Delphi paso a paso (II): Menús

Por Vladimir Algara

Exponer de nuevo el tema de los menús en Algoritmo (o en ***clippeRmanía***) es algo que no va a deslumbrar a nadie, pero hacerlo desde *Delphi* sí puede interesar a más de uno.

Como ya se comentó en la entrega anterior, Delphi posee un editor donde implementar cualesquiera de los elementos habituales de una ventana; en este número vamos a ver cómo utilizar el específico de menús.

Cualquier editor de menús de cualquier lenguaje suele ser fácil de manejar, y con ellos implementamos un elemento omnipresente en las ventanas principales de las aplicaciones.

Con este artículo describiremos el uso genérico de este editor, así como aquellas características más peculiares.

Dado que un menú va asociado a una ventana (principal o secundaria), la lógica de creación consiste en la implementación de uno de estos menús y, una vez terminado, su asociación posterior a la ventana. Una vez que el menú se crea está disponible para cualquier ventana.

Editor de Menús

Una vez dadas estas nociones generales pasaremos a describir el funcionamiento y los elementos del editor de menús (ver figura 1).

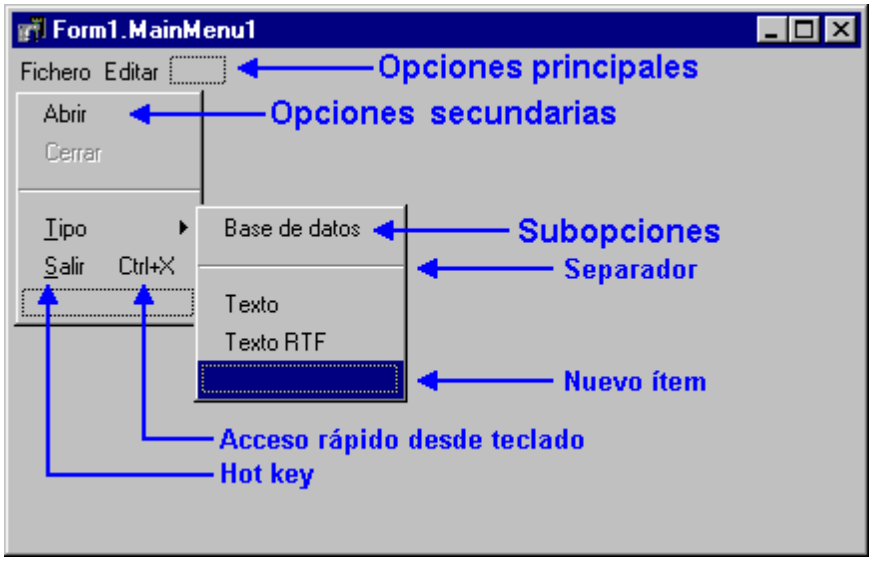

Figura 1.

1.- Línea donde se escribirá cada opción principal del menú.

En esta línea se especifican las distintas opciones principales que componen el cuerpo central del menú.

2.- Líneas donde se escribirán cada una de las opciones secundarias.

En esta línea se especifican las distintas subopciones de las opciones principales.

3.- Líneas donde se escribirán cada una de las subopcines.

En esta línea se especifican las distintas subopciones anidadas.

En cualquiera de los tres casos anteriores (o cualquier otro nivel de anidación de subopciones), si quisiéramos activar un *Hot-Key* para cada opción, bastaría con anteponer a la letra en cuestión el símbolo &:

```
&Salir crearía como Hot-Key la letra S
```

Para crear subopciones a una opción, lo único que tendremos que hacer es pulsar el botón izquierdo del ratón sobre la opción secundaria seleccionada, eligiendo *Create Submenu* (o pulsando *[Ctrl-flecha derecha*] como se informa en la figura 2) en el menú local asociado.

4.- Separadores entre subopciones. Para añadir este elemento visual basta con escribir un guión y pasar a la siguiente línea. Aparecerá una línea separadora.

5.- Abreviatura de teclado para acceso rápido a opciones.

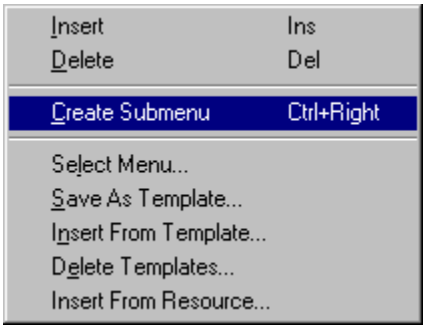

Figura 2

Como podemos observar en la figura 1, el menú creado posee todos estos elementos. Es bastante intuitivo y en él podemos observar distintos subniveles, así como los *hot-keys* asociados a cada opción del mismo.

El inspector de objetos

El cometido de esta ventana, como ya se adelantó algo en la entrega anterior, es el de la especificación de cada uno de los elementos que, a lo largo del ciclo de desarrollo, vamos dando a las partes que completan la aplicación.

En los menús, podremos declarar los siguientes conceptos:

Propiedades del menú

Las características del menú como un todo, como algo que engloba a los distintos ítems. La figura 3 se muestra cuando pinchamos sobre un elemento de tipo menú.

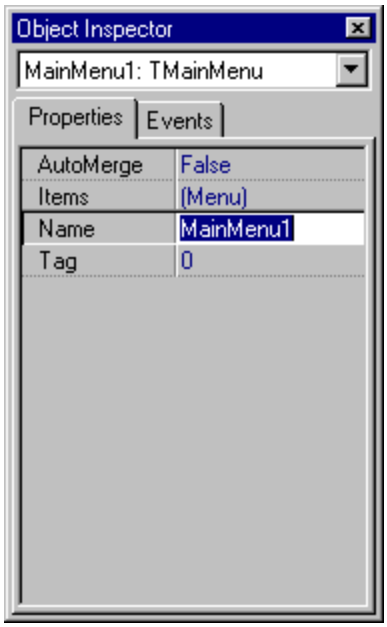

Figura 3.

AutoMerge: Premite que el menú se muestre nada más arrancar la ventana, o no.

Items: Permite entrar en el editor de menú (pinchando en los puntos suspensivos asociados), y, por tanto, modificarlo y/o consultarlo. Tiene el mismo efecto que hacer doble clic sobre un elemento de tipo menú.

Name: Nombre simbólico a través del que nos referiremos al menú (nombre de la clase que lo alberga).

Tag: Valor numérico por el cual conocemos el orden de tabulación de un elemento. Aplicable a todos los componentes visuales de una ventana.

Propiedades de los ítems

La relación de cada característica de los ítems de un menús (ver figura 4).

Figura 4

Break: Fija uno o varios puntos de ruptura. Esta característica permite obtener menús en forma de caja, en la que cada ruptura (agrupación) engloba elementos de un mismo tipo para una determinada opción. Por ejemplo, si queremos obtener la figura 5, bastaría con decir que el *Break* de los ítems "Base de datos" estuviera a *mbBreak*, en vez de a *mbNone* (opción por defecto). Si como en la figura 5 se desea, además, separar cada una de estas rupturas, lo que se tendrá que especificar será *mbBarBreak*. El resto las dejaremos en *mbNone*.

Figura 5: Menú con agrupaciones de opciones

Caption: Literal descriptivo del ítem. En este punto es donde se puede añadir el carácter & para subrayar una letra del *caption*.

Checked: Añade una marca al ítem del menú. Este tipo de símbolos se utiliza cuando deseamos que una cierta operación este siempre sujeta, o no, a un comportamiento estándar. Por ejemplo, habilitar que las ventanas siempre aparezcan expuestas en mosaico, o que los iconos aparezcan

organizados, etc. La marca también aparece en el propio editor de menús.

Enabled: Muestra atenuado (valor *false*) o no el ítem afectado. Se muestra así en el editor de menús, no sólo en la prueba ejecutable.

GroupIndex: Establece el número de iconos (asociados a los ítems) que pertenecerán a una sola agrupación.

HelpContext: Especifica, a través de un número, el tipo de ayuda que se visualizará cuando el usuario pulse la tecla [F1] en algún lugar de la aplicación. *HelpContext* es un aspecto aplicado no sólo a menús, sino a otros contextos; debido a su especificidad y a lo prodigado de su uso, se le dedicará, en otra entrega, un estudio pormenorizado.

Name: Como ocurría con el menú, nombre simbólico del ítem.

ShotCut: Combinación de teclas para la llamada rápida al proceso asociado a un ítem del menú. Están contempladas las teclas [Ctrl-tecla], las teclas de función hasta la [F12], en combinación o no con Ctrl y/o Shift y demás teclas auxiliares como [Supr], [Inicio], etc., también con Ctrl y/o Shift. El literal de la abreviatura de teclado aparece a continuación del ítem al que se le ha asociado.

Tag: Valor numérico para el orden de tabulación.

Eventos

Los menús, y más concretamente los ítems que componen un menú, sólo tienen un evento; aquél que sucede cuando se elige alguna de las opciones. Por eso la pestaña *Events* sólo dispone del evento *OnClick().*

Si queremos especificar un evento concreto, basta con escribir en el *combo* asociado a *OnClick()* el nombre del procedimiento que queremos ejecutar cuando hagamos *Clic,* y pulsar [Intro]. Esto hará que el nombre de dicho evento se registre en el bloque *type*, y que se escriba, automáticamente, las líneas esenciales de ese método. Por ejemplo si escribimos Salir veremos el código del fuente 1:

```
// --- Fuente 1 ---------------------------------------------------
type
. . .
. . .
   procedure Salir(Sender: TObject);
. . .
. . .
procedure TForm1.Salir(Sender: TObject);
begin

end;
```

Ahora sólo hay que escribir entre el *begin* y el *end* lo que queremos ejecutar. En este caso, como lo que queremos es salir, diremos *Close*.

```
// --- Fuente 2 ---------------------------------------------------
procedure TForm1.Salir(Sender: TObject);
```

```
begin
      Close;
end;
```

Otra posibilidad es dejar a Delphi que ponga nombre al método asociado a *OnClik()*. Como será habitual, esto lo haremos haciendo doble clic con el ratón en vez de escribir nosotros el nombre.

Si ya existen método definidos, podremos elegir, si ha lugar, algunos de los ya escritos.

Si más de un ítem ejecuta el mismo método, dentro de éste podremos distinguir si es uno u otro, por medio del parámetro *Sender*. Supongamos que el ítem *Cortar* y el ítem *Copiar* ejecutan un método común que realiza una determinada cantidad de operaciones, al final de las cuales, efectivamente, queremos que lo que esté marcado en ese momento sea *Cortado* o *Copiado*. Para resolverlo nada más fácil que definir (en *Cortar*, por ejemplo) el método *CortaryCopiar()* y pulsar [Intro]; se generará el código asociado y comenzaremos a introducir el código común a ambos, luego, por medio de *Sender*, determinaremos cuál se pulsó y actuaremos en consecuencia (ver fuente 3).

```
// --- Fuente 3 ------------------------------------------------------
procedure TForm1.CortaryCopiar(Sender: TObject);
begin
      // Código común
      Form1.caption := 'Se está usando el Cut&Paste';
      . . .
      . . .
      // Fin del código común

      if Sender==Cortar then Edit1.cut
            else Edit1.copy;
end;
```

Donde *Edit1* es el *Name* de un hipotético control de edición y *Cortar* el *Name* asociado al ítem *Cortar* del menú. Se supone que el método *CortaryCopiar* sólo es llamado por *Cortar* o *Copiar*, de ahí que si el que llama (*Sender*) no es *Cortar* no quede más remedio que sea *Copiar*. Si hubiera más posibilidades habría bastado con proteger cada operación particular con su *if* correspondiente, o con una estructura *case* que tuviera en cuenta la casuística.

Ejemplo

Pongamos un ejemplo en el que vamos a jugar con dos controles de edición, en los que vamos a permitir cortar y/o copiar, así como pegar. También se permitirá, cuando tenga algo, vaciar el contenido del portapapeles. Para que el efecto visual sea aparente, inhabilitaremos las opciones del menú no permitidas.

La ventana que recoge este proceso es el la de la figura 6 y el código que hay que ensayar el del fuente 4.

Figura 6: Aplicación para uso de menús

```
// --- Fuente 4 ----------------------------------------------------------
unit menu2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, Buttons, ExtCtrls, OleCtrls, ChartFX, StdCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    Editar: TMenuItem;
    Cortar: TMenuItem;
    Copiar: TMenuItem;
    Pegar: TMenuItem;
    Vaciar: TMenuItem;
    N2:        TMenuItem;
    SBCortar: TSpeedButton;
    SBCopiar: TSpeedButton;
    SBSalir: TSpeedButton;
    Edit1: TEdit;
    Edit2: TEdit;
    SBPegar: TSpeedButton;
    SBVaciar: TSpeedButton;
    procedure Salir(Sender: TObject);
    procedure CortaryCopiar(Sender: TObject);
    procedure PegarDesde(Sender: TObject);
    procedure VaciaPapelera(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Salir(Sender: TObject);
begin
     Close;
end;
```

```
procedure TForm1.CortaryCopiar(Sender: TObject);
begin
     Form1.caption := 'Se está usando Cut&Paste';
     Pegar.Enabled := True;
     Vaciar.Enabled := True;
     SBPegar.Enabled := True;
     SBVaciar.Enabled := True;
     if Sender=Cortar then Edit1.CutToClipboard
        else Edit1.CopyToClipboard
end;

procedure TForm1.PegarDesde(Sender: TObject);
begin
     Edit2.PasteFromClipboard
end;

procedure TForm1.VaciaPapelera(Sender: TObject);
begin
     Pegar.Enabled := False;
     Vaciar.Enabled := False;
     SBPegar.Enabled := False;
     SBVaciar.Enabled := False;
end;

end.
```

Porciones de código a resaltar

En la porción identificada como *type* se declara la variable *TForm1,* que se instancia desde la clase *TForm*

```
TForm1 = class(TForm)
```

Una relación de todos los ítems del menú (clase *TMenuItem*), de los botones de acceso rápido (*TSpeedButton*), de los controles de edición (clase *TEdit*) y de los métodos llamados al producirse un cierto evento (ver fuente 5).

```
// --- Fuente 5 ------------------------------------------------------
type
  TForm1 = class(TForm)
    MainMenu1:      TMainMenu;
    Editar:         TMenuItem;
    Cortar:         TMenuItem;
        . . .
    SBCortar:       TSpeedButton;
    SBCopiar:       TSpeedButton;
    SBSalir: TSpeedButton;
    Edit1:          TEdit;
    Edit2:          TEdit;
    SBPegar: TSpeedButton;
    SBVaciar:       TSpeedButton;

    procedure Salir(Sender: TObject);
    procedure CortaryCopiar(Sender: TObject);
    procedure PegarDesde(Sender: TObject);
    procedure VaciaPapelera(Sender: TObject);
```

En lo que a declaración de variables se refiere, se define *Form1* como representación de la ventana.

```
var
  Form1: TForm1;
```

Los procedimientos

Salir

Se invoca al pulsar sobre el botón de acceso rápido *Salir*. Su cometido: cerrar la ventana y, por ende, la aplicación.

```
procedure TForm1.Salir(Sender: TObject);
begin
        // Cierra la ventana
     Close;
end;
```

CortaryCopiar

Como se ha comentado un poco más arriba, este procedimiento es común a las opciones de cortar y copiar (en lo que se refiere a la habilitación y deshabilitación de controles) y particular a la hora de almacenar en el portapapeles y respetar el texto seleccionado o eliminarlo. Ver porción de código en el fuente 6.

```
// --- Fuente 6 ------------------------------------------------------
procedure TForm1.CortaryCopiar(Sender: TObject);
begin

        // Parte común a Cortar y a Copiar
        Form1.caption := 'Se está usando Cut&Paste';
     Pegar.Enabled := True;
     Vaciar.Enabled := True;
     SBPegar.Enabled := True;
     SBVaciar.Enabled := True;

        // Parte diferenciada
        // Si el que ha llamado al procedimiento ha sido Cortar se hace Cut
        // Si no se hace Copy
     if Sender=Cortar then Edit1.CutToClipboard
        else Edit1.CopyToClipboard
end;
```

PegarDesde

Este otro se encarga de traer del portapapeles lo que haya allí grabado, almacenándolo en el control de edición cuyo *Name* es *Edit2*.

```
procedure TForm1.PegarDesde(Sender: TObject);
```

```
begin
     Edit2.PasteFromClipboard
end;
```

VaciarPapelera

Por último el método que vacía el portapapeles. Realmente el portapapeles no se vacía, lo único que se hace es inhabilitar la opción de *Pegar*, para impedir traerse, al menos desde las opciones de la ventana, el contenido del portapapeles. También se inhabilita la opción de *Vaciar*, pues es ilógico vaciar más de una vez el portapapeles ya vacío (ver fuente 7).

```
// --- Fuente 7 ---------------------------------------------------------
procedure TForm1.VaciaPapelera(Sender: TObject);
begin
     Pegar.Enabled := False;
     Vaciar.Enabled := False;
     SBPegar.Enabled := False;
     SBVaciar.Enabled := False;
end;
```

Conclusión

Aunque el programa es susceptible de muchas mejoras, como vaciar efectivamente el portapapeles, detectar que existe algo marcado en el control de edición *Edit1* y sólo entonces permitir el cortado o el copiado, etc.; se da una idea de cómo interactuar desde los ítems del menú con ellos mismos, con otros controles de la ventana (botones de acceso rápido y de edición) y con elementos de Windows (como es el portapapeles).

Además, se ha intentado ir introduciendo algunos elementos de programación que enseñen cómo especificar una orden desde el código fuente (inhabilitar botones, cerrar la ventana, etc.).