Str.Free;
```

O podemos guardar un *RichEdit* en un campo *Blob* utilizando el siguiente código:

```
var
  bs: TBlobStream;
begin
  bs := TBlobStream.Create( tablememo, bmwrite );
  Richedit1.plaintext := false;
  Richedit1.Lines.SaveToStream(bs);
```

Los pasos básicos para una operación de copiar serían los siguientes:

1) Paso previo: Registrar un formato. Normalmente haremos esto en la rutina de **initialization** llamando a la API *RegisterClipboardFormat*

```
var
  cfIcon: Word;
...
initialization
  cfIcon := RegisterClipboardFormat('Icon Data');
```

2) Rellenar el Stream. Debemos crear una variable de tipo *TMemoryStream* y “rellenarla” con los datos que deseamos copiar.

```
var
  memStream: TMemoryStream;
...
  // Copiar el icono al Stream
  Self.Image1.Picture.Graphic.SaveToStream(memStream);
```

3) Pasar el Stream a memoria. Debemos bloquear y reservar un bloque de memoria global y “volcar” el Stream con los datos en ella.

```
// Reservar memoria
Data := GlobalAlloc(GMEM_MOVEABLE, memStream.Size);
// Bloquear la memoria
DataPtr := GlobalLock(Data);
// Copiar el Stream a memoria
```

```
Move(memStream.Memory^, DataPtr^, memStream.Size);
```

4) Copiar los datos al portapapeles. Llamamos al Método *SetAsHandle* de la clase *TClipboard* con el identificador del formato que hemos registrado y el apuntador a memoria.

```
// Guardar el dato en el portapapeles  
Clipboard.SetAsHandle(cfIcon, Data);
```

Los pasos para recuperar el objeto almacenado son básicamente los mismos, pero en orden inverso.

1) Crear un stream para almacenar los datos. Utilizamos un *memoryStream* igual que para el paso anterior.

```
// Crea un Stream para almacenar los datos  
MemStream := TMemoryStream.Create;
```

2) Obtener los datos del Portapapeles. Utilizando el método *GetAsHandle* de la clase y bloquear la memoria donde se almacena. Podemos comprobar además si ha habido algún error.

```
// obtiene datos del Clipboard  
Data := Clipboard.GetAsHandle(cfIcon);  
// No ha obtenido nada correcto?  
if Data = 0 then  
    Exit;  
End;  
// Bloquea la memoria  
DataPtr := GlobalLock(Data);  
// No ha podido bloquear memoria ?  
if DataPtr = nil then  
    Exit;  
End;
```

3) Pasar los datos al Stream. Escribimos los datos obtenidos utilizando el método *WriteBuffer*.

```
// Escribe los datos en el Stream  
MemStream.WriteBuffer(DataPtr^, GlobalSize(Data));  
Memstream.Seek(0, 0);
```

4) Volcar la información desde el Stream a nuestro objeto. Por último sólo queda recuperar la información desde el Stream.

```
// Actualiza el icono del Form  
Self.Icon.LoadFromStream(Memstream);  
// Actualiza la imagen  
Image2.Picture.Graphic := Icon;
```

NOTA: Ambos pasos, tanto el de copiar como el de pegar están desarrollados completamente en uno de los ejemplos que se adjunta con el artículo.

Hook sobre el portapapeles

Un último detalle sobre el que me gustaría comentar algo acerca del portapapeles es la posibilidad que tenemos de interceptar los mensajes del sistema acerca de los cambios en el

Portapapeles del sistema. La utilización más usual, sería un visualizador del portapapeles, o que en nuestro programa nos interesara saber por ejemplo cuando se “pega” algo en el *clipboard* con un formato compatible para nuestro programa (para activar el “pegar”). Para esto podríamos pensar en un **Timer** que cada cierto tiempo interrogara al portapapeles, aunque esta solución implicaría tiene sus inconvenientes:

- * Una sobrecarga del sistema debido al funcionamiento continuado del timer.
- * Y tampoco asegura su correcto funcionamiento debido al lapso de tiempo entre cada interrogación; Y no vale decir “pues interrogo más a menudo”, porque eso empeora el primer problema.

El sistema correcto de conseguir esto es los capturar mensajes del sistema acerca de los cambios sobre el portapapeles. Para entenderlos mejor podemos ver primero como se organiza el sistema y luego comentarlos brevemente.

Windows almacena una cadena donde se encuentran todos los procesos/ventanas (Handles) que recibirán los mensajes de cambio en el portapapeles (chain of clipboard viewers, a partir de ahora CCB). Para que nuestro programa intercepte éstos mensajes lo único que tenemos que hacer es “añadirlo a ésta cadena”. Cuando se produce un cambio, Windows envía el mensaje al primer elemento de la cadena y cada elemento lo propaga al siguiente objeto que hay en la CCB.

Adicionalmente Windows nos informa cuando uno de los elementos de la cadena se ha eliminado. En ese caso nos interesa saber si ese cambio nos afecta (en concreto si es el siguiente elemento de la cadena después de nosotros).

Revisamos cada uno de los pasos a seguir, con algunos ejemplo de código, para registrar uno de nuestros formularios como visor de portapapeles.

PASO 1: Al crear el formulario registramos nuestro formulario en la cadena de procesos que reciben los mensajes del Portapapeles (CCB), para ello utilizamos la API *SetClipboardViewer*. Como respuesta el sistema nos devuelve el *Handle* del siguiente elemento de la CCB.

```
Self.SiguienteHandle := SetClipboardViewer(Self.Handle);
```

PASO 2: Al destruir nuestro formulario debemos eliminarlo de la cadena; Para ello podemos utilizar la API *ChangeClipboardChain* cuyos parámetros son el *Handle* de nuestro formulario y el *Handle* de la siguiente (con lo que el sistema reestructurará la CCB).

```
ChangeClipboardChain(Self.Handle, Self.SiguienteHandle);
```

PASO 3: Debemos programar la función que captura el mensaje de cambio en la CCB. Cuando un elemento se elimina de la cadena el sistema envía en mensaje *WM_CHANGECHAIN*. El primer parámetro del mensaje es el *Handle* de la ventana que se elimina de la cadena, y el segundo parámetro es el *Handle* de siguiente ventana a la que se elimina. Si la ventana que se elimina es la que nosotros tenemos como siguiente, debemos actualizar nuestro apuntador; Si no es así, debemos propagar el mensaje al siguiente elemento de la cadena.

```

Procedure TFVisor.WMCHANGECHAIN(var Message: TMessage);
begin
  // Se ha eliminado el siguiente en la cadena a nosotros ?
  if (Message.Remove= Self.SiguienteHandle) then begin
    // Actualizamos nuestro apuntador al siguiente
    SiguienteHandle := Message.Next;
  end
  // el eliminado no es el siguiente a nosotros ==> propagamos el mensaje
  else begin
    // Hay otro después?
    If (SiguienteHandle <> 0) then begin
      // Pasamos:
      // + A quien le enviamos el mensaje
      // + el mensaje
      // + los dos parámetros en el orden correcto
      SendMessage(SiguienteHandle, WM_CHANGECHAIN,
                  TMessage(Message).WParam,
                  TMessage(Message).LParam);
    end;
  end;
end;

```

PASO 4: Por último debemos capturar el “mensaje de cambio” en el contenido del *Portapapeles* (que realmente es el que nos interesa...). Cuando se produce un cambio el sistema envía el mensaje al primer elemento de la CCB. Cada elemento de la cadena cuando recibe el mensaje, además de realizar su propia tarea está obligado a propagar el mensaje al siguiente elemento.

```

procedure TFVisor.WMDRAWCLIPBOARD(var Message: TMessage);
begin
  // Propagamos el mensaje al siguiente elemento de
  //la cadena (no tiene parámetros)
  sendmessage(SiguienteHandle, WM_DRAWCLIPBOARD,0,0);

  // Realizamos la tarea que necesitamos...

  // ó comprobar si es una imagen...
  if Clipboard.HasFormat(CF_BITMAP) then begin
    //...

    // ó comprobar si es texto...
    if Clipboard.HasFormat(CF_TEXT) then begin
      //...

      // ó Activar/desactivar "Pegar"...
      Self.btPegar.Enabled := Clipboard.HasFormat(CF_BITMAP);

      // ó Visualizar contenido...
      Img.Picture.LoadFromClipboardFormat(CF_BITMAP,
                                          Clipboard.GetAsHandle(CF_BITMAP),0);
    end;
  end;
end;

```