Capítulo 13
XML

13.1 CONCEPTO Y UTILIDAD DE LA DOCUMENTACIÓN XML

La documentación de los tipos de datos creados siempre ha sido una de las tareas más pesadas y aburridas a las que un programador se ha tenido que enfrentar durante un proyecto, lo que ha hecho que muchas veces se escriba de manera descuidada y poco concisa o que incluso que no se escriba en absoluto. Sin embargo, escribirla es una tarea muy importante sobre todo en un enfoque de programación orientada a componentes en tanto que los componentes desarrollados muchas veces van a reutilizados por otros. E incluso para el propio creador del componente puede resultar de inestimable ayuda si en el futuro tiene que modificarlo o usarlo y no recuerda exactamente cómo lo implementó.

Para facilitar la pesada tarea de escribir la documentación, el compilador de C# es capaz de generarla automáticamente a partir de los comentarios que el programador escriba en los ficheros de código fuente.

El hecho de que la documentación se genere a partir de los fuentes permite evitar que se tenga que trabajar con dos tipos de documentos por separado (fuentes y documentación) que deban actualizarse simultáneamente para evitar inconsistencias entre ellos derivadas de que evolucionen de manera separada ya sea por pereza o por error.

El compilador genera la documentación en XML con la idea de que sea fácilmente legible para cualquier aplicación. Para facilitar su legibilidad a humanos bastaría añadirle una hoja de estilo XSL o usar alguna aplicación específica encargada de leerla y mostrarla de una forma más cómoda.

Por otra parte, Visual Studio 2005 incorpora una nueva función que permite configurar propiedades de aplicación (*application settings*) o de usuario (*user settings*) guardándolas en un archivo XML.

En nuestro caso la empleamos para guardar la configuración del puerto serie y los parámetros de la última orden X-10 seleccionada por el usuario. De este modo, en el siguiente arranque, el programa muestra en el formulario los últimos parámetros seleccionados por el usuario.

Veremos a continuación una introducción al lenguaje XML y posteriormente la creación del documento XML con las propiedades seleccionadas por el usuario.

13.2 INTRODUCCIÓN A XML

XML (Extensible Markup Language) es un **metalenguaje de etiquetas**, lo que significa que es un lenguaje que se utiliza para definir lenguajes de etiquetas. A cada lenguaje creado con XML se le denomina **vocabulario XML**, y la documentación generada por el compilador de C# está escrita en un vocabulario de este tipo.

Los comentarios a partir de los que el compilador generará la documentación han de escribirse en XML, por lo que han de respetar las siguientes reglas comunes a todo documento XML bien formado:

- La información ha de incluirse dentro de **etiquetas**, que son estructuras de la forma:

  <<*etiqueta***>>** <*contenido*> **</**<*etiqueta***>**

  En <etiqueta> se indica cuál es el nombre de la etiqueta a usar. Por ejemplo:

  <*EtiquetaEjemplo*> *Esto es una etiqueta de ejemplo* </*EtiquetaEjemplo*>

  Como <contenido> de una etiqueta puede incluirse tanto texto plano (es el caso del ejemplo) como otras etiquetas. Lo que es importante es que toda etiqueta cuyo uso comience dentro de otra también ha de terminar dentro de ella. O sea, no es válido:

  <*Etiqueta1*> <*Etiqueta2*> </*Etiqueta1*></*Eetiqueta2*>

  Pero lo que sí sería válido es:

  <*Etiqueta1*> <*Etiqueta2*> </*Etiqueta2*></*Etiqueta1*>

  También es posible mezclar texto y otras etiquetas en <contenido>. Por ejemplo:

  <*Etiqueta1*> *Hola* <*Etiqueta2*> *a* </*Etiqueta2*> *todos* </*Etiqueta1*>

- XML es un lenguaje sensible a mayúsculas, por lo que si una etiqueta se abre con una cierta capitalización, a la hora de cerrarla habrá que usar exactamente la misma.

- Es posible usar la siguiente sintaxis abreviada para escribir etiquetas sin <contenido>:

  <<*etiqueta***>/>**

  Por ejemplo:

  ```
  <<EtiquetaSinContenidoDeEjemplo>/>
  ```

- En realidad en la <etiqueta> inicial no tiene porqué indicarse sólo un identificador que sirva de nombre para la etiqueta usada, sino que también pueden indicarse **atributos** que permitan configurar su significado. Estos atributos se escriben de la forma <nombreAtributo> = "<valor>" y separados mediante espacios. Por ejemplo:

  ```
  <EtiquetaConAtributo AtributoEjemplo="valor1" >
     Etiqueta de ejemplo que incluye un atributo
  </EtiquetaConAtributo>

  <EtiquetaSinContenidoYConAtributo AtributoEjemplo="valor2" />
  ```

- Sólo puede utilizarse caracteres ASCII, y los caracteres no ASCII (acentos, eñes, ...) o caracteres con algún significado especial en XML han de ser sustituidos por secuencias de escape de la forma &#<códigoUnicode>. Para los caracteres más habituales también se han definido las siguientes secuencias de escape especiales:

Carácter	Secuencia de escape Unicode	Secuencia de escape especial
<	&#60;	&lt;
>	&#62;	&gt;
&	&#38;	&amp;
'	&#39;	&apos;
"	&#34;	&quot;

TABLA XXVIII: Secuencias de espace XML de uso frecuente

13.3 DOCUMENTO XML CON LAS PROPIEDADES DEL USUARIO

Dentro de Visual Studio es posible organizar nuestro trabajo en Soluciones y Proyectos. Una solución es un agrupador de proyectos que representa nuestro sistema en su totalidad. Cuando realizamos nuestros sistemas, éstos están divididos en componentes de código, interfaces de usuario (controles de usuario, ejecutables, páginas Web), componentes de configuración, componentes de acceso a datos, etc. Cada uno de ellos tiene una funcionalidad específica y el conjunto forma nuestro sistema. Dentro del Visual Studio, podemos trabajar con cada uno de ellos, sin necesidad de abrir un entorno diferente. Eso es lo que hace la solución: los agrupa y nos permite trabajar con todos, como si fueran una unidad.

Una vez seleccionado el proyecto con el que queremos trabajar, antes de empezar a escribir código, una buena práctica es establecer ciertos parámetros del proyecto para que las clases y todo lo que escribamos tengan un orden coherente.

Para ello podemos acceder a las propiedades del proyecto, que se encuentran en la ventana de Propiedades, como su propio nombre indica. Para acceder a dicha ventana, en el Solution Explorer, haremos doble clic en el namespace Properties o clic con el botón derecho del ratón en la misma carpeta y, seleccionaremos la opción **Open** del menú contextual.

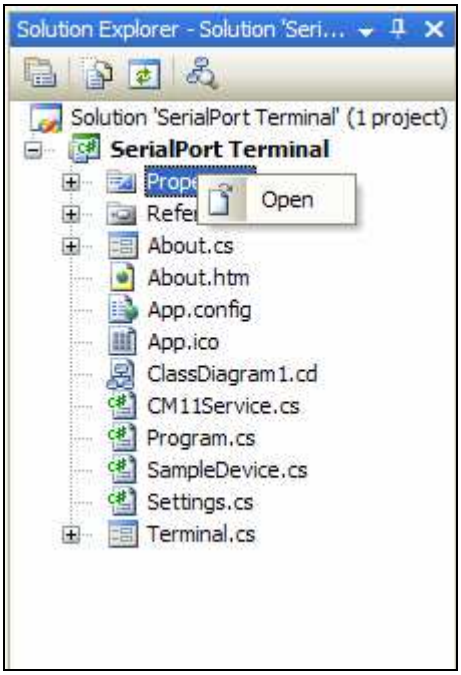

Figura 97: Propiedades del namespace

Una vez abierta, la ventana de Propiedades aparece en el área de trabajo. Esta ventana tiene las solapas a la izquierda y presenta las diferentes opciones de configuración del proyecto, como vemos en la figura inferior.

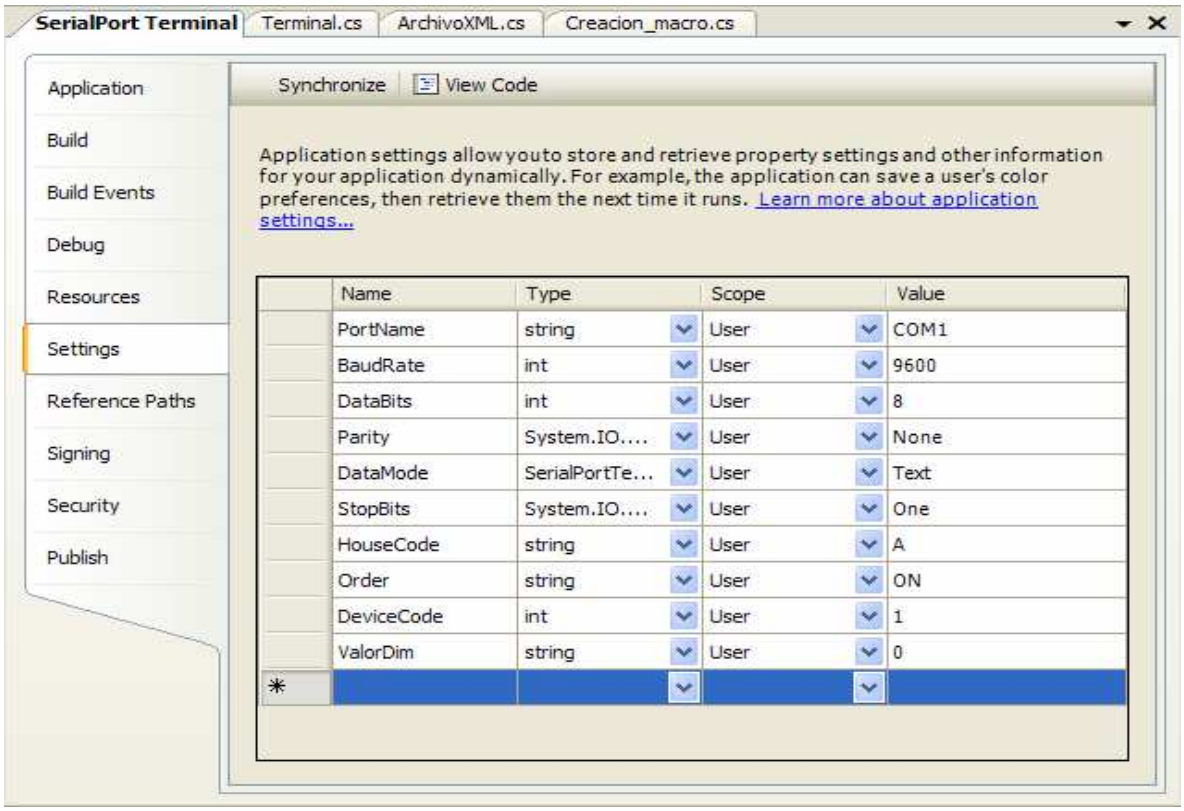

Figura 98: Propiedades en Visual Studio

La pestaña de nuestro interés es la pestaña 'Settings', donde podemos añadir el nombre, tipo, visibilidad (application o user) y valor asignado del parámetro deseado. Estos valores permanecen siempre como valores por defecto y, si se desea recuperarlos en tiempo de diseño, sólo hay que pulsar el botón Sincronizar.

En nuestro caso, como se puede observar en la figura, se han creado 9 propiedades que guardan los parámetros de configuración del puerto serie y la última orden X-10 seleccionada por el usuario.

```
<userSettings>
    <SerialPortTerminal.Properties.Settings>
      <setting name="PortName" serializeAs="String">
        <value>COM1</value>
      </setting>
      <setting name="BaudRate" serializeAs="String">
        <value>9600</value>
      </setting>
      <setting name="DataBits" serializeAs="String">
        <value>8</value>
      </setting>
      <setting name="Parity" serializeAs="String">
```

```
            <value>None</value>
        </setting>
```

Figura 99: Documento XML con la configuración del puerto serie

```
<setting name="HouseCode" serializeAs="String">
    <value>A</value>
</setting>
<setting name="Order" serializeAs="String">
    <value>ON</value>
</setting>
<setting name="DeviceCode" serializeAs="String">
    <value>1</value>
</setting>
<setting name="ValorDim" serializeAs="String">
    <value>0</value>
```

Figura 100: Documento XML con los valores de la orden seleccionada

A estos valores se puede acceder mediante la clase Settings alojada en SerialPortTerminal.Properties y es posible asignar su valor a la propiedad de la aplicación que se desea configurar. Así obtendremos los datos de la configuración guardada al cerrar el formulario por última vez.

```
cmbParity.Text = Settings.Default.Parity.ToString();
cmbStopBits.Text = Settings.Default.StopBits.ToString();
cmbDataBits.Text = Settings.Default.DataBits.ToString();
cmbParity.Text = Settings.Default.Parity.ToString();
```

Y también se puede modificar su valor y guardarlo al salir de la aplicación, usando 'Settings.Default.Save':

```
Settings.Default.BaudRate = int.Parse(cmbBaudRate.Text);
Settings.Default.DataBits = int.Parse(cmbDataBits.Text);
Settings.Default.DataMode = CurrentDataMode;

Settings.Default.Save();
```

13.4 DOCUMENTO XML ASOCIADO A LA MACRO

Después del diseño de una macro, la aplicación crea automáticamente dos documentos XML de gran utilidad para el usuario, denominados "IniciadoresXML.xml" y "TimersXML.xml". Estos documentos recogen, en lenguaje XML, las macros diseñadas por el usuario, de modo que en cualquier

instante podemos saber los timers programados o los iniciadores, y las órdenes asociadas a cada uno de ellos.

A continuación se recoge un fragmento de cada archivo, a modo de ejemplo.

```
<?xml version="1.0" encoding="utf-8" ?>
- <INICIADORES>
  - <NuevoIniciador>
      <Nombre_del_iniciador>A4 ON</Nombre_del_iniciador>
    - <Órdenes_asociadas_al_iniciador>
        <Orden>A3 ON offset 0</Orden>
        <Orden>A3 OFF offset 1</Orden>
      </Órdenes_asociadas_al_iniciador>
    </NuevoIniciador>
  </INICIADORES>
```

Figura 101: XML de Iniciador de macro

```
<?xml version="1.0" encoding="utf-8" ?>
- <TIMERS>
  - <NuevoTimer>
    - <Fecha_y_hora_del_timer>
        <Días_en_que_se_ejecutará>sábado</Días_en_que_se_ejecutará>
        <Día_del_año_de_inicio>20/10/2007</Día_del_año_de_inicio>
        <Día_del_año_final>20/10/2007</Día_del_año_final>
        <Hora_del_evento_de_inicio>22</Hora_del_evento_de_inicio>
        <Minuto_del_evento_de_inicio>17</Minuto_del_evento_de_inicio>
        <Hora_del_evento_de_fin>22</Hora_del_evento_de_fin>
        <Minuto_del_evento_de_fin>19</Minuto_del_evento_de_fin>
      </Fecha_y_hora_del_timer>
    - <Órdenes_asociadas_al_timer>
      - <Órdenes_del_evento_de_inicio>
          <Orden>A3 ON offset 0</Orden>
        </Órdenes_del_evento_de_inicio>
      - <Órdenes_del_evento_de_fin>
          <Orden>A3 OFF offset 1</Orden>
        </Órdenes_del_evento_de_fin>
      </Órdenes_asociadas_al_timer>
    </NuevoTimer>
  </TIMERS>
```

Figura 102: XML de Timer