Creación de componentes VCL (II)

Eventos en los componentes

Vamos a profundizar algo más en la programación de componentes, centrándonos en los eventos y cómo podemos definir los nuestros propios.

Índice

Introducción

En nuestro anterior artículo (www.lawebdejm.com/?id=22310) acabamos desarrollando un pequeño componente que nos permitía realizar una conversión de divisas, aunque dejamos algunos aspectos en el aire. Uno de estos aspectos trataba sobre cómo podíamos definir nuestros propios eventos, para que el desarrollador que utilice el componente, pueda programar lo que quiera en ellos.

Pero... ¿qué es un componente?

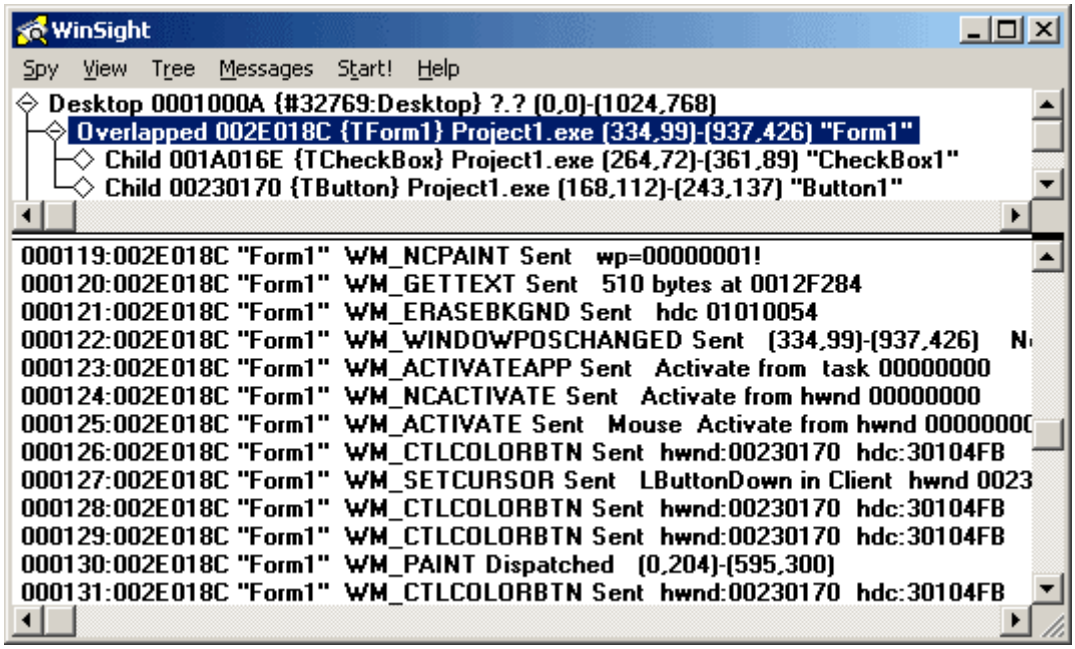
Los eventos son uno de los aspectos más importantes en entornos gráficos como Windows, ya que una misma acción puede realizarse de distintas formas.
Por ejemplo, una opción de menú tiene distintas maneras de activarse:

1. Podemos hacer clic sobre el menú.
2. O bien podemos utilizar el acceso directo del menú (la combinación de teclas que aparece al lado del nombre, como Ctrl+V, F5, etc.)
3. O incluso podemos utilizar los aceleradores del teclado asignados menú (la letra subrayada que aparece en el nombre, como Archivo, Edición, etc.)

Como vemos, tenemos muchos caminos para llegar al mismo destino, y en ciertas ocasiones sería demasiado complicado controlar todas estas opciones. Es por esto que el propio sistema operativo Windows nos "avisa" cada vez que ocurre algo importante: "Eh aplicación!, que el usuario está haciendo clic sobre tu botón Aceptar". "¡¡Atención!! el usuario ha dicho que quiere cerrar tu programa ¿qué hacemos?"

Cada uno de estos "avisos" recibe el nombre de "mensaje" y existen gran cantidad de ellos: cada vez que se pulsa el botón derecho del ratón, cuando se presiona una tecla del teclado, cuando se pinta la ventana, etc. Cada uno de los mensajes tiene un número único, y una constante para representar a ese número. Por ejemplo: WM_LBUTTONUP (el número 514) o WM_SYSKEYDOWN (el 260) son algunos de ellos.

Por ello, la programación basada en eventos encaja muy bien en los entornos gráficos, ya que cada vez que la aplicación reciba un mensaje del sistema operativo, podremos lanzar un evento al programador.
En entornos RAD, como Delphi, los eventos son funciones que se ejecutan solas, como por arte de magia, cada vez que ocurre un suceso especial.
A veces, como hemos dicho, el evento se ejecuta cuando nuestra aplicación recibe un mensaje del sistema operativo (que nos avisa de que algo está ocurriendo), y otras veces el evento se lanza cuando el propio Delphi nos quiere avisar de algo.



Por ejemplo: como todos sabemos, el evento `OnClick` de un botón se ejecuta cada vez que este se pulsa. Pero recordemos que hay varias formas de conseguir esto: haciendo clic sobre el propio botón, pulsando la barra espaciadora cuando tiene el foco, pulsando la tecla "Enter" cuando se trata del botón por defecto, o incluso tecleando la combinación "Alt" más la letra subrayada del texto del botón. Windows lanzará distintos mensajes dependiendo de lo que haya ocurrido: `WM_LBUTTONUP`, que corresponde a la acción de soltar el botón izquierdo del ratón, o bien `WM KEYUP` si hemos utilizado el teclado.

En el momento de recibir el mensaje, Delphi interpretará por nosotros las distintas posibilidades, y lanza el evento `OnClick` cuando corresponda. De este modo, no tenemos que preocuparnos de lo que haya ocurrido (ha hecho clic, ha pulsado Enter...) sino que finalmente nuestro botón ha sido pulsado, por el método que sea, por lo que posiblemente, algo tendremos que hacer algo.

Los eventos en Delphi

Existen otro tipo de eventos que no tienen nada que ver con los mensajes del sistema operativo. En ocasiones, el programador de un componente quiere avisar al programador de algo. Por ejemplo: un componente que hace un proceso largo (como descargar un archivo de internet) debe avisar al usuario de que la operación ha terminado. El sistema operativo no tiene nada que decir en este caso, sino que somos nosotros los que lanzamos el evento para que el programador que utilice el componente, pueda programar lo que quiera en ellos.

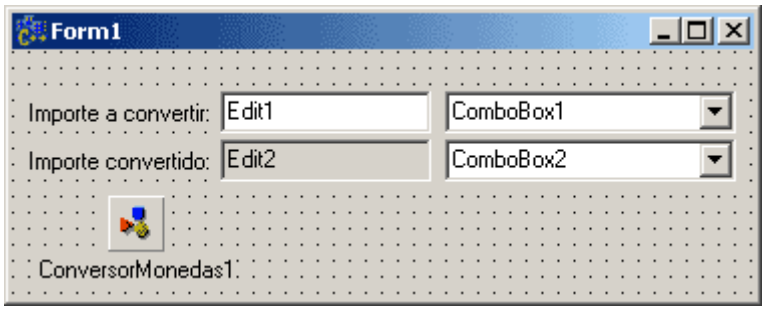
Por ejemplo, es habitual avisar al programador, a través de un evento, de que una operación se va a realizar, pudiendo así ser cancelada (como el evento `OnCloseQuery` de los formularios) o que el valor de cierta propiedad ha sido modificado.

De cara al usuario del componente, estos eventos son iguales que los otros, y el propio programador no sabrá si el evento surge de un mensaje de Windows o bien es un evento interno de Delphi.

Manos a la obra

Bueno, ahora que ya sabemos la teoría, podemos retomar nuestro componente de conversión de monedas, para añadir los eventos que necesitemos.

Si recordamos lo que hicimos, teníamos un componente que convertía una cantidad origen (de un tipo de divisa concreto) a otra cantidad destino (de otro tipo de divisa diferente).
Cada vez que cambiábamos la cantidad original, se recalculaba el resultado, y cuando cambiábamos el tipo de moneda origen o destino, también se obtenía el nuevo resultado.



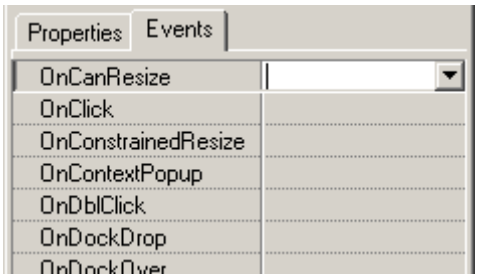
Así que tenemos una acción (el cálculo del resultado) que puede ejecutarse por diversas razones.
Supongamos que hemos situado nuestro componente en un formulario, y hemos añadido dos campos de edición (`TEdit`): uno para el valor origen y otro para el destino. Además, añadimos dos cuadros combinados para que el usuario seleccione con qué tipos de monedas quiere trabajar.

Sabemos que, cada vez que cambie el valor de `Edit1`, tenemos que actualizar la propiedad "ValorConvertir" de nuestro componente y cada vez que se seleccione un tipo de moneda, cambiar las propiedades "MonedaConvertir" y "MonedaConvertido".

¿Pero cuándo mostramos el nuevo valor calculado? Podríamos hacerlo en cada uno de esos tres momentos: al cambiar el Edti1 o al cambiar cualquiera de los cuadros combinados.
Pero... ¿Y si tuviéramos un evento que saltase cada vez que se haya calculado el resultado? ¿Nos resultaría útil para mostrar el nuevo valor? Pues podría ser buena idea, ya que simplemente codificaríamos este evento, y si, por cualquier razón, el resultado cambia, el evento se disparará y nuestro código podrá ejecutarse. Así que, como esta vez somos nosotros los que tenemos la sartén por el mango, vamos a hacerlo así.

Empezado desde el principio

Lo primero que tenemos que saber es cómo definir un evento en Delphi, para que aparezca como cualquier otro en el "Inspector de Objetos".



Pues para esto basta con saber que un evento no es más que un puntero a una función. Bueno, quizá esto sea algo nuevo para muchos de vosotros.
Vayamos por partes.

¿Un puntero a qué...?

Todos sabemos que los ordenadores tienen memoria RAM, donde se almacena información volátil (que se pierde al apagar el ordenador). Bueno, en realidad se almacena en la memoria virtual (http://www.lawebdejm.com/?id=22101), pero eso es otra historia y debe ser contada en otra ocasión. (:
Para simplificar las cosas podemos pensar en esta memoria como si fuera una fila muy larga de casillas, donde cada casilla tiene asignado un número secuencial: 0, 1, 2, 3... etc. A este número se le llama "dirección de memoria" y podemos estar seguros de que sólo habrá una casilla por cada dirección.

Un puntero es una variable (como cualquier otra) que almacena una dirección de memoria.
Así de sencillo. En esta variable podemos almacenar cualquier dirección, desde la dirección 0 (el valor nil), hasta la dirección $FFFFFFFF, que es la máxima dirección posible en plataformas Windows de 32 bits. Cuando un puntero contiene el valor nil (0) decimos que es un puntero nulo, y debemos tratarlo como si estuviera vacío (no apunta a nada).

Cuando en un puntero almacenamos una dirección de una variable de tipo integer, decimos que es un *puntero a integer*, cuando almacenamos la dirección de una variable char, se trata de un *puntero a char*. En realidad, podemos utilizar un puntero para guardar la dirección de cualquier tipo de variable: un integer, real, char, registros, etc.

Y como todos sabemos, en la memoria de un ordenador, además de datos (las variables), también hay código (las funciones). Este código reside en otras direcciones, que, por qué no, podemos guardar también en un puntero. Cuando en un puntero, almacenamos la dirección de memoria donde está localizada una función, decimos que es un *puntero a función*.

Cuando tenemos un puntero a una variable, podemos leer (o cambiar) el contenido de esa variable. Del mismo modo, cuando tenemos un puntero a una función, podemos ejecutar esa función fácilmente.

Punteros en Object Pascal

En Delphi es sencillo definir punteros a variables y a funciones. Como norma general, para definir un puntero basta con declarar una variable, y anteponer el símbolo ^ al tipo de dato. Por ejemplo:

```
var
   punteroInteger: ^Integer;
   punteroChar:    ^char;
```

Para definir punteros a funciones, la cosa cambia un poco, ya que hay que definir los parámetros que acepta la función. Basta con declarar una variable de tipo "`procedure (parámetros)`" o "`function (parámetros): retorno`". Por ejemplo:

```
var
   punteroProc: procedure(parametro: boolean; otro: string);
   punteroFunc: function(const param: char; var otro: char): boolean;
```

Si vivimos en el mundo orientado a objetos, las cosas cambian ligeramente, porque tenemos que tener en cuenta un pequeño detalle: en vez de funciones utilizamos métodos. Para definir un puntero a método, basta con definir un puntero como si fuera a una función normal, y añadir la palabra "of object" al final. Por ejemplo:

```
var
   ptrMetodo: procedure(parametro, otro: string) of object;
   ptrMetodoFunc: function(param, otro: char): boolean of object;
```

Esto es todo lo que se puede decir sobre la declaración de punteros.
Sobre su uso, hay que conocer dos operadores:

`@variable`: que nos devuelve la dirección de una variable

`puntero^`: que nos devuelve el valor de la variable apuntada por el puntero

A continuación, podemos ver un ejemplo de uso de punteros.

```
var
   ptrInteger: ^Integer;
   ptrChar:    ^char;
   unInteger:   integer;
   unChar:      char;
begin
   unInteger := 1;
   unChar    := 'a';

   // se almacena en los punteros, las direcciones de
   // las variables que queremos apuntar
   ptrInteger := @unInteger;
   ptrChar    := @unChar;

   ptrInteger^ := 2;   // unInteger ahora vale 2
   ptrChar^    := 'b'; // unChar ahora vale 'b'
end;
```

Con funciones, la situación es parecida, tal y como podéis ver en el siguiente código:

```
function UnaFuncion(valor: boolean; otroValor: string): integer;
begin
   // ...
end;

function OtraFuncion(valor: boolean; otroValor: string): integer;
begin
   // ...
end;

function UsandoPunteros();
var
   ptrFuncion:  (valor: boolean; otroValor: string): integer;
begin
   ptrFuncion := @UnaFuncion;

   // ambas llamadas son equivalentes.
   UnaFuncion(true, 'hola');
   punteroFuncion(true, 'hola');

   // apunto a otra dirección, donde está otra función
   ptrFuncion := @OtraFuncion;

   // utilizo el mismo puntero para llamar a otra función
   OtraFuncion(false, 'adiós');
   punteroFuncion(false, 'adiós');
end;
```

Los eventos son propiedades

Pues sí, un evento no es más que un atributo de tipo "puntero a método", que se accede a través de una propiedad de lectura y escritura en la sección *published*. El primer paso será definir un nuevo tipo de dato, que sea un puntero a un método, del siguiente modo:

```
procedure TForm1.ConversorMonedas1Convertido(Sender: TObject;
                           nuevoResultado: float);
begin

end;
```

Con esto, hemos definido un nuevo tipo de dato que representa a un puntero a método. Posteriormente, podremos utilizarlo para declarar el evento de nuestro componente. El método "apuntado" utiliza dos argumentos: un objeto descendiente de `TObject` y un valor de tipo float.

Lo siguiente que tenemos que hacer es definir un atributo de este tipo, es decir: definir un atributo de tipo puntero a método. En el siguiente listado podéis ver cómo quedaría la sección *private* de nuestro componente después de declarar nuestro nuevo atributo de tipo puntero, que nos servirá para almacenar la

dirección de un método. Este método que apuntaremos, será el que codifique el programador cuando haga doble clic sobre el evento.

```
TConversorMonedas = class(TComponent)
private
  // declaración del puntero que contendrá la función del evento
  FOnConversion: TEventoConversion;

  FValorConvertir:  float;
  FValorConvertido: float;

  FMonedaConvertir:  TTipoMoneda;
  FMonedaConvertido: TTipoMoneda;

  ...
```

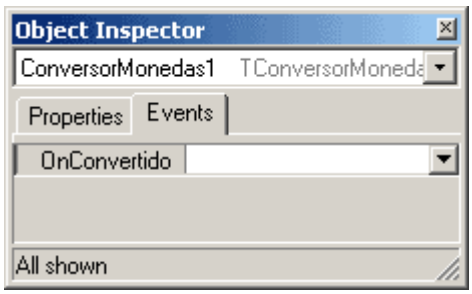
El siguiente paso es la definición de una propiedad, en la sección *published*, que nos permita crear o modificar el evento en tiempo de diseño. Podéis ver la nueva sección *published* en el siguiente código:

```
published
   property ValorConvertir:  float
                read  FValorConvertir
                write SetValorConvertir;
   property ValorConvertido: float
                read FValorConvertido;

   property MonedaConvertir:  TTipoMoneda
                read  FMonedaConvertir
                write SetMonedaConvertir;
   property MonedaConvertido: TTipoMoneda
                read  FMonedaConvertido
                write FMonedaConvertido

   // declaración del evento
   property OnConversion: TEventoConversion
                read  FOnConversion
                write FonConversion
```

Una vez que hemos declarado el evento, podemos registrar el componente para ver si aparece en el Inspector de Objetos, como cualquier otro evento, tal y como podéis ver en la imagen de la derecha.



Bien, ya tenemos nuestro evento, y si hacemos doble clic sobre él, veremos que el entorno de Delphi nos crea un nuevo método, en la clase que representa al formulario con el siguiente aspecto:

```
procedure TForm1.ConversorMonedas1Convertido(Sender: TObject;
                          nuevoResultado: float);
begin

end;
```

En realidad, al hacer doble clic, lo que ha ocurrido es que Delphi nos ha definido un nuevo método en la clase del formulario, y ha asignado la dirección de este método a la propiedad "OnConvertido" del componente ¡Y todo esto de un simple ratonazo!

Esta es una de las grandes maravillas de Delphi, y otros lenguajes han utilizado la misma idea (como C# de Microsoft).

Pero nuestro componente no está completo, ya que el nuevo evento `OnConvertido` no se ejecutará nunca, porque nadie ha dicho que se ejecute ¿no?

Simplemente debemos hacer una llamada al evento, cada vez que se produzca una conversión. Esto, como sabemos, se produce dentro del procedimiento "Convertir", así que, justo antes de asignar el resultado final, debemos llamar al evento. Hay que tener cuidado de llamar al evento sólo cuando el puntero tenga un valor distinto de `nil` ya que si tiene este valor, significará que el programador no ha hecho doble clic para codificar el evento, por lo que no debemos realizar la llamada, o se producirá un error de memoria. Esto lo podemos hacer con una simple comparación (`if FOnConvertido <> nil then...`) o bien utilizando la función `Assigned`, que sirve precisamente para esto: para ver si un puntero está apuntado (asignado) a algún sitio: `if Assigned(FOnConvertido) then...` Podéis ver el código final del método "Convertir" en el siguiente listado:

```
procedure TConversorMonedas.Convertir();
const
   FactorAEuro: array[TTipoMoneda] of float =
   (
       1,
       1.10,
       2,
       0.5,
       0.18,
       166.386
   );
var
   aux: float;
begin
   // se calcula utilizando el euro como referencia.
   aux := FValorConvertir / FactorAEuro[FMonedaConvertir];

   // ahora aux contiene el valor origen en euros. Se pasa
   // a la moneda destino
   aux := aux * FactorAEuro[FMonedaConvertido];

   // llamar al evento sólo si ha sido codificado
   if Assigned(FOnConvertido) then
      FOnConvertido(Self, aux);

   FValorConvertido := aux;
end;
```

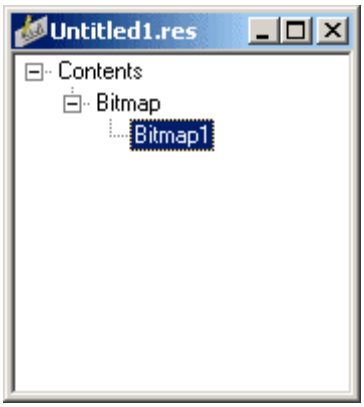
Bien, pues ya sabemos cómo definir eventos en nuestros propios componentes. Ahora nuestro conversor de monedas ha quedado más completo, y sabemos que el programador que lo utilice podrá realizar sus acciones en el nuevo evento.

Maquillando a nuestro componente

Ya para terminar tan sólo queda deciros que podéis descargar el código Después de todos los pasos que hemos ido dando, ya podemos decir que nuestro componente está funcionando al cien por cien. Sin embargo hemos dejado algunos flecos estéticos para el final. Uno de estos es el icono con el que aparece en la paleta de componentes.

Como ya sabéis, cada componente muestra su propio icono en la paleta, para que así el programador, pueda reconocerlo rápidamente durante el desarrollo. Para decirle a Delphi qué icono debe utilizar debemos realizar unos pasos muy sencillos.
En primer lugar, desenpolvaremos nuestra vena artística y crearemos un icono para nuestro componente. El resultado debe ser una imagen de 24x24 pixels, con un máximo de 256 colores.

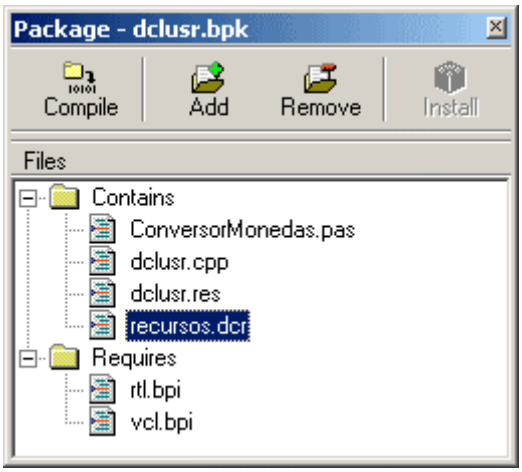
Una vez que tenemos el icono, lanzaremos el Editor de Imágenes de Delphi (Tools - Image Editor) y crearemos un nuevo Recurso para el Componente, desde el menú “File - New - Resource File (.res)”. Acto seguido aparecerá en nuestra pantalla un árbol vacío, así que procederemos a crear nuestro primer recurso. Con el menú “Resource - New – Bitmap” aparecerá una ventana en la que indicaremos la nueva imagen que será de 24 pixel de ancho (Width) y el mismo tamaño para el alto (Height). Después marcaremos una profundidad de color “SuperVGA (256 colores)”



Nuestro árbol aparecerá tal y como se ve en la imagen de la izquierda. Haciendo doble clic sobre el recurso "Bitmap1", se abrirá una ventana para editar su contenido. En esta ventana podemos dibujar la imagen, con la herramientas típicas (el lápiz, bote de pintura, spray, etc.) o bien pegar una imagen que hayamos creado desde otro programa (desde el MSPaint hasta el Photoshop pueden valer).

Una vez que tenemos la imagen a nuestro gusto, cerraremos la ventana de edición, y sobre el recuro “Bitmap1” mostraremos el menú contextual (botón derecho) para seleccionar la opción “Rename”. A este recurso debemos llamarlo con el mismo nombre que nuestro componente, para que así Delphi sea que este icono debe utilizarse para mostrar un componente en la paleta. En nuestro caso, debe llamarse TCONVERSORMONEDAS. El propio editor de recursos pondrá el nombre completo en mayúsculas ya que estos recursos se identificar por un número o una cadena en mayúsculas.

Cuando tengamos el recursos bien nombrado, podemos salir y guardar el archivo resultante con extensión RES: “recursos.res” podría estar bien.



Ya tenemos nuestro recursos correctamente creado en su archivo RES. Ahora iremos a Delphi y abriremos el paquete que contiene el componente, a través de la opción de menú “Component - Install Packages”, seleccionando el paquete en cuestión (Borland User Components) y haciendo clic sobre el botón “Edit”. Una vez hecho esto, tendremos el paquete abierto, viendo el contenido del mismo. Debemos añadir al paquete nuestro nuevo archivo de recursos. Para ello simplemente hacemos clic sobre el botón “Add”, y escribimos la ruta del archivo de recursos en el campo de edición “Unit file name”, o bien hacemos clic sobre “Browse...” para buscarlo manualmente. Cuando hayamos terminado, pulsaremos sobre "OK" y veremos que la ventana de edición del paquete tiene el aspecto de la imagen de la derecha. Lo último que queda por hacer es una compilación completa del paquete, a través de la opción de menú “Project - Build NombreDelPaquete”.

Si vamos a la paleta de componentes, veremos que ha cambiado el icono, como el de la imagen del al lado, para dejar bien claro que nuestro componente es un conversor de monedas.

Los ejemplos

Podéis descargar la segunda versión del componente de conversión de divisas y ver los cambios que hemos hecho para añadir los eventos:

http://users.servicios.retecal.es/sapivi/src/ConversorMonedas2.pas

Este artículo apareció publicado por primera vez en el número 1 de la revista Todo Programación, editada por Studio Press, S.L. y se reproduce aquí con la debida autorización.