Delphi paso a paso (I)

Por Vladimir Algara

Cada vez menos, tratamos de discutir cómo o por qué tenemos que introducirnos en *Windows*, sólo queda por dilucidar cuándo y con qué herramienta. Para quienes trabajaban en *Clipper* el salto evidente parecía ser *Visual Objects*, para los eternos "xBaseros" la balanza se inclinaba hacia *Visual FoxPro*, para los de “Ceros” el Visual C++, etc.

Actualmente, a mi entender, existen cinco grandes herramientas para trabajar bajo Windows, que enumero alfabéticamente para no levantar sospechas: *Delphi*, *Visual Basic*, *Visual C++, Visual FoxPro* y *Visual Objects.* Todas ellas podrían someterse a examen y, unas u otras saldrían bien o mal paradas según el baremo aplicado. No cabe la menor duda de que cada programador pretende conseguir unas metas concretas, pero lo que también es cierto es que para conseguir estas metas el camino debería ser lo menos escabroso posible. Según estas dos premisas se podría hacer una primera división de esas cinco herramientas; en dos grandes grupos, uno que simbolizara la potencia del producto (en el que se valoraría total potencia *OOP* -fundamental-, tipificación de variables, implementación *DDE* entre aplicaciones, manejo ágil de los datos almacenados en archivo de bases de datos, acceso sin trabas al *API* de Windows, manipulación de los 32 bits de Windows 95, así como otras de menos importantes, pero que vinieran a engordar la bonanza del producto, como accesibilidad a controles *OLE* y controles de *Windows 95*, etc.) y otro su facilidad de manejo.

- Por potencia: *Delphi*, *Visual C++* (ordenados alfabéticamente) y, un poquitín más alejado, *Visual Objects*.
- Por facilidad de uso: *Delphi*, *Visual Basic* y *Visual FoxPro* (ordenados alfabéticamente por fabricante, *Borland* y *Microsoft*).

Dado que se ordene como se ordene siempre es Delphi quien encabeza las listas, en esta primera entrega se comienza un deambular (por no decir curso) a través de las facilidades de uso y potencia que *Delphi 2.0* (32 bits) ofrece a los desarrolladores que quieran programar bajo Windows y aún no sepan con qué hacerlo.

Características de Windows

Para empezar debemos dejar claros varios conceptos, no siempre evidentes para usuarios DOS.

El Sistema Operativo

El crecimiento del hardware ha marcado las pautas del software. Entornos de trabajo como Windows son impensables en máquinas de bajas prestaciones.

DOS es un sistema operativo de pocos recursos y, por consiguiente, rápido. Sin embargo es excesivamente artesanal, pues todo control, toda validación, todo supuesto error ha de ser previsto por el programador, no por el sistema operativo (como por ejemplo Windows). Además, Windows dota de grandes recursos con apenas codificación.

La multitarea

Windows es un sistema que permite realizar varias tareas a la vez; por ejemplo, una persona puede estar ejecutando una aplicación y mandando por módem un archivo a otra estación

remota. Esta potencia hay que tenerla en cuenta a la hora de diseñar las aplicaciones, pues esa segunda tarea puede ser, nuevamente, la propia aplicación. Incluso dentro de una aplicación, se puede hacer que un proceso se lance un número indefinido de veces (aplicaciones *MDI*), sin menoscabo de lo ya ejecutado.

La comunicación entre aplicaciones y/o procesos se puede efectuar a través de cuatro grandes caminos: el uso del portapapeles (cortar y pegar), la técnica de *Drag&Drop* (arrastrar y soltar), el intercambio dinámico de datos (enlaces *DDE*) y las incrustaciones *OLE*.

Los eventos

Por medio de los sucesos acaecidos en la vida de una aplicación, ésta le brinda al sistema una información; el sistema los lleva a cabo. Esta comunicación *aplicación-sistema* es fácilmente asimilable por una mente DOS, pero lo que puede no serlo tanto es la información que genera el sistema y que nuestra aplicación tiene la potestad de utilizar o no. Como se aprecia es una comunicación de dos direcciones y, además, constante.

La gestión de memoria

Windows reutiliza y compacta la memoria según sus necesidades. Cuando se cierra alguno de los programas que se mantienen abiertos Windows se encarga de liberar la memoria éste utilizaba, manipulando los datos y código a su conveniencia y optimizando al máximo la ubicación de los segmentos de memoria. Asimismo, el uso de las librerías de enlace dinámico (*DLL*) permite:

- Cargar y descargar la librería en tiempo de ejecución (enlace dinámico)
- Que varios programas distintos accedan a una misma DLL sin que ocurran conflictos (independencia del ejecutable).
- Que una DLL esté confeccionada en un lenguaje y sea aprovechada desde otro (independencia del lenguaje).

Las herramientas visuales

Dado que Windows es un entorno gráfico, el diseño de ventanas (con una lógica similar en todos los lenguajes), iconos, cursores, etc., se realiza por medio de las herramientas que los distintos lenguajes proporcionan, por lo que los programadores pueden dedicar menos tiempo al diseño final y más al buen funcionamiento del algoritmo interno.

Por qué Delphi

La elección de un lenguaje u otro, lo expuse al principio, dependen de muchos factores determinantes.

Entre toda la oferta, y teniendo en cuenta que yo previamente trabajaba en Clipper y que sabía algo (no gran cosa) de *Turbo Pascal*, tenía dos claros candidatos, Visual Objects (con el que efectivamente empecé debido a mi herencia) y Delphi. Con Visual Objects he pasado alegrías y penurias, y si bien es una de las arquitecturas más sólidas, no posee un entorno amable. Con él tengo la seguridad de que cualquier cosa la voy a poder llevar a *cabo*, pero no sé cuánto voy a tener que invertir en tecnología ni en tiempo. Con Delphi y Visual C++ tengo esa misma potencia, pero con ambos dispongo de infinita más flexibilidad, tengo la posibilidad de no ser artesano

(aunque puedo serlo), de no depender de la feliz idea, de la posibilidad de ser más tonto, ¡qué gusto!

Como no sé *C*, ni *BASIC* y quería olvidarme del *@...SAY*, mi suerte estaba echada. Voy a intentar comentar las facilidades de uso de Delphi 2.0 e intentar abordar, a lo largo de las distintas entregas, las operaciones más comunes a la hora de desarrollar una aplicación para Windows.

El entorno

Los requerimientos

De todos los paquetes, también Delphi es el que menos requerimientos hardware necesita. Aunque puede realizar una instalación compacta de unos *20Mb* cuando se quiere cargar todo (aunque no se utilice en la vida) se necesitan alrededor de *60 Mb*.

Al producto, en tiempo de desarrollo, le hacen falta un mínimo de *4 Mb* de memoria (lo que es justo y necesario para Windows 95) aunque con más va mejor, claro; *8 Mb* es una cifra con la que nos podemos empezar a sentir a gusto. Visual Objects, Visual FoxPro, etc. necesitan *8 Mb* para empezar a hablar y 16 para empezar a sentirse a gusto, Visual C++ fija el mínimo en *16 Mb* y *20Mb* para un buen hacer.

La mesa de trabajo

Se entiende por mesa de trabajo el lugar donde se encuentran todas las piezas disponibles para la confección de una aplicación. En Delphi se advierten cinco grupos de elementos claramente diferenciados.

- *El menú principal*. En él se recogen todas las opciones que se pueden realizar desde Delphi: empezar un nuevo trabajo, imprimir el que está en curso, compilar, ejecutar, acceder a la ayuda dependiendo del contexto en el que nos encontremos, etc.

- *La barra de herramientas*. Conjunto de iconos que simbolizan las operaciones más comunes. La que se ve en la figura 1 no es la que ofrece Delphi por defecto, sino una configurada al gusto del consumidor (o sea, al mío).

- *La paleta de controles*. Colección de los elementos que se pueden ubicar en las ventanas de una aplicación. Se trata de una colección de carpetas (*tabs*), cada una de ellas, a su vez, con una colección de controles al uso. Este elemento y los dos anteriores se pueden ver en la figura 1.

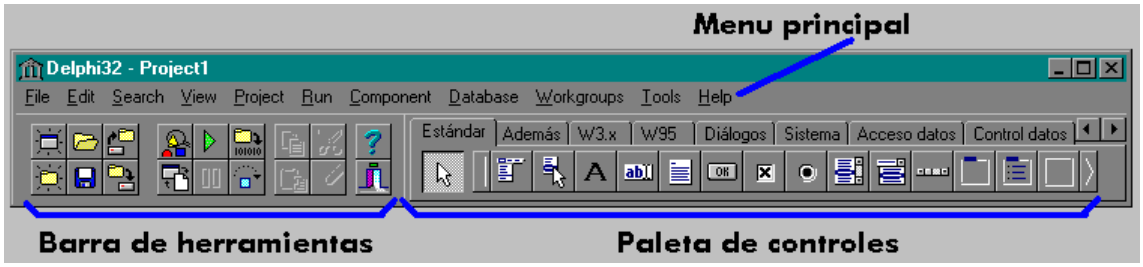

Figura 1: Menú principal, barra de herramientas y paleta de controles.

- *El inspector de objetos*. Ventana donde se permite modificar en fondo y forma el control ubicado en la ventana, o la propia ventana. Dado que un control (o la ventana) está sujeto a sus características y a los eventos que se pueden producir en él, el inspector de objetos posee dos carpetas (*tabs*) para cada una de las operativas. Ver figura 2.

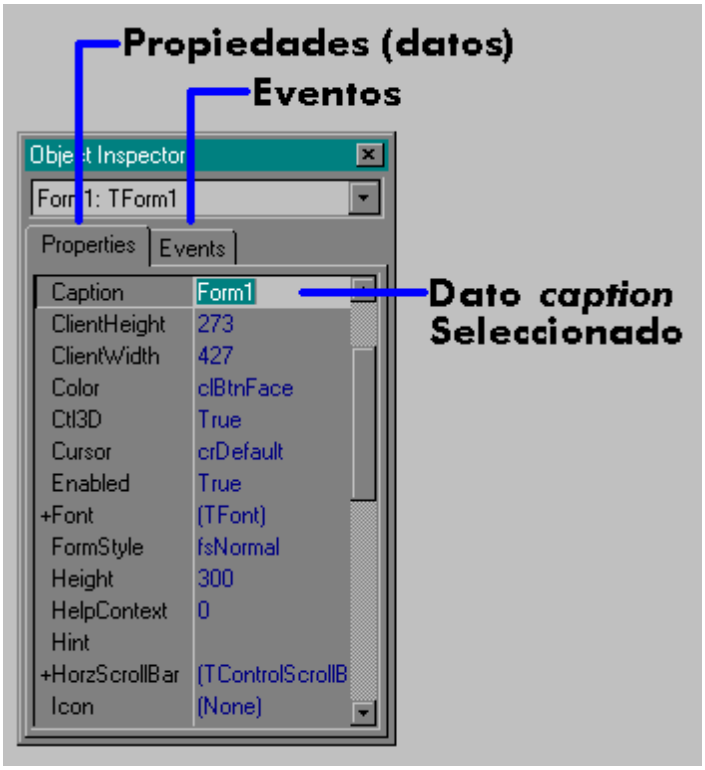

Figura 2: El inspector de objetos

- *Ventana de trabajo* (la que se verá en la aplicación una vez que la *echemos a andar*) y código asociado a esa ventana. Ver figura 3

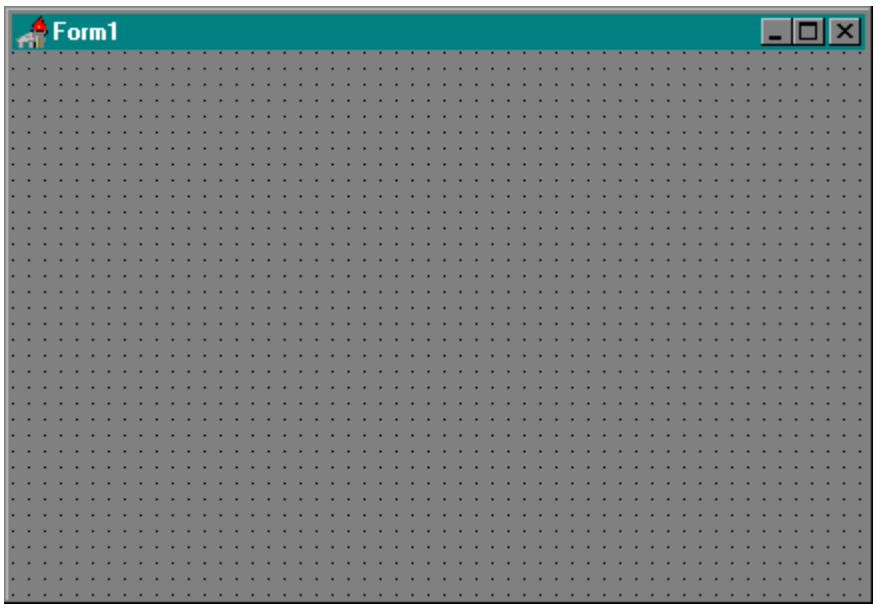

Figura 3: Ventana Form1

Los archivos y la OOP

La unidad de trabajo en cualquier aplicación hecha desde Delphi es el proyecto (*Project*). Dentro de un proyecto se almacenan los formularios (cada ventana de la aplicación es un formulario) o unidades (*units*) que utilizará nuestra aplicación. Dada la arquitectura de Delphi todo se entiende como *OOP*, incluso los mencionados formularios son una instanciación de la clase *TForm*.

Pero, antes de asustar al personal con tanto OOP para arriba y para abajo, veamos en qué se traduce eso a la hora de implementar un programa. La respuesta es sencilla, una ventana se manipulará como un objeto, y, como tal, llevará un nombre puesto por mí. Por ejemplo, si la ventana queremos que se llame *Lucero*, acudiremos a la propiedad *Name* en el inspector de objetos y cambiaremos *Form1* por *Lucero*. Este cambio provoca la siguiente modificación automática:

```
Lucero: TLucero
```

Añadiéndose una línea donde se dice:

```
TLucero = class(TForm)
. . .
var
   Lucero: Tlucero;
```

Eso quiere decir que la ventana creada, se llame como se llame, es una instanciación de la clase padre *TForm*, que, como tal, tiene todas sus características. Seguidamente se define la variable *Lucero* de esa clase recién creada

Una vez en este punto, sabemos que el objeto *Lucero* (la ventana) se compondrá de un ancho, de un alto, de una posición de comienzo, de un título (*caption*), de un botón de cierre (aspa en Windows 95) o no, etc.

Todos estos datos de los objetos vienen meridianamente explicados en la ayuda que acompaña a Delphi; así, por ejemplo, si se desea saber más sobre el dato *caption* de la ventana *Lucero*, buscaremos *caption* en la ayuda en línea (*Contents* en *Help*). Aparecerá la descripción de la figura 4.

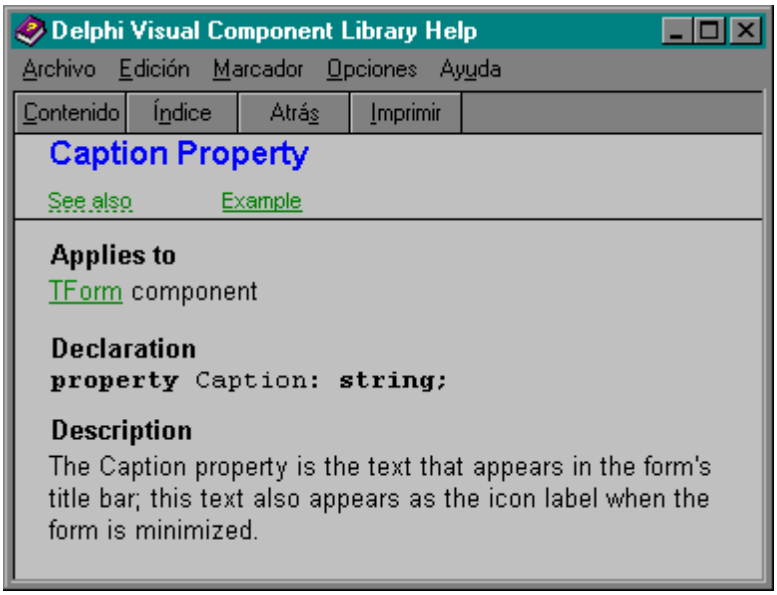

Figura 4: Ayuda sobre el dato *caption*

La información aportada en esta ventana del *Help*, además de la descripción, es el componente al que pertenece (*TForm*), un ejemplo de uso (zona sensible *example*), temas relacionados (zona sensible *see also*) y tipo de dato que contiene (**property** caption: **string;**) en nuestro caso una cadena de caracteres.

El *caption*, como ya se sabe, corresponde al título de la ventana. Existen dos formas de alterar su contenido, una por medio de la propiedad *caption* del inspector de objetos y otra mediante programa:

```
Lucero.caption := 'Título de ventana'
```

Intuitivamente se puede interpretar este código de la siguiente manera. El objeto *Lucero* tiene un dato llamado *caption* (al cual accedo mediante la sintaxis *Lucero.caption*) que modifico sin más que asignar un dato acorde a su tipo (según rezaba la ayuda, este tipo es carácter). Usando el operador de asignación (:=) almaceno la cadena entrecomillada.

El código fuente

Una interesante posibilidad, sobre todo al comienzo, consiste en ver qué código se va generando para la ventana en curso, además del siempre disponible del proyecto. Para ello pinchamos en dicha ventana con el botón derecho del ratón (izquierdo para los zurdos) y elegimos, en el menú local, verlo en forma de código fuente. Allí habrá una línea similar a la de más arriba. Es más, cualquier cambio en esta línea también afectará al contenido que visualiza el inspector de objetos. Para regresar a la ventana se vuelve a pinchar con el botón izquierdo y elegimos en el menú local verlo en forma gráfica.

Para aquellos que no conozcan nada de *Pascal* y vengan de un entorno *xBase*, decir que, como todo lenguaje, uno se empieza a encontrar a gusto cuando domina su sintaxis. La de *Pascal* es estructuralmente diferente, por ejemplo, a la de Clipper. A saber.

Estructura de los programas

En Clipper se comienza con una función principal en un archivo *PRG* principal que va llamando sucesivamente a otras funciones y/o procedimientos. En Delphi el módulo principal es el del proyecto y tiene una estructura como la del fuente 1:

```
// --- Fuente 1 ------------------------------------------------------
program Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Lucero},
  Proy in 'PROY.PAS';

{$R *.RES}

begin
  Application.CreateForm(TLucero, Lucero);
  Application.Run;
end.
```

En primer lugar la palabra reservada *program*, que indica el comienzo de un programa Delphi, luego los módulos que intervienen en el proceso (*uses*), indicadores útiles para el compilador (*{$R *.RES}*) y comienzo de los elementos ejecutables (entre las palabras reservadas *begin* y *end*) y punto final (.). Lo que se ejecuta, en este caso, es la creación de un formulario, tomando como base la clase *Tlucero* y su variable *Lucero*. Esto mismo, en Clipper sería algo idéntico a:

```
Lucero := Application.CreateForm(TLucero);
```

Por otra parte, las distintas unidades presentan la estructura del fuente 2.

```
// --- Fuente 2 ------------------------------------------------------
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TLucero = class(TForm)
private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Lucero: TLucero;

implementation

{$R *.DFM}

end.
```

Encontramos una primera sección para el interfaz a usar (*Interface*), seguida de los módulos necesarios para *echar a andar* la aplicación (*uses*), otra sección para la definición de nuevos tipos de datos (*type*) donde se declara la nueva clase *TLucero* (que hereda de *TForm*), la declaración de variables privadas y públicas a nivel de aplicación (luego habrá locales a nivel de función y/o método), las variables de aplicación (sección *var*) donde se especifica *Lucero* como tipo *TLucero*. Tanto el fuente 1 como el 2 se generan de forma automática.

Separación de líneas

Las líneas se separan por punto y coma (;) como en C. Si hubiera una instrucción con más de una línea bastaría con escribir una detrás de otra y, al final, poner el punto y coma.

```
nValor1 := 1;
nValor2 := 1;
nValor3 := 1;
nValor4 := 1;
nSuma := nValor1 + nValor2
         nValor3 + nValor4;
```

Declaración de variables

Se usa la palabra reservada *var* para definir una o un conjunto de variables. Para ello se da el nombre de la variable, el identificador dos puntos (:) y se define un tipo válido.

```
var
    nValor1 : integer;
    nValor2, nValor3, nValor4 : integer;
    cCadena : string[10];
    oVentana : Tlucero;
```

Estructuras *begin...end*

Cuando en una estructura se especifica una sola acción no es necesario el uso de agrupaciones *begin...end*, pero si se van a realizar dos o más sí lo será:

```
if nValor1=1 then nValor2:=0
    else nValor2:=1000

if nValor1=1 then
    begin
        nValor2:=0;
        nValor3:=0;
        nValor4:=0;
    end
    else
    begin
        nValor2:=1000;
        nValor3:=1000;
        nValor4:=1000;
    end;
```

Valores de funciones

Existen funciones que devuelven valores que pueden ser almacenados en variables auxiliares (como en Clipper), pero hay otras que reciben como parámetro dicha variable auxiliar y es dentro de la función donde se altera su contenido (se pasan internamente como referencia).

```
dFecha := Date()           // Correcto

cValor1 := Str( nValor1 )  // No correcto
Str( nValor1, cValor1 )     // Correcto
```

Comentarios

Los comentarios a los programas comienzan abriendo un paréntesis y un asterisco y finalizan con un asterisco y un paréntesis cerrado. En la versión 2 también se pueden utilizar las dos barras (como en C o en Clipper).

```
(* Comentario a programa Delphi *)
// Comentario a programa Delphi
```

Ubicación de funciones

Todas las funciones, procedimientos y métodos han de estar al principio del programa, antes del *begin...end* principal. El *begin...end* principal debe acabar en punto.

```
procedure ...(...)
var
    ...
begin
    ...
end;

procedure ...(...)
var
    ...
begin
    ...
end;

begin
    ...
end.
```

Antes de terminar

De aquí en adelante todo va seguir la misma sintaxis expuesta. Por ejemplo, para codificar cada evento basta con hacer doble clic en alguno de los de la relación mostrada en el inspector de objetos. Automáticamente se escribe el correspondiente código, el cual, como se puede observar, se ciñe a la estructura de más arriba; eso sí, orientado a objetos.

Como empezar a modificar todos los datos de la ventana o los eventos que en ella se pueden manifestar haría que me extendiese en demasía, postergo toda esta básica maquinaria para una segunda entrega, en la que empezaremos a sacarle más jugo al lenguaje. En este momento, si se quiere ver cómo funciona la ventana *Lucero* con su *caption* modificado, bastará con pinchar sobre el icono de ejecución (triángulo verde) y, efectivamente, verla.