Curso de C++ Builder

4.1 Introducción

Para ser un buen programador en *C++ Builder* existen distintos aspectos que se deben conocer: El Entorno de desarrollo integrado de *C++ Builder* (el *IDE*).

Los componentes disponibles para crear aplicaciones Windows, y sobre éstos, sus propiedades más importantes, los métodos asociados y los eventos a los que pueden responder.

Un conocimiento genérico acerca de la biblioteca de clases visuales de Borland (la *VCL*), en definitiva, conocer a grandes rasgos la jerarquía de clases de la que forman parte los componentes

4.2. El IDE (Entorno de Desarrollo Integrado)

IDE es el acrónimo de ***I**ntegrated **D**evelopment **E**nvironment* o entorno de desarrollo integrado. *C++ Builder* es una aplicación Windows que proporciona un entorno de trabajo visual para construir aplicaciones Windows que integra distintos aspectos de la programación en un entorno unificado o integrado.

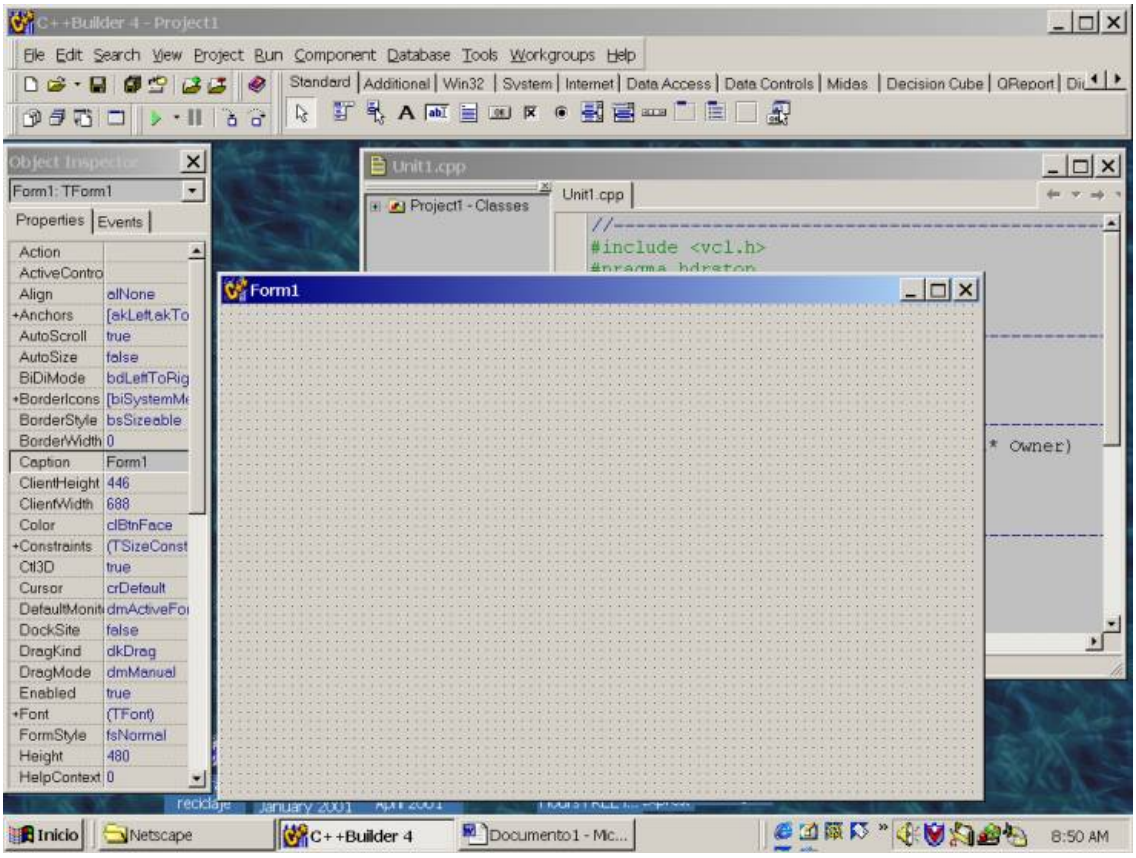
La integración y facilidad de manejo hace que sea una herramienta indispensable para el desarrollo rápido de aplicaciones o RAD (***R**apid **A**pplication **D**evelopment*). Guarda una gran similitud con el IDE de *Visual Basic*, aunque existen ciertas diferencias que veremos.

El IDE de *C++ Builder* es una aplicación Windows 95 y como tal cumple con los estándares de aspecto, diseño y comportamiento que aconseja Microsoft a los desarrolladores de aplicaciones. En consecuencia, cualquiera que esté familiarizado con el manejo a nivel de usuario de Windows 95 no le supondrá ningún esfuerzo manejarlo con soltura.

4.2.1. Una visión general del IDE de C++ Builder.

El entorno de desarrollo se divide, básicamente, en tres partes. Una serie de ventanas, que pueden estar visibles u ocultas, constituyen la base de *C++ Builder*. El aspecto de la aplicación al inicio de una sesión de trabajo es el mostrado en la figura 4.1.

Figura 4.1. Aspecto del *C++ Builder* al inicio de una sesión.



En la parte superior se coloca la **ventana principal**, que contiene el menú principal, la barra de herramientas (a la izquierda) y la paleta de componentes (a la derecha). Debajo de la ventana principal, y a la izquierda se coloca el **inspector de objetos**. A la dercha del inspector de objetos está el área de trabajo de *C++ Builder*, que inicialmente muestra el **diseñador de formularios**, y escondido u oculto parcialmente tras éste aparece el **editor de código**. Veamos a grandes rasgos la misión de cada uno de ellos.

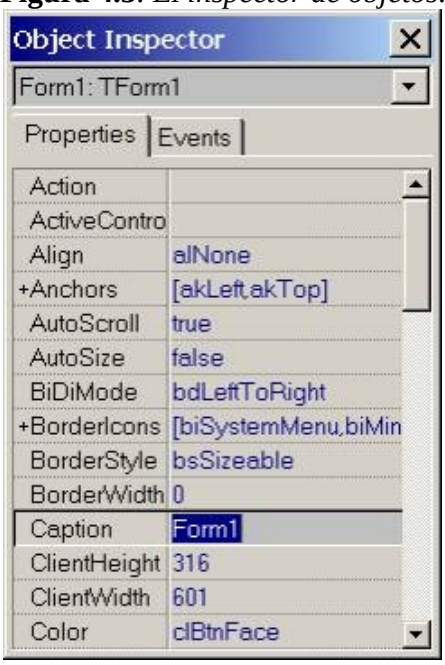
Ventana principal.

En la ventana principal se ubican el menu principal, la barra de herramientas y la paleta de componentes (figura 4.2).

Figura 4.2. Ventana principal de *C++ Builder*.

Menú principal. Permite el acceso a todas las operaciones y posibilita la configuración del programa.
Barra de herramientas. Permite un acceso rápido a las operaciones que se realizan más frecuentemente.
Paleta de componentes. Agrupa a los componentes que pueden incluirse en las aplicaciones.
Inspector de objetos. Para cambiar las *propiedades* de los objetos que forman la aplicación y seleccionar los *eventos* a los que debe responder la aplicación.

Figura 4.3. El *inspector de objetos*.

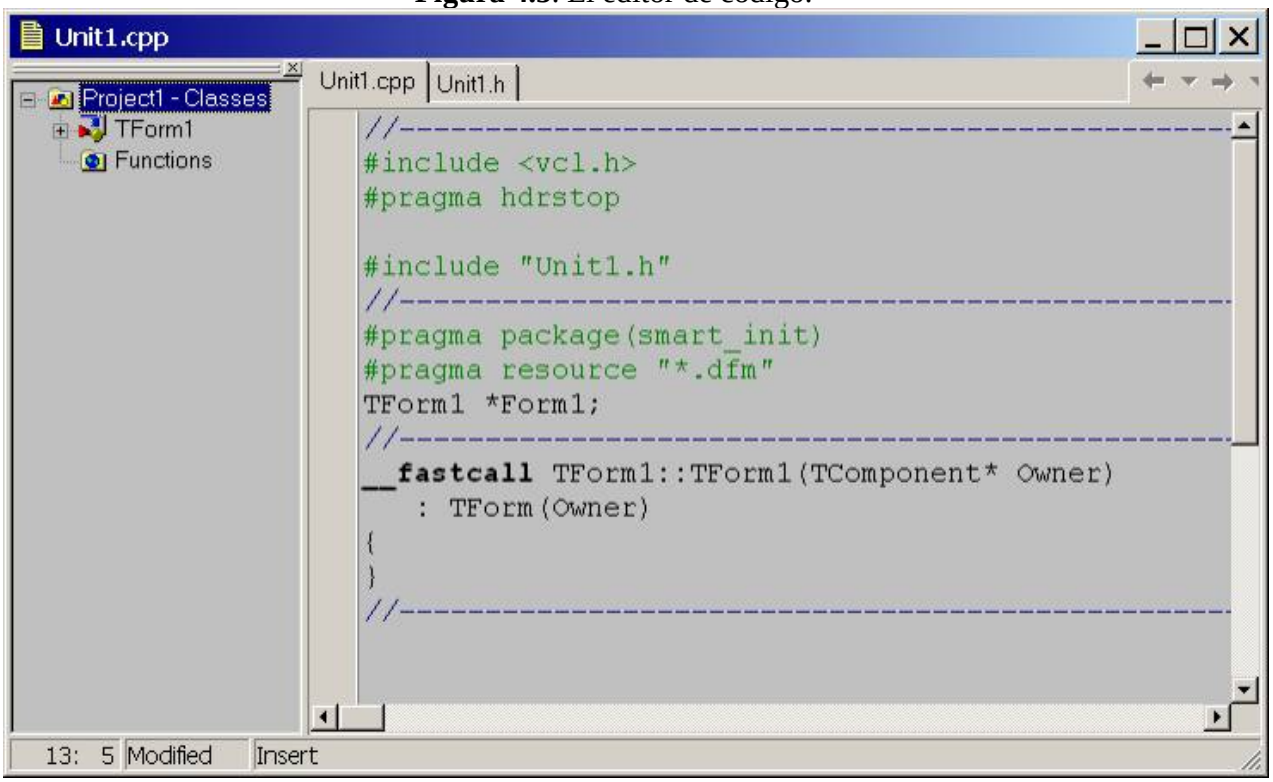


Diseñador de formularios. Es una ventana cuadriculada sobre el que se disponen los componentes para diseñar las ventanas que formarán la aplicación.

Figura 4.4. El *diseñador de formularios*.

Editor de código. Un típico editor de texto multiventana para ver y editar el código de la aplicación. Está perfectamente integrado con el inspector de objetos y el diseñador de formularios.

Figura 4.5. El editor de código.



Existen otras partes del entorno, que ya iremos comentando conforme vayamos profundizando en el curso. A continuación trataremos con más detalle las partes que hemos enumerado anteriormente.

4.2.2. Un estudio detallado del IDE de *C++ Builder*.

Menú Principal

Se puede decir que es la ventana principal del IDE de *C++ Builder* y siempre está visible. En él podemos encontrar todas las operaciones disponibles.

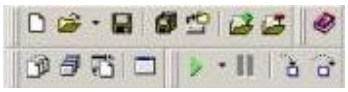
Figura 4.6. El menú principal.

Tiene los menús comunes a cualquier aplicación Windows: *File*, *Edit*, *Search*, *Help*..., además de otros que ya iremos estudiando a lo largo del curso. En el título del menú principal aparece el nombre de la aplicación (*C++ Builder*) y el nombre del proyecto/grupo de proyectos con el que actualmente se está trabajando. Por defecto, asigan el nombre Project1 al proyecto con el que se

va a trabajar, aunque éste deberá cambiarse, lógicamente. Como veremos, el concepto de *proyecto* es fundamental en *C++ Builder* ya que es el mecanismo de organizar sensatamente todos los ficheros (formularios, código fuente, recursos, etc.) asociados a una aplicación.

Barra de Herramientas La barra de herramientas tiene como objeto acelerar las operaciones más comunes del menú principal. Si nos acostumbramos a utilizarla agilizaremos el uso del entorno significativamente. Su aspecto habitual es el que mostramos en la figura 4.7.

Figura 4.7. La barra de herramientas.



Si pasamos el ratón sobre los iconos nos aparecerán unos *globos* o *cuadros de ayuda*, informándonos de cual es la funcionalidad de cada uno de ellos. La barra de herramientas se puede personalizar (al igual que casi todas las partes del entorno) pulsando con el botón derecho y seleccionando Properties.

Paleta de Componentes Los componentes constituyen los bloques básicos sobre la que se construyen aplicaciones Windows con *C++ Builder* basadas en la VCL (*Visual Component Library*) y son la base de la VCL.

Todos los componentes (visuales y no visuales) están accesible rápida y cómodamente gracias a la paleta de componentes. Podemos decir que es un gran almacén de componentes listos para ser incorporados a una aplicación, seleccionándolo y colocándolo en un formulario. Por conveniencia, todos los componentes disponibles se encuentran organizados en distintas páginas o carpetas según su funcionalidad. Cada pestaña de la paleta de componentes da acceso a un conjunto de iconos que representan a componentes que pueden usarse para diseñar la aplicación. Su aspecto es el que mostramos en la figura 4.8.

Figura 4.8. La paleta de componentes.

A diferencia con *Visual Basic*, todos los componentes de *C++ Builder* se encuentran cargados por defecto, aunque se le pueden añadir más mediante las opciones del menú de componentes (opción Component del menú principal). Cada componente tiene asignada una página, por lo que el usuario no puede elegir la ubicación de los componentes disponibles por defecto en la paleta de componentes, aunque puede modificar la disposición de las páginas y añadir nuevas páginas o componentes (Tools | Environment Options)

Para colocar un componente en un formulario se debe seleccionar en la paleta de componentes el botón que representa al componente y a continuación, pinchar sobre el formulario donde se desea colocalo. Una estrategia alternativa (y menos utilizada) es seleccionar el componente en la ventana *Components*, que se abre seleccionando View | Component List.

Diseñador de Formularios Como hemos indicado anteriormente, es una ventana cuadriculada quer sirve para diseñar las ventanas (o *formularios*) que formarán la aplicación (ver figura 2.4). Es una herramienta visual destinada a diseñar y presentar la interfaz de usuario (la apariencia externa) de una aplicación. Un formulario puede ser la ventana principal de un programa, un cuadro de diálogo o cualquier otra ventana.
Mediante el uso del ratón podemos colocar componentes de la paleta de componentes en el área de diseño. Lo único que tenemos que hacer es:

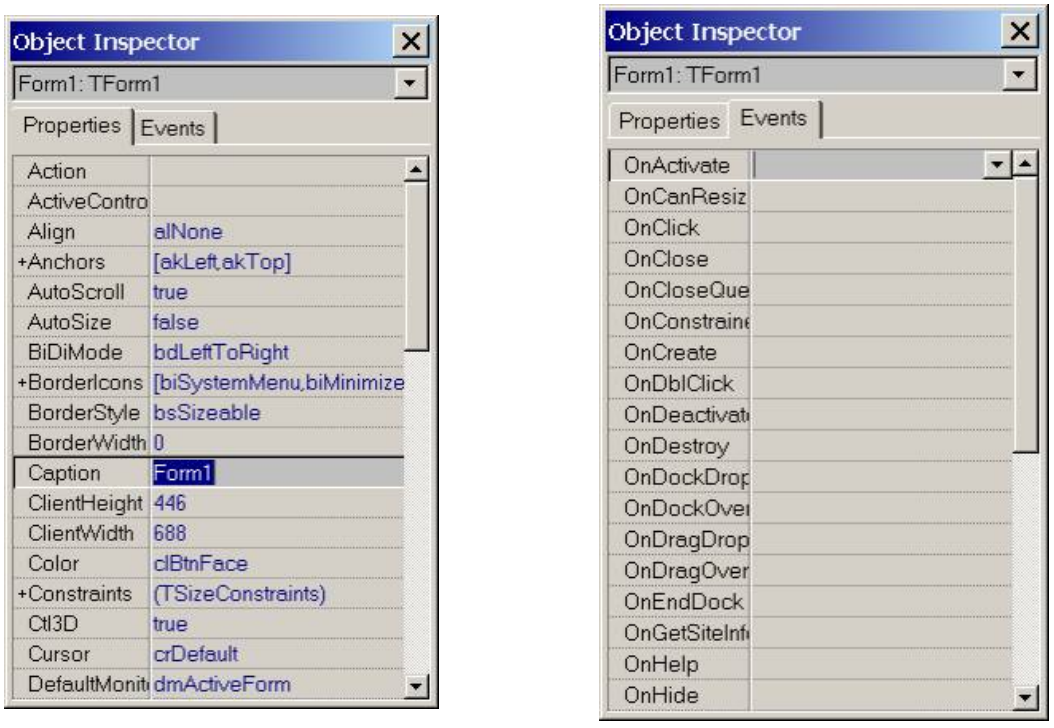
- Buscar el componente en la paleta de componentes.
- Pinchar sobre el componente para seleccionarlo.
- Pichar en el formulario: el componente ha quedado asociado al formulario.
- Arrastrarlo hasta su posición final.

La posición de un componente siempre se ajusta al punto más próximo de la rejilla para facilitar su ajuste (su alineación). La opciones relacionadas con la rejilla pueden modificarse en Tools | Environment Options.

Siempre hay un componente (o en su defecto, el propio formulario) *activo*. Visualmente se sabe cuál es porque aparece enmarcado con trazos discontínuos en el diseñador de formularios. Se puede activar cualquier componente pinchando (**un sólo click**) sobre él.

Inspector de Objetos Se trata, sin duda, de una de las herramientas más potentes y atractivas de *C++ Builder*. Su aspecto es el que mostramos anteriormente en la figura 4.3. Se trata de una ventana que contiene dos páginas: *Properties* y *Events*. En la figura 4.9 mostramos el aspecto de las dos páginas en cuestión para el formulario que aparece por defecto al iniciar *C++ Builder*.

Figura 4.9. Las dos páginas (*Properties* y *Events*) del inspector de objetos.



Cada componente tiene asociado un conjunto de propiedades y métodos y un conjunto de eventos a los que puede responder. Con el inspector de objetos podremos moldear los componentes de una aplicación según nuestras necesidades, en cuanto a su apariencia (*propiedades*) y funcionalidad (*eventos a los que puede responder*). En definitiva, podemos modificar las *propiedades* de los componentes y construir los *gestores de eventos* a los que éstos pueden responder.

En la parte superior se especifica el objeto activo (en la figura 4.9, el formulario, llamado Form1). Las propiedades del objeto activo aparecen en la página con la pestaña *Properties* y los eventos a los que puede responder en la página con la pestaña *Events*.

Para seleccionar un objeto desde el inspector de objetos se despliega la lista de objetos y se selecciona el objeto en la lista. También se puede seleccionar desde el diseñador de formularios pinchando (un solo click) sobre el objeto.

Si lo que se desea es modificar sus propiedades, se abre la carpeta de propiedades pinchando sobre la pestaña *Properties*. Si lo que se desea es asociarle un gestor de eventos, se abre la carpeta de gestores de eventos seleccionando la pestaña *Events*.

Por ejemplo, podríamos poner el texto Escuchar saludo en un botón colocado sobre un formulario (modificando la propiedad *Caption*) y escribir un código para reprodicir un fichero .WAV que contiene una grabación (escribiendo un gestor para el evento *OnClick*)

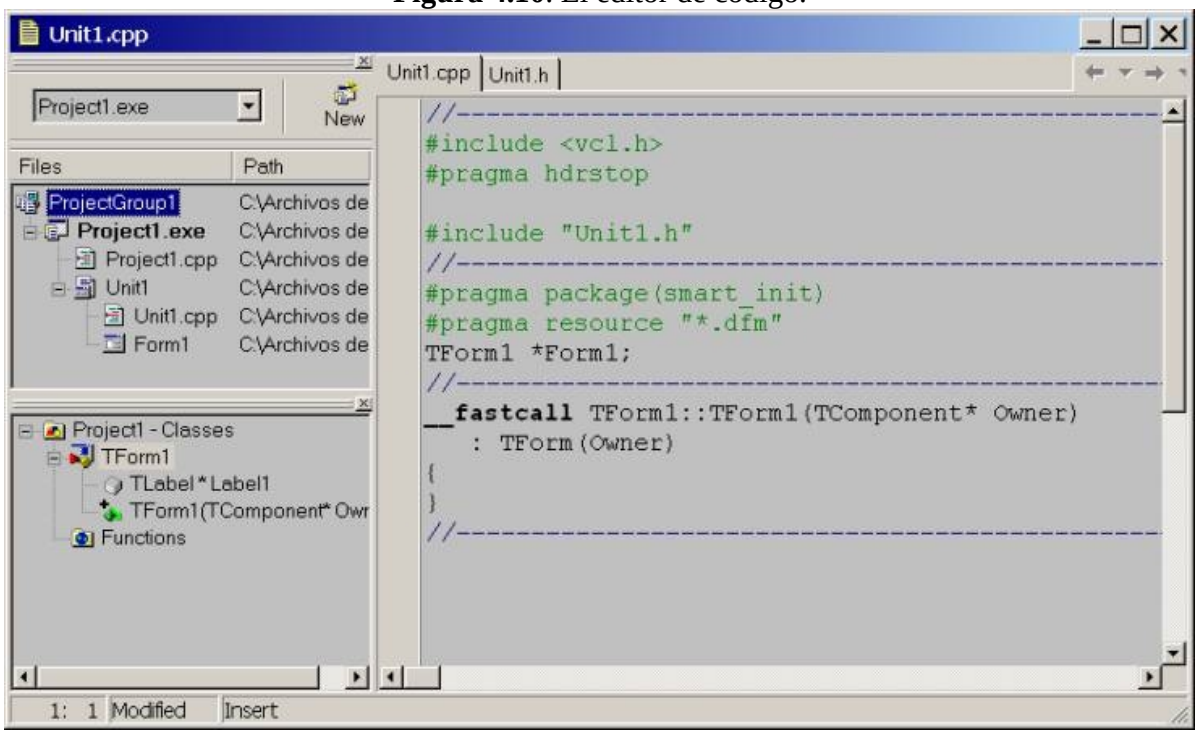
Editor de Código Permite editar el código de una aplicación de una forma cómoda. Admite coloreo simple de la sintaxis y distintos archivos abiertos simultáneamente.

En la ventana del editor pueden "pegarse" el *gestor de proyectos* y el *inspector de clases* (véase en la figura 4.10) aunque estas dos herramientas pueden aparecer también como ventanas separadas.

Inspector de clases: Es un navegador que muestra las *clases*, *objetos* y *métodos* asociados a la aplicación. Aparece por defecto asociada al editor (en la figura 4.10, en la parte inferior izquierda). Para abrir esta ventana: View | ClassExplorer.

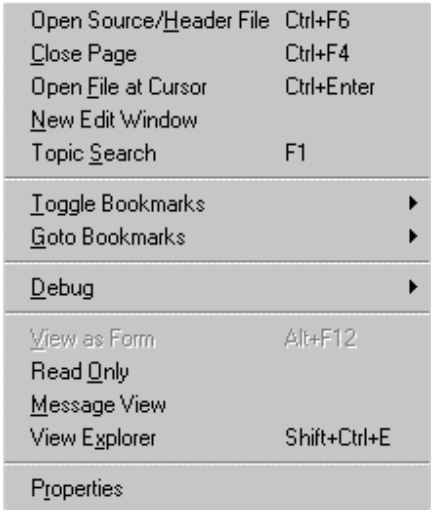
Administrador de proyectos: Es básicamente un navegador entre los diferentes ficheros que forman la aplicación. No aparece por defecto, y cuando se abre (View | Project Manager) se muestra como una ventana independiente. En la figura 4.10 se muestra asociada al editor, en la parte superior izquierda.

Figura 4.10. El editor de código.



Pulsando con el botón derecho aparece un menú contextual (figura 4.11) con algunas opciones bastante útiles.

Figura 4.11. El menú contextual del editor de código.



A diferencia con *Visual Basic*, *C++ Builder* nos permite el acceso a la totalidad del código fuente de la aplicación. Por esto es importante saber qué partes de ese código está mantenido automáticamente, y sobre todo ¡¡¡evitar modificarlo!!!.

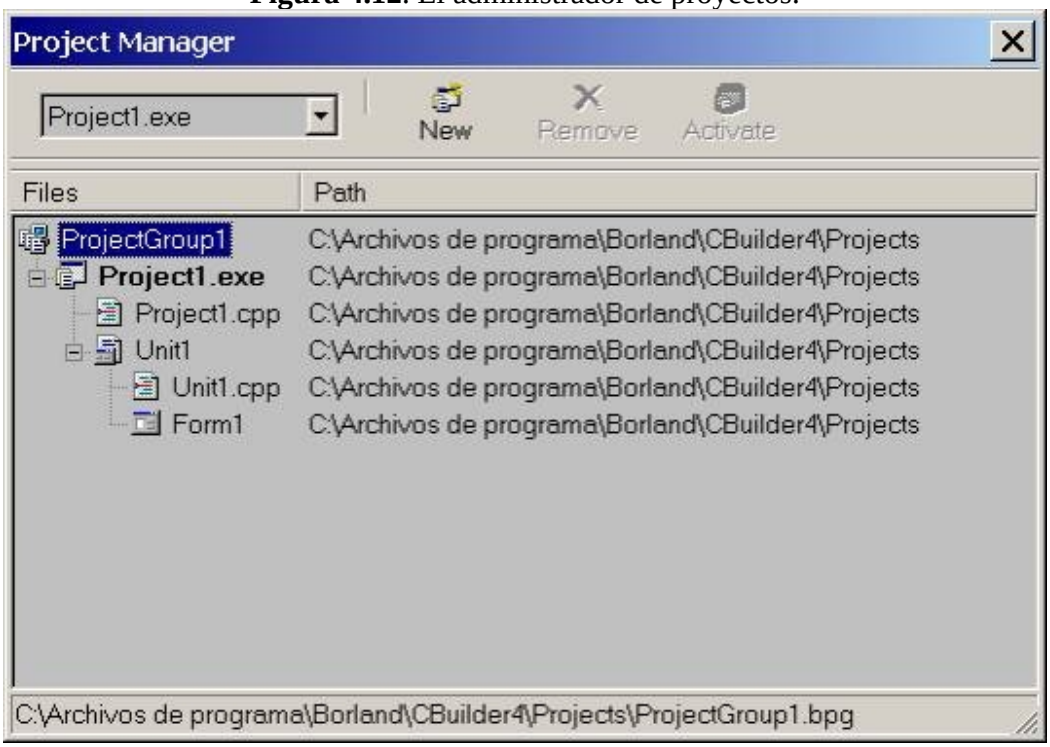
El editor de código está relacionado muy estrechamente con el inspector de objetos. Al hacer doble click en un evento, el IDE genera automáticamente el código para la función manejadora

para tal evento. No tenemos que preocuparnos de cuál es exactamente el prototipo que debemos de usar para la función que maneje el evento, ya que este se generará correctamente en la unidad de código asociada al *Form* actual.

Cuando queramos eliminar un manejador de evento que ha sido generado automáticamente es conveniente no borrar "a mano" la función. Lo mejor es borrar "el cuerpo" de dicha función (lo que el programador escribe) y dejar que el resto lo elimine *C++ Builder* (lo que ocurrirá cuando se guarde el archivo).

Administrador de Proyectos Un proyecto es un conjunto de archivos que trabajan en equipo para crear un archivo ejecutable independiente o una DLL. Un grupo de proyectos es un conjunto de proyectos.

Figura 4.12. El administrador de proyectos.



Los proyectos que componen un grupo de proyectos, y los archivos que componen cada uno de esos proyectos, es lo que presenta, en forma de árbol, el administrador de proyectos. Puede emplearse como navegador para seleccionar el módulo con el que se va a trabajar. Para visualizar el gestor de proyectos, seleccionar View | Project Manager. El gestor de proyectos puede "pegarse" al editor de código, arrastrándolo hasta colocarlo sobre éste último (ver figura 4.10).

En todo momento existe un único proyecto activo (en la figura 4.12, Project1), y será este el que se ejecute si elegimos la opción Run|Run.

Los ficheros de proyecto especifican todos los recursos necesarios (ficheros .cpp, .h, ficheros de descripción de formularios, etc.) que se necesitan para la construcción del ejecutable. Los

ficheros de proyecto tienen extensión .bpr y el ejecutable que se genera tiene el mismo nombre que el proyecto y la extensión .exe, lógicamente.

Todo proyecto en *C++ Builder* se compone, al menos, de un archivo de código que contiene la función principal (WinMain()). Su nombre es el nombre del proyecto, con la extensión .cpp (en la figura 4.12, Project1.cpp). Este fichero no está, habitualmente, visible, ya que no es necesario modificarlo. Puede abrirse en el editor de código con la opción Project|View Source. Cualquier aplicación típica tendrá al menos una ventana. Para cada ventana (en la figura 4.12, Form1) habrá un módulo, formado por una pareja de ficheros: un .cpp (en la figura 4.12, Unit1.cpp) y su correspondiente .h: en el fichero .cpp estarán los gestores de los eventos asociados a los componentes de esa ventana y en el .h (que no se modificará, normalmente) estará la declaración de los componentes de esa ventana.

Además del fichero que contiene la función principal, un proyecto puede tener asociados una serie de módulos adicionales en los cuales pueden incluirse funciones y clases de objetos, como en cualquier aplicación C++. Cada uno de estos módulos estará formado por una pareja de ficheros: un .cpp y su correspondiente .h. A un grupo de proyectos se le pueden añadir proyectos, archivos, formularios, módulos... nuevos o que ya existan. Hemos dicho que, en definitiva, el administrador de proyectos es únicamente un organizador de archivos. Veamos brevemente qué tipos de archivos pueden formar parte de un proyecto/grupo de proyectos, y que cuál es su cometido:

EXT	**Descripción**
BPR	Es el archivo makefile del proyecto. Define qué y cómo se debe compilar.
CPP	Archivos fuente de C++.
H	Archivos de cabecera de C++.
OBJ	Archivos objeto resultado de la compilación.
EXE	Es el programa ejecutable resultante.
TDS	Archivos temporales para la compilación incremental.
DFM	Archivos de descripción de formulario. Contiene los valores de las propiedades de cada componente. Aunque está en formato binario, puede verse como texto seleccionando View as text en el menú contextual que aparece al pulsar con el botón derecho del ratón cuando se está situado sobre el formulario.
RES	Un archivo de recursos.
DSK	Es el archivo que contiene la configuración del escritorio para un proyecto.
BPG	Es un archivo de grupo de proyectos. Describe qué proyectos conforman el grupo de proyectos.
HPP	Archivos de cabecera creados automáticamente por *C++ Builder*.

4.2.3. El sistema de ayuda

El sistema de ayuda será una de las herramientas que más útiles nos resultará en nuestro trabajo con *C++ Builder*. Especialmente la documentación de los componentes y clases predefinidas.

Saber programar con *C++ Builder* no significa dominar todos y cada uno de los aspectos del entorno, sino que más bien es conocer los principios en los que éste se basa, y los detalles concretos ya los buscaremos en la Ayuda cuando nos sean necesarios.

La ayuda es una ayuda estándar de *Windows* por lo que no entraremos en más detalles, sólo comentar que pulsando F1 obtendremos una ayuda contextual. Así, por ejemplo, en el caso de hacerlo en el editor de código, se nos ofrecerá la ayuda correspondiente a la palabra que se encuentre bajo el cursor (figuras 4.13, 4.14 y 4.15).

Figura 4.13. Ventana de ayuda que se despliega al pulsar F1 sobre la palabra clave Application

Figura 4.14. Ayuda detallada sobre Application variable (for standard applications).

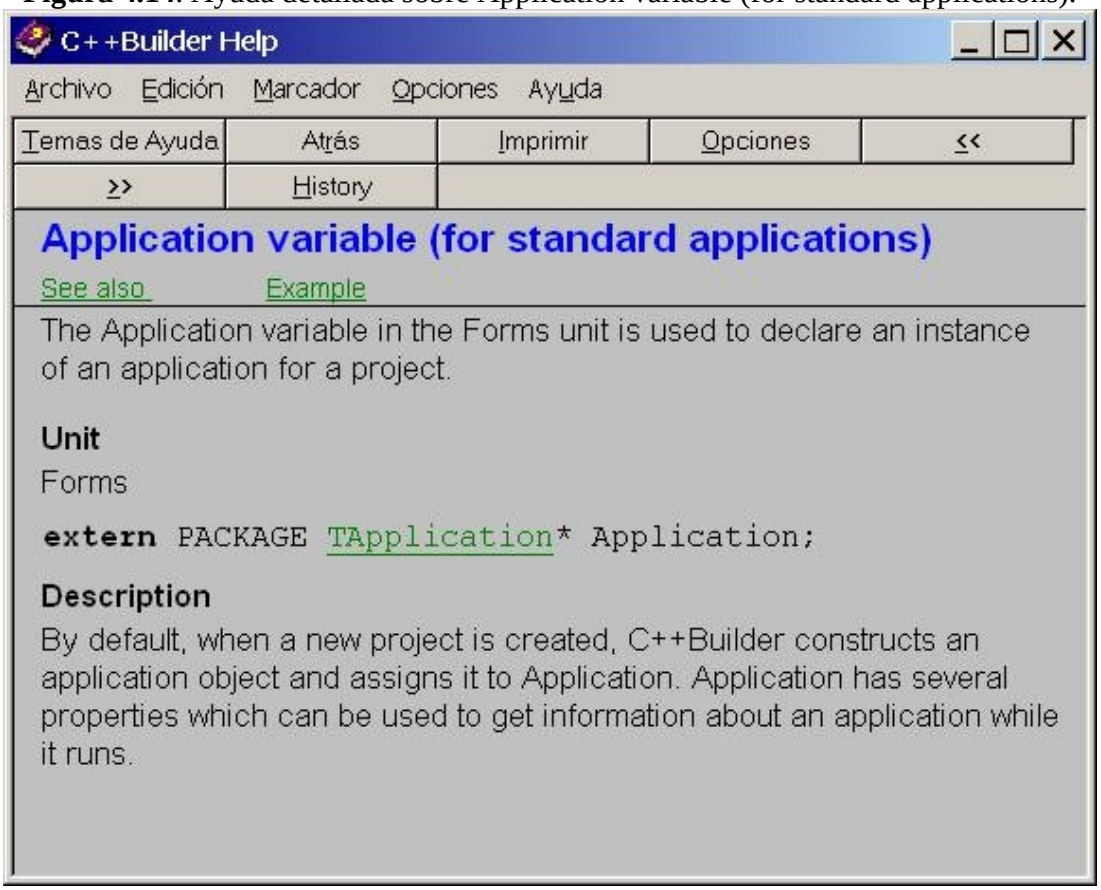
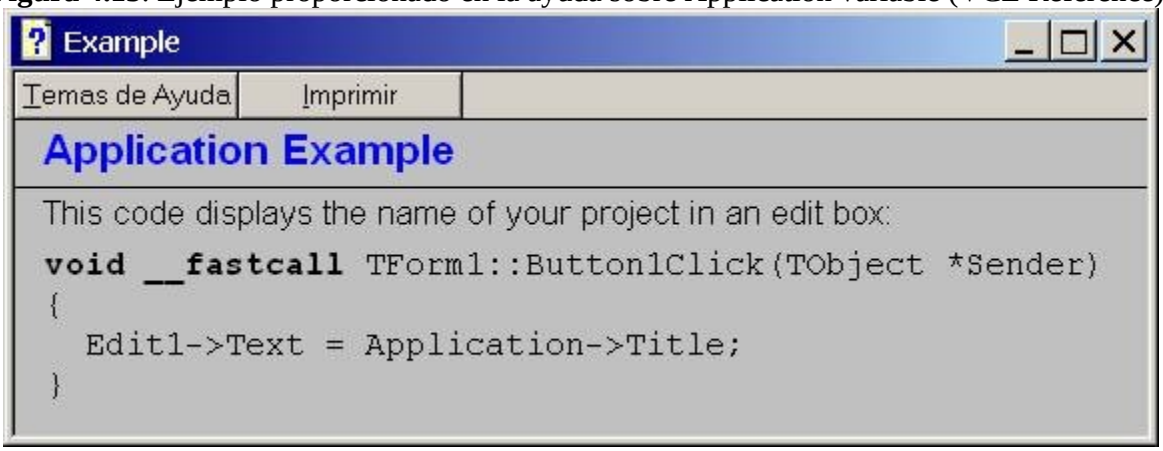


Figura 4.15. Ejemplo proporcionado en la ayuda sobre Application variable (VCL Reference).



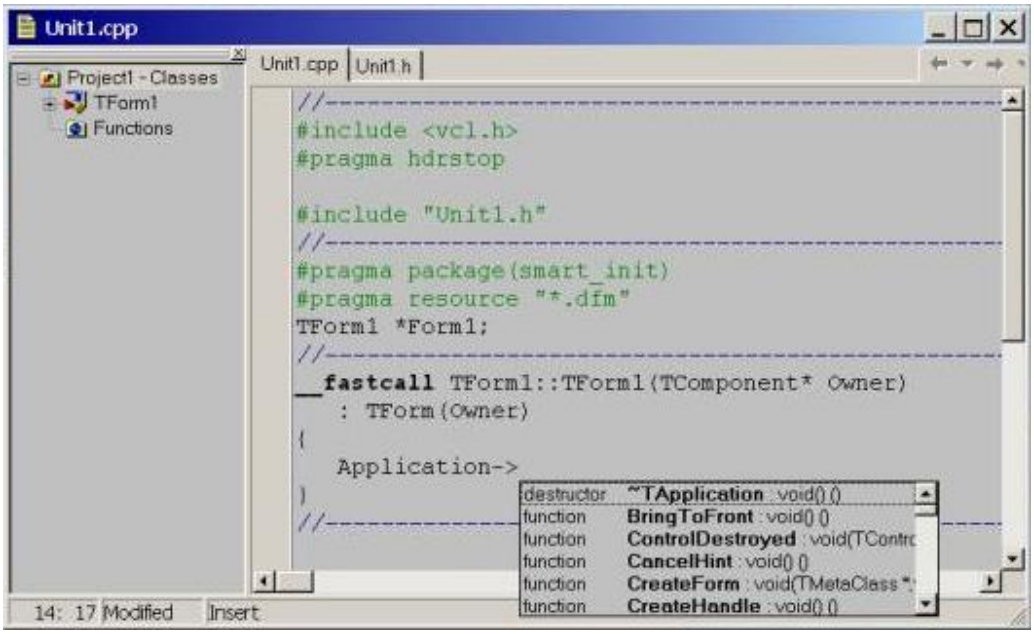
Pulsando F1 sobre cualquier opción de un menú se mostrará la ayuda asociada a esa opción (figura 4.16).

Figura 4.16. Ventana de ayuda mostrada al pulsar F1 estando seleccionada la opción View | Project Manager

C++ Builder proporciona algunas facilidades muy útiles a la hora de escribir código. Las más importantes son:

Completar código. Al escribir el nombre de una clase seguido del operador de acceso punto (.) o el nombre de un puntero que referencia a un objeto seguido del operador de acceso flecha (->) se muestra la lista de propiedades, métodos y eventos asociados a esa clase (figura 2.17).

Figura 4.17. Ayuda para completar código: Propiedades, métodos y eventos asociados a la clase TApplication.



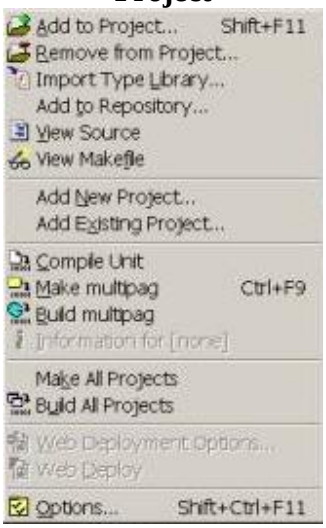
Patrones de código. Al pulsar Ctrl+J se muestran los patrones de código (también llamados "esqueletos") que pueden insertarse en el código de nuestro programa. Pueden crearse y guardarse nuevos patrones. Es posible configurar estas utilidades seleccionando Tools | Environment Options | Code Insight.

4.2.4. Compilación, ejecución y depuración de programas.

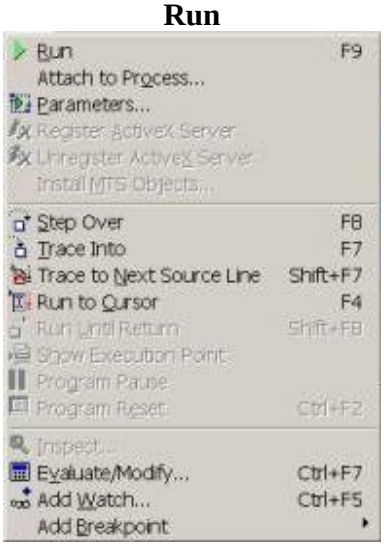
El objetivo final es la creación de un programa ejecutable correcto que funcione bajo Windows. El ejecutable se construye tomando como referencia los ficheros que forman el proyecto activo. Para esta tarea se utilizan los menús Project y Run (figura 4.18).

Figura 4.18. Submenús Project y Run del menú principal.

Project



Run



En la compilación se trata de la obtención del programa ejecutable (extensión .EXE). Las operaciones asociadas a este objetivo se encuentran en el menú que se despliega al seleccionar la opción Project del menú principal (figura 4.18), y las más importantes son:

- Compile Unit. Compila el modulo fuente activo (extensión .CPP) generando un fichero objeto (extensión .OBJ).
- Make. Genera el fichero ejecutable a partir de los ficheros objeto asociados al proyecto recompilando únicamente los módulos fuente que se hayan modificado desde la última vez que se compilaron.
- Build. Genera el fichero ejecutable a partir de los ficheros objeto asociados al proyecto recompilando todos los módulos fuente, aunque no se hayan modificado desde la última vez que se compilaron.

Si durante la compilación se detectaran errores, se mostraran en el editor de código y se puede acceder directamente a las líneas de código en las que se han detectado para facilitar su corrección.

Para ejecutar el programa basta con pinchar sobre el botón correspondiente de la barra de herramientas o seleccionar la opción Run | Run.

C++ Builder proporciona facilidades para la depuración de programas, seleccionables al desplegar el menú Run (figura 4.18). Las más importantes son:

Step Over F8. Ejecuta instrucción a instrucción el programa, pero ejecuta las llamadas a funciones como una instrucción más, sin mostrar la ejecución de las instrucciones de las funciones.

Trace Into F7. Ejecuta instrucción a instrucción el programa, incluidas las instrucciones de las funciones llamadas.

Run to Cursor F4. Ejecuta desde el principio del programa hasta la línea en la que está situada el cursor.

Para que la depuración sea más completa y versátil se incorporan las siguientes opciones en el mismo menú:

Evaluate/Modify Ctrl+F7. Evaluar expresiones que involucren a las variables y modificar los valores de las variables.

Add Watch Ctrl+F5. Visualizar el contenido de una variable permanentemente.

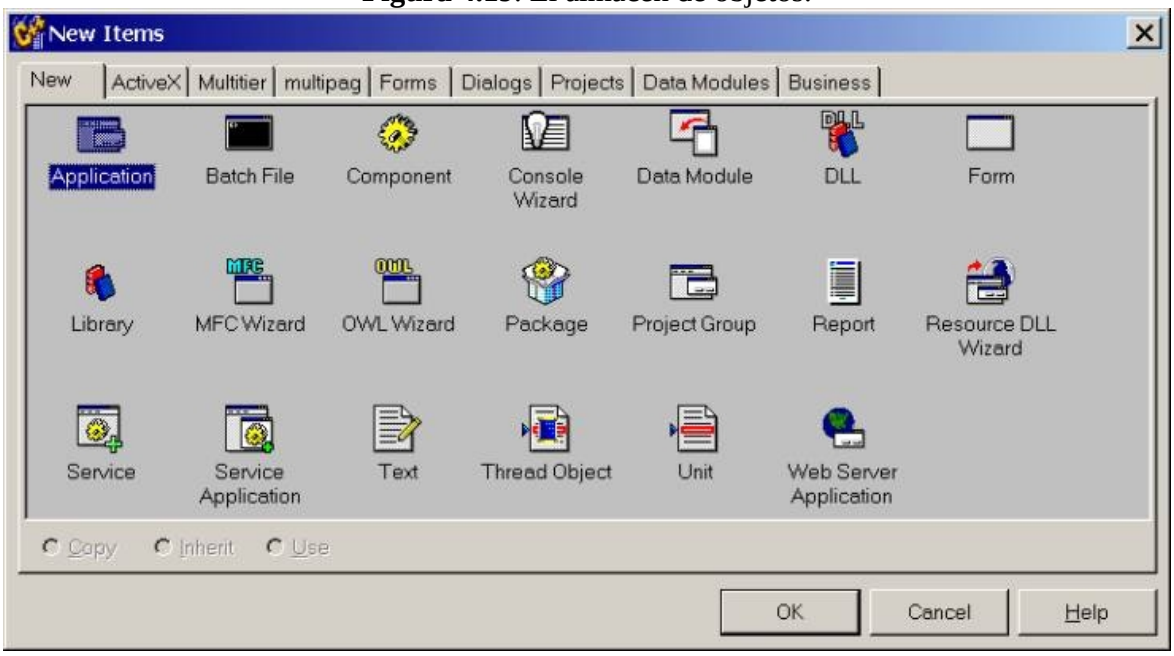
Add Breakpoint. Añade un punto de ruptura en la línea en la que está situado el cursor. de modo que cuando se ejecute el programa se detendrá su ejecución al llegar al siguiente punto de ruptura.

4.2.5. El almacén de objetos

Se accede al almacén seleccionando File | New.

Contiene formularios, cuadros de diálogo, módulos de datos, asistentes, DLLs, etc. que podemos utilizar para simplificar el desarrollo de aplicaciones. Todos ellos están prediseñados y pueden servirnos como punto de partida para nuestros propios diseños. Además se pueden incorporar nuevos elementos que nosotros desarrollemos, consiguiendo de esta forma reutilizar nuestros diseños.

Figura 4.19. El almacén de objetos.



En este curso utilizaremos con frecuencia el almacén para crear nuevos proyectos (File | New | Application), para crear aplicaciones de consola (File | New | Console Wizard), para añadir nuevos formularios (File | New | Form) o ficheros de código (File | New | Unit), para crear módulos de datos (File | New | Data Module), etc.

4.3. Componentes

Los componentes son unos elementos genéricos con una funcionalidad muy concreta, cuya única finalidad es la reutilización. Cada uno de ellos está destinado a realizar una tarea típica en una aplicación.

Un componente de la VCL es una *clase* que caracteriza a un *control* de Windows agregando *propiedades*, *métodos* y *gestores de eventos* a cada control.

La filosofía de los componentes en *C++ Builder* es exactamente la misma que en *Visual Basic*. Esto es tanto así que algunos componentes pueden utilizarse en ambos entornos (los componentes *ActiveX*).

4.3.1. Páginas de componentes

Como ya hemos comentado anteriormente los componentes se agrupan en la paleta de componentes en distintas páginas, a saber:

- ***Standard***: Incluye los componentes comunes y más habituales de los programas Windows.
- ***Additional***: Los componentes de esta página son controles especializados. propios de *C++ Builder*.
- ***Win32***: Componentes de cuadros de diálogo propios de *Windows 95/NT*.
- ***System***: Esta página incluye controles muy especializados para interacción con el sistema.
- ***Internet***: Componentes para distintos protocolos de acceso a Internet.
- ***Data Access*** y ***Data Controls***: Componentes especializados para acceso a bases de datos.
- ***Midas***: Esta página incluye componentes que permiten el desarrollo de aplicaciones multicapa con MIDAS.
- ***Decision Cube***: Incluye componentes para realizar análisis multidimensionales de datos con objeto de tomar decisiones.
- ***QReport***: Componentes para diseñar rápidamente informes y resúmenes.
- ***Dialogs***: Contiene cuadros de diálogo comunes en aplicaciones Windows listos para usar.
- ***Win 3.1***: Componentes propios de *Windows 3.1*, la mayoría de ellos se mantienen en *C++ Builder* sólo por compatibilidad con versiones anteriores.
- ***Samples***: Componentes de demostración (cómo hacer componentes personalizados).
- ***Active X***: Los componentes de esta página son objetos *Active X*, creados por otros desarrolladores puede que utilizando otros lenguajes de programación.

4.3.2. Propiedades, métodos y gestores de eventos

De un componente podemos destacar tres aspectos: sus propiedades, los métodos que puede ejecutar y los eventos a los que puede responder.

Propiedades. Las ***propiedades*** son los elementos del componente que configuran su aspecto y controlan su comportamiento.

Muchos componentes tienen propiedades en común. Por ejemplo, todos los componentes visuales tienen las propiedades *Top* y *Left* que controlan la posición del componente en el formulario, tanto en tiempo de diseño como en tiempo de ejecución.

Hemos visto como pueden establecerse y modificarse las propiedades de los componentes en tiempo de diseño utilizando el *inspector de objetos*. Las propiedades tienen, normalmente, un método de acceso asociado que se ejecuta al modificar la propiedad en tiempo de ejecución. Para modificar una propiedad basta (generalmente) con asignarle el nuevo valor. Al realizar un cambio, la VCL invoca el método de acceso a la propiedad. En el caso concreto de la propiedad *Left*, la VCL dibuja de nuevo todo el formulario en la nueva ubicación.

A modo de resumen, las propiedades tienen dos *especificadores de acceso* que se emplean al consultar o modificar el valor de una propiedad en tiempo de ejecución. En definitiva, hay un *especificador de lectura* y un *especificador de escritura*. Los especificadores de acceso asocian métodos de lectura o de escritura (*funciones*, en definitiva) con las propiedades. Al consultar o

modificar el valor de una propiedad se invocan, automáticamente, las funciones respectivas asociadas:

- Al asignar un nuevo valor a una propiedad se está accediendo al especificador de escritura. La VCL comprueba si existe un método de acceso asociado al especificador de escritura y si es así, lo ejecuta. Si no existe, simplemente asigna el nuevo valor a la propiedad.
- Al consultar el valor de una propiedad se está accediendo al especificador de lectura. En la mayoría de los casos el especificador de lectura no hace más que devolver el valor de la propiedad.

Métodos. Los ***métodos*** son *funciones* asociadas al componente que pueden invocarse para que el componente realice distintas acciones.

Por ejemplo, todos los componentes visuales tienen un método llamado *Show()* para mostralos y otro llamado *Hide()* para ocultarlos. En el ejemplo 3 invocamos al método *Terminate()* asociado a la aplicación que se estaba ejecutando (Objeto *Application*) para terminar su ejecución como respuesta a la pulsación del botón.

En C++ los métodos son *miembros* de una clase, al igual que las propiedades.

Eventos. Un ***evento*** es cualquier suceso que puede ocurrirle a un componente (movimiento del ratón, pulsación de algún botón del ratón, pulsación de una tecla del teclado, desplazamiento o redimensionamiento de una ventana, etc.) que pueden condicionar el comportamiento y apariencia del programa.

Cada componente poseerá una serie de eventos que puede recibir o generar. Se pueden tratar los eventos de un componente que necesitemos, y dejar que los demás sean tratados por defecto. Cuando se responde a un evento se dice que se está *manipulando* el evento. Los eventos se manejan mediante los *gestores* o *manipuladores de eventos*.

La forma de tratar un evento asociado a un componente en *C++ Builder* es sencilla: se activa el componente que va a responder al evento y utilizando el *Inspector de Objetos* (pestaña *Events*) se selecciona el evento al que debe responder y, cuando se active el editor de código, se escribe el código asociado al gestor del evento.

Se dice que Windows está *orientado a eventos*. Esto significa que cualquier programa está condicionado por lo eventos que ocurren en el entorno Windows. Un programa Windows está continuamente sondeando al sistema ante la ocurrencia de cualquier evento y de ocurrir, Windows avisa al programa enviándole un *mensaje*. Un programa Windows está *ocioso* la mayor parte del tiempo esperando a que ocurra algún evento.

4.3.3. Componentes visuales y no visuales

Se pueden establecer muchas clasificaciones para los componentes. Una de ellas es la de visuales o **controles**, frente a no visuales.

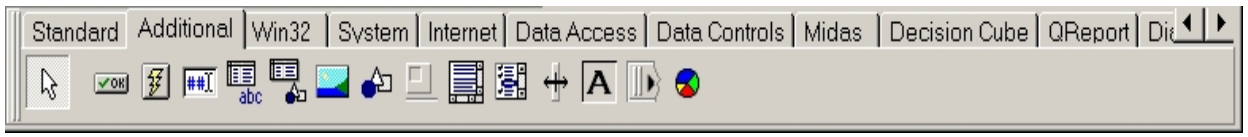
Un componente es visual cuando tiene una representación gráfica en tiempo de diseño y ejecución (botones, barras de scroll, cuadros de edición, etc.), y se dice no visual en caso contrario (temporizadores, cuadros de diálogo -no visibles en la fase de dieño-, etc). Por lo demás no existen más diferencias entre ellos, excepto, claro está, las derivadas de la visualización del componente.

Los componentes no visuales se pueden colocar en los formularios de la misma manera que los controles, aunque en este caso su posición es irrelevante.

4.4 La paleta de componentes

La paleta de componentes se estructura, por defecto, en 14 páginas. En la siguiente tabla agrupamos las páginas según nuestra conveniencia, siguiendo el criterio de utilidad en este curso.

Página *Additional* Iconos de los componentes agrupados en la página *Additional*

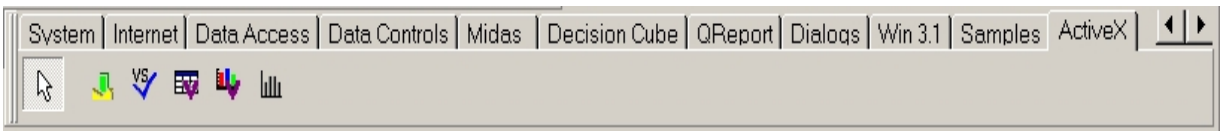


Los componentes de esta página son controles especializados:

BitBtn	Crea un botón que puede contener un gráfico (tipo "bitmap").
SpeedButton	Crea un botón que puede contener un gráfico pero no contiene texto. Estos botones se agrupan, normalmente, dentro de un panel para crear una barra de herramientas.
MaskEdit	Permite la introducción y edición de datos, como lo hace el componente *Edit*, salvo que proporciona la posibilidad de especificar formatos particulares, como códigos postales o números de teléfono.
StringGrid	Crea una rejilla que puede usarse para mostrar cadenas en filas y columnas.
DrawGrid	Crea una rejilla que puede usarse para mostrar datos en filas y columnas.
Image	Muestra un "bitmap" o un icono.
Shape	Dibuja figuras geométricas, como elipses, círculos, rectángulos, cuadrados o rectángulos y cuadrados con bordes redondeados.

Bevel	Crea líneas o cuadros con apariencia tridimensional y como si estuviera esculpida (alto o bajorrelieve).
ScrollBox	Crea un contenedor redimensionable que muestra barras de scroll cuando sea necesario.
CheckListBox	Muestra una lista de eleciones que está acompañada de una barra de scroll. Es muy parecido al componente *ListBox* salvo que cada elemento de la lista tiene asociado un *CheckBox*.
Splitter	Añade un divisor a un formulario entre dos componentes alineados que permite al usuario redimensionar los controles en tiempo de ejecución pinchando y arrastrando la línea de división.
StaticText	Es un componente de texto no editable, como el componente *Label*, sólo que *StaticText* tiene su propio gestor de ventana.
ControlBar	Un gestor para acompañar a barras de herramientas que se usa para poder mover este tipo de barras.
Chart	Un visualizador equivalene a *TTable*.

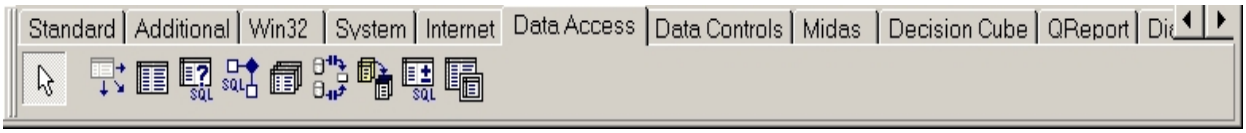
Página *Active X* Iconos de los componentes agrupados en la página *Active X*



Los componentes de esta página son objetos ActiveX. Son aplicaciones completas y portables creadas por otros desarrolladores.

Chartfx	Lets you create highly customized charts. Choose Properties to display a tabbed control panel that lets you define the values, appearance, and end-user functionality of the chart component.
VSSpell	Visual Speller, lets you customize a spelling checker.
F1Book	Formula One, lets you design a spreadsheet with its full-featured Designer.
VtChart	Lets you create true 3D charts.
Graph	Pinnacle Graph, lets you create 2D graphs.

Página *Data Access* Iconos de los componentes agrupados en la página *Data Access*



Esta página incluye componentes especialzados para acceso a bases de datos:

DataSource	Acts as a conduit between a dataset component such as TTable and data-aware components such as TDBGrid.
Table	Retrieves data from a physical database table via the BDE and supplies it to one or more data-aware components through a DataSource component. Conversely, it also sends data received from a component to a physical database via the BDE.
Query	Uses SQL statements to retrieve data from a physical database table via the BDE and supplies it to one or more data-aware components through a TDataSource component. Conversely, uses SQL statements to send data from a component to a physical database via the BDE.
StoredProc	Enables an application to access server stored procedures. Sends data received from a component to a physical database via the BDE.
Database	Sets up a persistent connection to a database, especially a remote database requiring a user login and password.
Session	Provides global control over a group of Database components. A default TSession component is automatically created for each C++Builder database application. You must use the TSession component only if you are creating a multithreaded database application. Each database thread requires its own session component.
BatchMove	Copies a table structure or its data. Can be used to move entire tables from one database format to another.
UpdateSQL	Lets you use cached updates support with read-only datasets. For example, you could use a TUpdateSQL component with a "canned" query to provide a way to update the underlying datasets, essentially giving you the ability to post updates to a read-only dataset. You associate a TUpdateSQL component with a dataset by setting the dataset's UpdateObject property. The dataset automatically uses the TUpdateSQL component when cached updates are applied.
NestedTable	Retrieves the data in a nested dataset field and supplies it to data-aware controls through a datasource component.

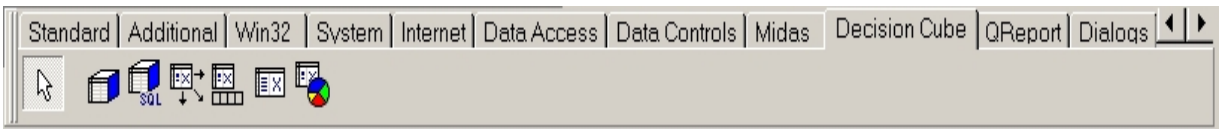
Página *Data Controls* Iconos de los componentes agrupados en la página *Data Controls*

Esta página incluye componentes especialzados para gestión de bases de datos:

DBGrid	Data-aware custom grid that enables viewing and editing data in a tabular form similar to a spreadsheet. Makes extensive use of TField properties (set in the Fields editor) to determine a column's visibility, display format, ordering, and so on.
DBNavigator	Data-aware navigation buttons that move a table's current record pointer forward or backward. The navigator can also place a table in Insert, Edit, or Browse state, post new or modified records, and retrieve updated data to refresh the display.
DBText	Data-aware label that displays a field value in the current record.
DBEdit	Data-aware edit box that displays or edits a field in the current record.
DBMemo	Data-aware memo box that displays or edits BLOB text in the current record.
DBImage	Data-aware image box that displays, cuts, or pastes bitmapped BLOB images to and from the current record.
DBListBox	Data-aware list box that displays a scrolling list of values from a column in a table.
DBComboBox	Data-aware combo box that displays or edits a scrolling list of values from a column in a table.
DBCheckBox	Data-aware check box that displays or edits a Boolean data field from the current record.
DBRadioGroup	Data-aware group of radio buttons that display or set column values.
DBLookupListBox	DBLookupListBox is a data-aware list box that derives its list of display items from either a Lookup field defined for a dataset or a secondary data source, data field, and key. In either case, a user is presented with a restricted list of choices from which to set a valid field value. When a user selects a list item, the corresponding field value is changed in the underlying dataset. To specify list box items using a lookup field, the dataset to which you link the control must

	already define a lookup field.
DBLookupComboBox	DBLookupComboBox is a data-aware combo box that derives its drop-down list of display items from either a lookup field defined for a dataset or a secondary data source, data field, and key. In either case, a user is presented with a restricted list of choices from which to set a valid field value. When a user selects a list item, the corresponding field value is changed in the underlying dataset. To specify combo box list items using a lookup field, the dataset to which you link the control must already define a lookup field.
DBRichEdit	A multiline edit control that can display and edit a rich text memo field in a dataset.
DBCtrlGrid	A DBCtrlGrid control displays multiple fields in multiple records in a tabular grid format. Each cell in the grid displays multiple fields from a single record.
DBChart	Place the component on a form and right-click it to display the third-party developer's Help topics.

Página *Decision Cube* Iconos de los componentes agrupados en la página *Decision Cube*

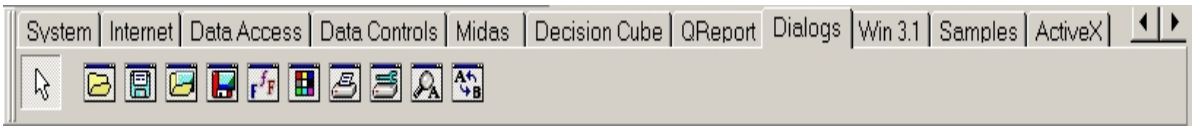


Esta página incluye componentes para realizar análisis multidimensionales de datos con objeto de tomar decisiones:

DecisionCube	A multidimensional data store. See Using decision cubes.
DecisionQuery	Specialized form of TQuery used to define the data in a decision cube. See Creating decision datasets with the Decision Query editor.
DecisionSource	Defines the current pivot state of a decision grid or a decision graph. See Using decision sources.
DecisionPivot	Use to open or close decision cube dimensions or fields by pressing buttons. See Using decision pivots.
DecisionGrid	Displays single and multidimensional data in table form. See Creating and Using decision grids.
DecisionGraph	Displays fields from a decision grid as a dynamic graph that changes when dimensions are modified. See Using decision

graphs.

Página *Dialogs* Iconos de los componentes agrupados en la página *Dialogs*



Esta página incluye las diferentes ventanas de diálogo comunes en aplicaciones Windows. Estas ventanas de diálogo proporcionan un interface consistente para realizar operaciones sobre ficheros como abrir, guardar e imprimir.

Una ventana de diálogo se abre al llamar al método Execute(). Este método devuelve un valor lógico:
true si el usuario elige *OK* en la ventana de diálogo,
false si el usuario elige *Cancel* o sale de la ventana de diálogo sin salvar los cambios.

Cada ventana de diálogo (excepto la asociada al componente *PrinterSetup*) tiene la propiedad *Options* que afecta a su apariencia y comportamiento.

Una ventana de diálogo puede cerrarse desde un programa mediante el método CloseDialog(). Para modificar en tiempo de ejecución su posición, usar las propiedades *Handle*, *Left*, *Top* y *Position*.

OpenDialog	Muestra una ventana de diálogo común de Windows para seleccionar y abrir ficheros.
SaveDialog	Muestra una ventana de diálogo común de Windows para seleccionar y guardar ficheros.
OpenPictureDialog	Muestra una ventana de diálogo modal común de Windows para seleccionar y abrir ficheros gráphicos. Es idéntica a la ventana asociada a *OpenDialog* salvo que contiene una región para previsualización de imágenes.
SavePictureDialog	Muestra una ventana de diálogo modal común de Windows para seleccionar y guardar ficheros gráphicos. Es idéntica a la ventana asociada a *SaveDialog* salvo que contiene una región para previsualización de imágenes.
FontDialog	Muestra una ventana de diálogo común de Windows para especificar la familia, tamaño y estilo de letra.
ColorDialog	Muestra una ventana de diálogo común de Windows para especificar color.
PrintDialog	Muestra una ventana de diálogo común de Windows para especificar información de impresión.

PrinterSetupDialog	Muestra una ventana de diálogo común de Windows para configurar impresoras.
FindDialog	Muestra una ventana de diálogo común de Windows para especificar la cadena a buscar.
ReplaceDialog	Muestra una ventana de diálogo común de Windows para especificar la cadena a buscar y la cadena por la que se va a reemplazar.

Página *Internet* Iconos de los componentes agrupados en la página *Internet*

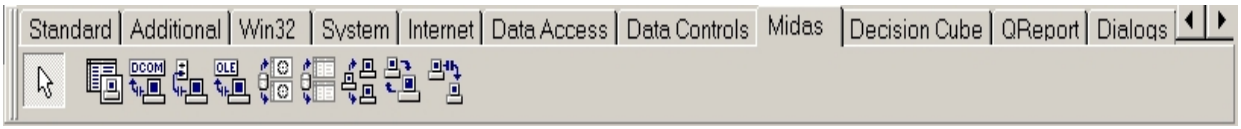
Los componentes de esta página ofrecen diferentes protocolos de acceso a Internet:

ClientSocket	Add to a form or data module to turn an application into a TCP/IP client. ClientSocket specifies a desired connection to a TCP/IP server, manages the open connection, and terminates the completed connection.
ServerSocket	Add to a form or data module to turn an application into a TCP/IP server. ServerSocket listens for requests for TCP/IP connections from other machines and establishes connections when requests are received.
WebDispatcher	Converts an ordinary data module to a Web module and enables the Web server application to respond to HTTP request messages.
PageProducer	Converts an HTML template into a string of HTML commands that can be interpreted by a client application such as a Web browser. The commands and HTML-transparent tags are replaced with customized content by the OnHTMLTag event.
QueryTableProducer	Assembles a sequence of HTML commands to generate a tabular display of the records from a TQuery object, which obtains its parameters from an HTTP request message.
DataSetTableProducer	Assembles a sequence of HTML commands to generate a tabular display of the records from a TDataSet object. This allows an application to create images of a dataset for an HTTP response message.
DataSetPageProducer	Converts an HTML template that contains field references into a string of HTML commands that can be interpreted by

	a client application such as a Web browser. Special HTML-transparent tags are replaced with field values.
NMDayTime	Gets the date and time from an internet/intranet daytime server.
NMEcho	Sends text to an internet echo server, and echoes it back to you.
NMFinger	Gets information about a user from an internet finger server, using the Finger protocol described in RFC 1288.
NMFTP	Implements file transfer protocol. Invisible ActiveX control provides easy access for Internet File Transfer Protocol (FTP) services for transferring files and data between a remote and local machine.
NMHTTP	Invisible ActiveX control implements the HTTP Protocol Client, allowing users to directly retrieve HTTP documents if no browsing or image processing is necessary.
NMMsg	Sends simple ASCII text messages across the internet or intranet using TCP/IP protocol.
NMMsgServ	Receives messages sent with the TNMMsg component.
NMNNTP	Invisible ActiveX Client Control allows applications to access Networking News Transfer Protocol (NNTP) news servers. It provides news reading and posting capabilities.
NMPOP3	Invisible ActiveX control that retrieves mail from UNIX or other servers supporting POP3 protocol.
NMUUProcessor	MIME encodes or UUEncodes files and decodes MIME-encoded or UUEncoded files.
NMSMTP	ActiveX control that gives applications access to SMTP mail servers and mail posting capabilities.
NMStrm	Sends streams to a stream server across the internet or an intranet.
NMStrmServ	Receives streams sent with the TNMStrm component.
NMTime	Gets the date and time from Internet time servers, as described in RFC 868.
NMUDP	Invisible WinSock ActiveX Control provides easy access to User Datagram Protocol (UDP) network services. It implements WinSock for both client and server and represents a communication point utilizing UDP network services. It can also be used to send and retrieve UDP data.

PowerSock	Serves as a base for creating controls for dealing with other protocols, or for creating custom protocols.
NMGeneralServer	Serves as a base class for developing multi-threaded internet servers, such as custom servers or servers that support RFC standards.
HTML	Invisible ActiveX control implements an HTML viewer, with or without automatic network retrieval of HTML documents, and provides parsing and layout of HTML data, as well as a scrollable view of the selected HTML page. The HTML component can also be used as a nonvisual HTML parser to analyze or process HTML documents.
NMURL	Decodes URL data into a readable string, and encodes standard strings into URL data format.

Página *MIDAS* Iconos de los componentes agrupados en la página *MIDAS*



Esta página incluye componentes que permiten el desarrollo de aplicaciones multicapa con MIDAS:

ClientDataSet	Implements a database-independent dataset that can be used independently in a single-tiered application, or to represent data received from a server in a multi-tiered database application. See Creating and using a client dataset
DCOMConnection	Establishes a DCOM connection to a remote server in a multi-tiered database application. See Connecting to the application server
SocketConnection	Establishes a TCP/IP connection to a remote server in a multi-tiered database application. See Connecting to the application server
OLEnterpriseConnection	Establishes an OLEnterprise connection to a remote server in a multi-tiered database application. See Connecting to the application server
DataSetProvider	Encodes data from a dataset into packets than can be sent to client applications and applies updates that are received from client applications to that dataset. See Creating a data provider for the application server and Providing from and resolving to a dataset

Provider	Encodes data from a dataset into packets than can be sent to client applications and applies updates that are received from client applications to a dataset or database server. See Creating a data provider for the application server.
SimpleObjectBroker	Locates a server for a connection component from a list of available application servers. See Brokering connections.
RemoteServer	(For backward compatibility only) Establishes a DCOM connection to a remote server in a multi-tiered application.
MIDASConnection	(For backward compatibility only) Establishes a DCOM, TCP/IP, or OLEnterprise connection to a remote server in a multi-tiered application.

Página *Qreport* Iconos de los componentes agrupados en la página *Qreport*

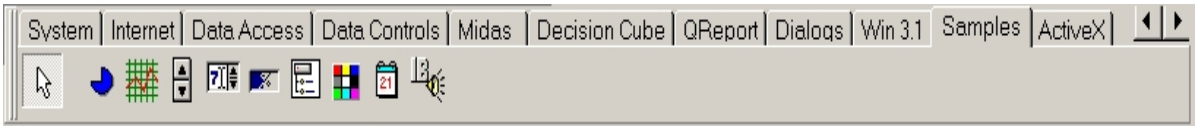
Esta página incluye componentes para diseñar rápidamente informes y resúmenes. Se pueden incorporar cabeceras y pies de páginas, resúmenes, agrupaciones con cabeceras y pies, etc. Estos informes pueden realizarse a partir de cualquier fuente de datos: *TTable*, *TQuery*, listas, matrices, etc. arrays. Puede usarse un visalizador previo para comprobar los resultados. Automáticamente se realizan cálculos similares a como se hacen en una hoja de cálculo.

QuickRep	The basic report form on which you build all your reports. It is a visual component that takes the shape of the currently selected paper size. Create reports by dropping bands and printable components on the TQuickRep component and connecting it to a dataset.
QRSubDetail	Links additional datasets into a report. Typically you would set up a master/detail relationship between table or query components and create a similar relationship with TQRSubDetail components.
QRStringsBand	Drops bands containing strings onto a report.
QRBand	Drop bands on a TQuickRep component and set the BandType property to tell how the band will behave during report generation.
QRChildBand	If you have bands with expanding components and want

	other components to be moved down accordingly you can create a child band and put the moving components on it. It's also useful if you have very long bands that span multiple pages.
QRGroup	Allows you to group bands together and provides control for headers, footers, and page breaks.
QRLabel	Prints static or other non-database text. Enter the text to be displayed in the Caption property. You can split text on multiple lines and even multiple pages.
QRDBText	A data-aware version of the TQRLabel that prints the value of a database field. Calculated fields and text field types can be printed, including String fields, various numeric fields, date fields and memo fields. Text can span multiple lines and pages. You connect the component to the data field by setting the DataSource and DataField properties. Unlike regular data-aware components, TQRDBText works even with dataset controls disabled to improve speed.
QRExpr	Prints database fields, calculations, and static text. Input a valid QuickReport expression in the Expression property.
QRSysData	Prints system information such as report title, current page number, and so on. Select the data to print in the Data property. Set any preceding text in the Text property.
QRMemo	Prints a large amount of text that does not come from a database field. It can be static text or you can change it during report generation. You can set the field to expand vertically as needed and then span multiple pages if necessary.
QRExprMemo	Allows you to programmatically generate contents using Quick Report expressions.
QRRichText	Allows you to embed rich text into your report.
QRDBRichText	Provides a Quick Report wrapper for accessing DBRichText fields in your reports.
QRShape	Draws simple shapes like rectangles, circles, and lines on a report.
QRImage	Displays a picture on a report. Supports all image formats supported by the TPicture class.
QRDBImage	Prints images stored in binary (BLOB) fields. Prints all graphics formats supported by C++Builder.
QRCompositeReport	Allows you to combine more than one report together.

QRPreview	Brings up a form that allows you to preview a report on the screen and print it.
QRTextFilter	Lets you export the contents of your report into text format.
QRCSVFilter	Lets you export the contents of your report into a comma-delimited database source file.
QRHTMLFilter	Lets you export the contents of your report into HTML.
QRChart	Allows you to take a TChart component and drop it onto your Quick Report form.

Página *Samples* Iconos de los componentes agrupados en la página *Samples*



Esta página incluye ejemplos de componentes personalizados que pueden construirse e incorporarse a la paleta de componentes. El código fuente está disponible en el directorio \EXAMPLES\CONTROLS\SOURCE de la instalación.

Pie

Performance Graph

CSpinButton

CSpinEdit

CGauge

CDirectoryOutline

CColorGrid

CCalendar

IBEVentAlerter

Página *Standard* Iconos de los componentes agrupados en la página *Standard*

Esta página incluye los componentes comunes y más habituales de los programas Windows:

 MainMenu	Crea una barra de menú (que actúa como menú principal). Para añadir opciones al menú y a los submenús, añadir el componente *MainMenu* al formulario y hacer doble click sobre él para acceder al diseñador de menús.
 PopupMenu	Crea menús desplegables (también llamados menús contextuales) que aparecen cuando se pincha con el botón derecho del ratón. Para configurar el menú desplegable, proceder como con el componente *MainMenu*.
 Label	Muestra texto que el usuario no puede seleccionar ni manipular. Se usa para mostrar textos de título, encabezamientos, o incluso para mostrar resultados, ya que puede establecerse su valor (propiedad *Caption*) en tiempo de ejecución.
 Edit	Muestra un área de edición de texto en la que el usuario puede introducir y modificar una única línea de texto.
 Memo	Muestra un área de edición de texto en la que el usuario puede introducir y modificar múltiples líneas de texto.
 Button	Crea un botón que el usuario puede pulsar para efectuar acciones.
 CheckBox	Presenta una opción binaria (Si/No - Verdad/Falso) de manera que cuando se selecciona este control, se permuta entre ambos valores. Este control puede emplearse para crear un grupo de estos controles que representen elecciones que no sean mutuamente exclusivas (al contrario que los *RadioButton*, por lo que el usuario puede seleccionar más de una opción en un grupo.
 RadioButton	Presenta una opción binaria (Si/No - Verdad/Falso) de manera que cuando se selecciona este control, se permuta entre ambos valores. Este control puede emplearse para crear un grupo de estos controles que representen elecciones mutuamente exclusivas (al contrario que los *CheckBox*, por lo que el usuario puede seleccionar sólo una en un grupo.
 ListBox	Muestra una lista de eleciones que está acompañada de una barra de scroll.
 ComboBox	Muestra una lista de eleciones. Es un control que combina aspectos de un componente *ListBox* y de un componente *Edit*: el usuario puede introducir datos en el área de edición o seleccionar en el área de lista.
 ScrollBar	Prociona una forma cómoda de modificar el área visible de un formulario o de una lista. También puede usarse para "desplazarse" en un rango -amplio- de valores por incrementos

prefijados.

GroupBox	Contenedor para agrupar opciones relacionadas en un formulario.
RadioGroup	Contenedor que crea un grupo de componentes *RadioButton* en un formulario.
Panel	Contenedor que puede contener otros componentes en un formulario. Se usa para crear barras de herramientas y líneas de estado. Los componentes que contiene están asociados al panel.
ActionList	Crea grupos de acciones que centralizan las respuestas de la plicación ante las acciones del usuario.

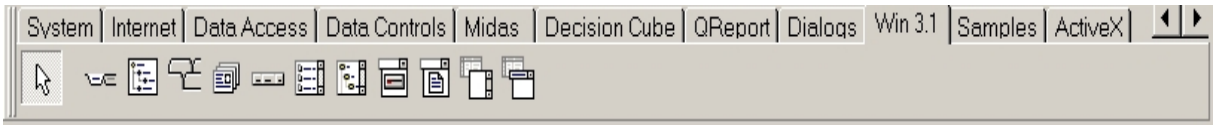
Página *System* Iconos de los componentes agrupados en la página *System*

Esta página incluye controles muy especializados:

Timer	Es un componente no visual que actúa como temporizador. Se emplea para ejecutar una serie de instrucciones (mediante el gestor del evento *OnTimer*) que deben ejecutarse cuando se alcanza el valor especificado como intervalo (propiedad *Interval*).
PaintBox	Especifica un área rectangular sobre un formulario que delimita la zona en la que puede dibujarse desde una aplicación.
MediaPlayer	Muestra una ventana con botones (similares a los de cualquier reproductor de audio o video) para reproducir video o sonido.
OleContainer	Crea un área de cliente OLE (Object Linking and Embedding) en un formulario.
DdeClientConv	Establece una conexión cliente a una aplicación servidora DDE (Dynamic Data Exchange).
DdeClientItem	Especifica los datos del cliente DDE (Dynamic Data Exchange) que se transfieren en una conversación DDE.
DdeServerConv	Establece una conexión servidora a una aplicación cliente DDE (Dynamic Data Exchange).

	DdeServerItem	Especifica los datos del servidor DDE (Dynamic Data Exchange) que se transfieren en una conversación DDE.

Página *Win 3.1* Iconos de los componentes agrupados en la página *Win 3.1*



Esta página incluye controles propios de Windows 3.1 para permitir compatibilidad con aplicaciones antiguas. Muchos de estos controles tienen su versión actualizada en componentes incluidos en la página Win32.

Estos controles no deberían usarse al desarrollar nuevas aplicaciones. En la siguiente tabla se indica qué control debería usarse en su lugar:

Control Win 3.1	**Sustituir por**	**Página**
DBLookupCombo	DBLookupComboBox	Data Controls
TabSet	TabControl	Win32
Outline	TreeView	Win32
TabbedNoteBook	PageControl	Win32
NoteBook	PageControl	Win32
Header	HeaderControl	Win32

DBLookupList	Data-aware list box that displays values looked up from columns in another table at runtime.
DBLookupCombo	Data-aware combo box that displays values looked up from columns in another table at runtime.
TabSet	Creates notebook-like tabs. You can use the TabSet component with the Notebook component to enable users to change pages.
Outline	Displays information in a variety of outline formats.
TabbedNotebook	Creates a component that contains multiple pages, each with its own set of controls. Users select a page by clicking the tab at the top of the page
Notebook	Creates a component that can contain multiple pages. Used with the Notebook component, it enables users to change pages.

Header	Creates a sectioned region for displaying data. Users can resize each section of the region to display different amounts of data.
FileListBox	Displays a scrolling list of files in the current directory.
DirectoryListBox	Displays the directory structure of the current drive. Users can change directories in a directory list box.
DriveComboBox	Displays a scrolling list of available drives.
FilterComboBox	Specifies a filter or mask to display a restricted set of files.

Página *Win32* Iconos de los componentes agrupados en la página *Win32*

Esta página incluye componentes comunes de aplicaciones para 32 bits.

TabControl	Divisor de páginas mutuamente exclusivas accesibles por pestañas.
PageControl	Se emplea para construir cuadros de diálodo con múltiples páginas dentro de la misma ventana.
ImageList	Una lista de imágenes es una colección de imágenes del mismo tamaño accesibles mediante un índice. Se emplea para gestionar eficientemente grandes conjuntos de imágenes o "bitmaps". Paracrear una lista de imágenes, añadir un componente *ImageList* al formulario y hacer doble click sobre él para acceder al editor de listas de imágenes.
RichEdit	Es una especialización del componente *Memo*: proporciona la posibilidad de 1) modificar propiedades acrerca de la fuente de letra (familia, tamaño, color, negrita o itálica), 2) modificar propiedades de formato (alineamiento, tabulaciones, indentación o numeración), y 3) marcación de texto y arrastre del mismo.
TrackBar	Es una barra que muestra un rango y un indicador que muestra el valor actual y que permite modificarlo.
ProgressBar	Es una barra rectangular que se va "llenando" de izquierda a derecha. Se emplea para mostrar cómo progresa una operación.
UpDown	Son botones para incrementar y decrementar valores.

	HotKey	Asigna una combinación de teclas a cualquier componente.
	Animate	Una ventana que se emplea para mostrar ficheros AVI (Audio Video Interleaved) o series de "bitmaps" dispuestos en secuencia, como una película.
	DateTimePicker	Muestra una lista de elecciones que está acompañada de una barra de scroll para seleccionar fechas. La fecha puede seleccionarse desde el calendario o empleando la barra de scroll o las flechas.
	MonthCalendar	Muestra un calendario que representa un solo mes.
	TreeView	Muestra un conjunto de objetos estructurados jerárquicamente.
	ListView	Permite ver una lista en columnas.
	HeaderControl	Muestra un encabezamiento sobre columnas de texto o números. Este control puede dividirse en varias partes para proporcionar cabeceras para múltiples columnas.
	StatusBar	Área para indicar el estado de acciones o textos de pista amplios ("Hints") en la parte baja de la ventana.
	ToolBar	Gestiona botones rápidos y otros controles, colocándolos adecuadamente ajustando su posición y tamaño.
	CoolBar	Muestra una colección de controles "CoolBand" dentro de bandas que pueden moverse y redimensionarse.
	PageScroller	Contiene objetos dentro del área de cliente que pueden desplazarse horizontal o verticalmente sin usar una barra de scroll sino empleando las flechas.

4.5 TApplication - Propiedades, métodos y eventos

4.5.1. Propiedades de las aplicaciones.

Propiedad	**Descripción**
Active	Especifica si la aplicación está activa y tiene el foco.
Hint y ***ShowHint***	***Hint*** especifica el texto que debe aparecer en el cuadro de ayuda o texto de sugerencia asociado a la aplicación. Este cuadro aparece cuando ocurre el evento ***OnHint***. Por defecto se muestra únicamente el cuadro asociado al componente sobre el que está el ratón por lo que

	usualmente se emplea esta propiedad para mostrar la información extensa de este componente en una barra de estado, por ejemplo. ***ShowHint*** determina si los cuadros de sugerencia estarán activados o desactivados para toda la aplicación. Pueden emplearse las funciones ***GetShortHint()*** y ***GetLongHint()*** para obtener las dos partes de la propiedad ***Hint***
MainForm y ***ShowMainForm***	***MainForm*** especifica qué formulario actúa como ventana principal de la aplicación. ***ShowMainForm*** especifica si la aplicación debe mostrar la ventana principal al iniciar su ejecución. Para ocultarla, establecer esta propiedad a *false* antes de la llamada a Application->Run y asegurarse de que la propiedad *Visible* del formulario esté también a *false*.
Icon y ***Title***	Especifican el icono y el texto, respectivamente, que aparecen en la barra de tareas de Windows cuando se minimiza la aplicación. Pueden establecerse en tiempo de diseño seleccionando Project | Options | Application.
HelpFile y ***CurrentHelpFile***.	Para especificar el fichero de ayuda asociado a la aplicación.

4.5.2. Métodos de las aplicaciones.

Método	**Descripción**
BringToFront()	Establece la última ventana activa como la que se muestra por encima de todas las demás.
CreateForm()	Crea un nuevo formulario. Generalmente, el programador no usa este método ya que las líneas de código para la creación de formularios las añade automáticamente *C++ Builder*.
HelpCommand(), ***HelpContext()*** y ***HelpJump()***	Se emplean para gestionar la ayuda asociada a una aplicación. Generan un evento *OnHelp*, y si no hay un gestor para este evento, se gestiona a través de *WinHelp*.
MessageBox()	Muestra un mensaje al usuario en cuadro de diálogo que puede incorporar botones. Devuelve un valor que depende del botón seleccionado para cerrar el cuadro de diálogo.

Minimize() y ***Restore()***	***Minimize()*** minimiza una aplicación y la aloja en la barra de tareas de Windows, mientras que ***Restore()*** reestablece el tamaño que tenía la aplicación antes de ser minimizada.
Run() y ***Terminate()***	***Run()*** ejecuta la aplicación, mientras ***Terminate()*** termina la ejecución de una aplicación.
ShowException()	Muestra un cuadro de diálogo cuando se produce una excepción que no es gestionada por la aplicación. Para especificar qué excepciones pueden gestionarse en la aplicación, construir un gestor para el evento ***OnException***.
HandleException()	Proporciona una manera de gestionar excepciones por defecto en la aplicación.

4.5.3. Eventos de las aplicaciones.

Evento	**Descripción**
OnActivate y ***OnDeactivate***	Ocurren cuando se activa o desactiva la aplicación, respectivamente. Una aplicación se activa cuando se genera la aplicación (empieza a ejecutarse) o cuando estaba activa otra aplicación y una ventana de la aplicación a la que se refiere recibe el foco. Una aplicación se desactiva cuando el usuario selecciona otra aplicación.
OnHint y ***OnShowHint***	***OnHint*** ocurre cuando el cursor se coloca sobre un control o una opción de un menú que puede mostrar un cuadro de sugerencia. Se suele escribir un gestor para este evento cuando se quiere mostrar el texto largo de sugerencia en la barra de estado. ***OnShowHint*** ocurre cuando la aplicación va a mostar un cuadro de sugerencia. Se suele usar para modificar la apariencia del cuadro. Pueden emplearse las funciones ***GetShortHint()*** y ***GetLongHint()*** para obtener las dos partes de la propiedad ***Hint***
OnMinimize y ***OnRestore***	Ocurren cuando la aplicación se minimiza o se reestablece al tamaño normal, respectivamente.
OnHelp	Ocurre cuando la aplicación recibe una petición de ayuda. Los métodos ***HelpContext()*** y ***HelpJump()*** gestionan automáticamente este evento.
OnException	Se emplea cuando se desea modificar el comportamiento

	por defecto de una aplicación cuando se manifiesta alguna excepción que no es tratada por el programa.
OnShortCut	Ocurre cuando se pulsa una tecla.  Se emplea este gestor para realizar acciones antes de que el formulario o los controles del formulario gestionen los eventos oportunos a la pulsación de teclas.

4.6 TComponent - Propiedades, métodos y eventos

4.6.1 Propiedades de los componentes.

4.6.1.1. Propiedades comunes más importantes.

Aunque cada componente tiene sus propias propiedades, métodos y eventos, si es verdad que existen algunas propiedades comunes, que comentaremos brevemente.

Propiedad	**Descripción**
Align	Establece distintas posiciones y ajustes del componente relativos al componente que lo contiene.
Caption y ***Name***	***Caption*** indica el texto que acompaña a algunos componentes. ***Name*** indica el nombre o identificador que se le asigna al componente en el programa y con el que se referirá al componente en la aplicación (en el código).  *C++ Builder* usará el valor de la propiedad ***Name*** para crear el puntero que referenciará al componente y el programador lo usará para referenciarlo.
Color	Se refiere al color de fondo del componente.
Cursor	Establece qué icono se muestra cuando el usuario mueve el cursor sobre un componente.
Font	Permite especificar las propiedades de la fuente de letra que se usará en el componente.
Enabled	Para activar y desactivar componentes: cuando un componente se desactiva no puede recibir el foco, por lo que no responde al ratón, al teclado ni a eventos del temporizador.
Hint y ***ShowHint***	***Hint*** especifica el texto que debe aparecer en el cuadro de ayuda o texto de sugerencia asociado a un componente. Tiene dos partes, separadas por la barra |. La primera

	(sugerencia breve) se visualiza cuando el usuario coloca el puntero sobre el componente. La segunda se presenta en la barra de estado. Esta información aparece cuando ocurre el evento ***OnHint***. ***ShowHint*** determina si la ayuda de sugerencia breve se debe mostrar o no.
ParentColor, ***ParentFont*** y ***ParentShowHint***	Cuando se establecen a verdad, heredan el valor para la propiedad a la que hacen referencia de su componente padre. En otro caso, emplean el valor que tienen establecido para esa propiedad.
Visible	Indica si el componente es visible o no.

4.6.1.2. Otras propiedades de uso corriente.

Las siguientes propiedades, aunque se usan frecuentemente, no están disponibles para todos los componentes.

Propiedad	**Descripción**
BorderStyle	Para diferenciar al componente del formulario o integrarlo en su fondo.
Height y ***Width***	Altura y anchura (en píxeles) del componente.
HelpContext	Para asociar un número de índice de un fichero de ayuda a un componente.
Left y ***Top***	Coordenadas *X* e *Y* del componente (su esquina superior izquierda).
TabOrder y ***TabStop***	Para establecer el order de tabulación y determinar si el componente forma o no parte de la secuencia de tabulación, respectivamente.

4.6.2. Métodos de los componentes.

No se emplean habitualmente porque las acciones que realizan se efectúan modificando las propiedades del componente.

Método	**Descripción**
ClientToScreen() y ***ScreenToClient()***	Convierte las coordenadas del área de cliente en coordenadas de pantalla y viceversa.
Hide() y	***Hide()*** oculta el componente. Puede volver a hacerse visible

Show()	posteriormente con ***Show()***. Ambos métodos modifican la propiedad ***Visible***.
Invalidate(), ***Refresh()***, ***Repaint()*** y ***Update()***	Redibujar un componente.
SetBounds()	Establece simultáneamente los valores de las propiedades ***Top***, ***Left***, ***Width*** y ***Height***.
SetFocus()	Sitúa el foco sobre un componente y lo convierte en el componente activo.
CanFocus()	Devuelve verdad si el componente puede recibir el foco (si las propiedades ***Visible*** y ***Enabled*** están a verdad).

4.6.3. Eventos de los componentes.

Una acción puede desencadenar varios eventos y el orden en ocurren puede ser muy importante: Eventos asociados al movimiento y pulsación con el ratón. Eventos asociados a la pulsación de alguna tecla.

Evento	**Descripción**
OnMouseDown, ***OnMouseMove***, ***OnMouseUp***, ***OnClick*** y ***OnDblClick***	Para responder a los eventos que suceden al mover el ratón o cuando se pincha con él sobre el componente.
OnEnter y ***OnExit***	***OnEnter*** ocurre cuando se activa un componente (recibe el foco) siempre que el foco se transfiera desde otro componente del mismo formulario. ***OnExit*** ocurre cuando el componente pierde el foco, siempre que el foco se transfiera a otro componente del mismo formulario.
OnKeyDown, ***OnKeyUp*** y ***OnKeyPress***	Para responder a los eventos que suceden al pulsar alguna tecla cuando el foco está en un componente.
OnPaint	Este evento ocurre cuando un objeto tiene que ser redibujado. Si el objeto es un *PaintBox*, este gestor se encargará de dibujar sobre el *Canvas* y si no se proporciona este gestor, el *PaintBox* no se verá en tiempo de ejecución.

4.7 TForm - Propiedades, métodos y eventos

4.7.1. Propiedades de los formularios.

4.7.1.1. Propiedades en tiempo de diseño y ejecución.

Propiedad	Descripción
ActiveControl	Establece qué componente tiene el foco cuando se abre el formulario.
Enabled	Indica si el formulario está activo (si puede recibir el foco).
AutoScroll, ***HorzScrollBar*** y ***VertScrollBar***	Controlan conjuntamente las barras de desplazamiento de un formulario.
BorderIcons	Indica qué iconos aparecen en la barra de título de un formulario.
BorderStyle	Indica qué tipo de borde debe tener el formulario.
Caption y ***Name***	***Caption*** indica la cadena que aparece en la barra de título del formulario mientras que ***Name*** indica con qué nombre se refiere al formulario en la aplicación (en el código).
Color	Color de fondo del formulario.
Cursor	Especifica qué cursor se muestra cuando está sobre el formulario.
Font	Permite especificar las propiedades de la fuente de letra que se usará en el formulario. Se aplica a todos los componentes ubicados en el formulario.
Icon	Establece el icono que se verá en la barra de título cuando se muestre el formulario en tiempo de ejecución y cuando se minimize.
Height y ***Width***	Altura y anchura (en píxeles) del formulario.
ClientWidth y ***ClientHeight***	Altura y anchura (en píxeles) del área de cliente del formulario (el área de cliente es la que queda dentro de los bordes del formulario y por debajo de la barra de título y de la barra de menú). Al modificar estas propiedades se actualizan ***Width*** *y* ***Height***.
Left y ***Top***	Coordenadas *X* e *Y* del formulario (su esquina superior izquierda) en la pantalla.
Position	Determina el tamaño y la posición del formulario.

FormStyle	Para especificar si un formulario es o no parte de una aplicación multiformulario.
HelpFile y ***HelpContext***	Para especificar el fichero de ayuda asociado a la aplicación y gestionar la ayuda contextual.
Hint y ***ShowHint***	***Hint*** especifica el texto que debe aparecer en el cuadro de ayuda o texto de sugerencia asociado al formulario. Este cuadro aparece cuando ocurre el evento ***OnHint***. Por defecto se muestra únicamente el cuadro asociado al componente sobre el que está el ratón por lo que usualmente se emplea esta propiedad para mostrar la información extensa de este componente en una barra de estado, por ejemplo. ***ShowHint*** determina si los cuadros de sugerencia estarán activados o desactivados para el formulario.
Visible	Indica si el formulario es visible o no. Los métodos ***Show()*** y ***ShowModal()*** hacen que un formulario sea visible y lo colocan por encima de todos.
WindowState	Para establecer el estado inicial del formulario (normal, minimizado o maximizado) o consultar su estado.

4.7.1.2. Propiedades en tiempo de ejecución.

Propiedad	**Descripción**
Active	Indica si el formulario está activo (si tiene el foco).
Canvas	El *lienzo* es la superficie del formulario sobre la que se puede dibujar. La propiedad ***Canvas*** da acceso a esta superficie para dibujar líneas, mapas de bits, texto, ... en el área de cliente. Suele usarse con el gestor del evento ***OnPaint***.
ClientRect	Coordenadas de los cuatro puntos que delimitan el área de cliente del formulario.
ModalResult	El valor devuelto cuando se cierra un formulario *modal* (abierto con el método ***ShowModal()***).
FormState	Proporciona información acerca del estado en que se encuentra un formulario (si se está creando, si es visible, si es modal, etc.).

4.7.2. Métodos de los formularios.

Método	Descripción
BringToFront() y ***SendToBack()***	Coloca al formulario en primer plano o en último, respectivamente.
Hide()	Oculta el formulario (establece la propiedad ***Visible*** a falso).
Print()	Imprime el contenido del formulario.
Show() y ***ShowModal()***	Para mostrar formularios. ***ShowModal()*** ejecuta (abre) el formulario *modalmente*: debe cerrarse para que el usuario pueda seguir trabajando con la aplicación.
Close() y ***CloseQuery()***	***Close()*** cierra el formulario tras llamar a ***CloseQuery()*** para asegurarse de que es correcto cerrarlo. Esta última llama al manipulador del evento ***OnCloseQuery***.
SetFocus()	Activa el formulario y lo coloca en primer plano, poniendo a verdad la propiedad ***Active***. Además, el componente especificado en la propiedad ***ActiveControl*** recibe el foco.
CanFocus()	Devuelve verdad si el formulario puede recibir el foco (si las propiedades ***Visible*** y ***Enabled*** están a verdad).
SetBounds()	Establece simultáneamente los valores de las propiedades ***Top***, ***Left***, ***Width*** y ***Height***.
ClientToScreen() y ***ScreenToClient()***	Convierte las coordenadas del área de cliente en coordenadas de pantalla y viceversa.
Invalidate(), ***Refresh()***, ***Repaint()*** y ***Update()***	Redibujar un formulario.
Release()	Destruye un formulario y libera toda la memoria asociada.

4.7.3. Eventos de los formularios.

Una acción puede desencadenar varios eventos y el orden en ocurren puede ser muy importante:

- Eventos asociados a la creación y destrucción de un formulario.
- Eventos asociados al movimiento y pulsación con el ratón.
- Eventos asociados a la pulsación de alguna tecla.

4.8. Estructura y configuración de un proyecto

4.8.1 Estructura habitual de un proyecto escrito en *C++*

Una aplicación escrita en C++ se compone de un módulo (fichero) que contiene la función main() y, opcionalmente, de una serie de módulos adicionales que pueden contener declaraciones de clases, declaraciones de funciones (prototipos), definiciones de clases, etc.

Para la creación de un ejecutable se requiere:

- Un fichero makefile en el que se especifica la dependencia entre los diferentes módulos y la manera en la que se generan los módulos objeto, bibliotecas y el ejecutable final.
- Ejecutar el programa make para que interprete el fichero makefile y desencadene las órdenes oportunas para la creación del ejecutable, de acuerdo a lo especificado en el fichero makefile.

4.8.2 Ficheros de proyecto en *C++ Builder*

En *C++ Builder* la gestión de proyectos software se realiza automáticamente y el usuario no necesita construir el fichero makefile ni ejecutar make para obtener el ejecutable: *C++ Builder* se encarga de todo ésto. Para este trabajo estructura los ficheros asociados a una aplicación en un **proyecto**. Se puede decir que un fichero de proyecto es un fichero makefile. La extensión de estos ficheros es .bpr (**B**orland **Pr**oject).

En todo proyecto existe un fichero que contiene la función principal. En aplicaciones que hacen uso de la VCL la función main() se llama WinMain() y se aloja en un fichero cuyo nombre coincide con el del proyecto.

Habitualmente **no** es preciso trabajar sobre este fichero.
Si fuera preciso, para mostrar en el editor de código el fichero que contiene a la función WinMain(), seleccionar Project | View Source).

Ejemplo. La función WinMain() se alojará en el fichero Inicial.cpp

Además, cualquier aplicación contendrá al menos una ventana. Para cada ventana existirán dos módulos adicionales:

Un módulo con extensión .cpp con el código asociado a esa ventana (usualmente contiene los gestores de los eventos que se pueden gestionar en esa ventana).

Un módulo cabecera con extensión .h que contiene la declaración de los componentes que se incluyen en la ventana.

Ejemplo. En este ejemplo, los ficheros asociados a la ventana principal (y única) se llamarán main.cpp y main.h.

4.8.3 Cómo configurar un proyecto en *C++ Builder*

El primer consejo es que *cada aplicación debería estar en un directorio propio* donde se guardarán todos los ficheros asociados a la aplicación.

Ejemplo. Crear un directorio (p.e. Ejercicio) para guardar los ficheros asociados al proyecto.

Ahora se procede a **configurar la aplicación**. Consiste, básicamente en proporcionar nombres significativos al proyecto y a los ficheros asociados al proyecto.

- Si se acaba de arrancar *C++ Builder* aparece, por defecto aparece un formulario vacío llamado Form1 asociado al proyecto Project1.bpr.
- Si existe un proyecto vigente, lo más cómodo es crear una nueva aplicación (File | New Application) salvando, si se desea el proyecto vigente.

En cualquier caso se crean, por defecto:
Project1.bpr, fichero de proyecto.
Project1.cpp, fichero que contiene la función WinMain(),

Unit1.cpp, fichero que contiene las funciones asociadas al formulario principal (constructor de la ventana y que posteriormente contendrá, además, los gestores de los eventos que puede gestionar), y

Unit1.h (especificacion de los componentes del formulario principal).
Puede comprobarse usando el gestor de proyectos (View | Project Manager).
Los pasos a seguir son los siguientes:

Cambiar el nombre y título del formulario principal utilizando el inspector de objetos.

Ejemplo. Cambiar la propiedad *Caption* por *Form de Prueba* y la propiedad *Name* por *MainForm*.

Guardar ficheros asociados a ventanas (File | Save As) en el directorio adecuado para dar nombres significativos a los ficheros.

Ejemplo. Guardar Unit1.cpp como main.cpp. (Unit1.h cambia automáticamente a main.h).

Guardar el proyecto: (File | Save Project As) en el directorio adecuado.

> **Ejemplo.** Guardar Project1.bpr como Inicial.bpr. (Observar que Project1.cpp ha cambiado automáticamente a Inicial.cpp, y que éste contiene a la función WinMain). Comprobar que Inicial.bpr tiene la estructura de un fichero makefile.

Observar que el proyecto vigente se llama ahora Inicial. Podemos comprobarlo desplegando el menú Project.

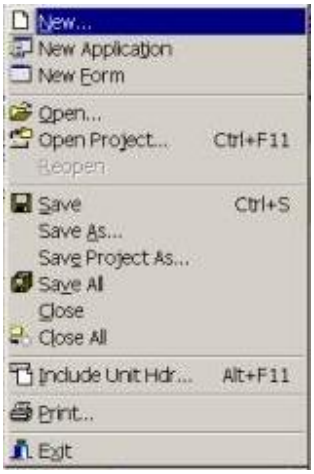
Dar un nombre a la aplicación y fijar un icono.

Estos datos aparecerán al minimizar el programa durante su ejecución. Selecionar Project | Options | Application.

4.9 Lista completa de menús

Sobre el menú principal y sus submenús...

File

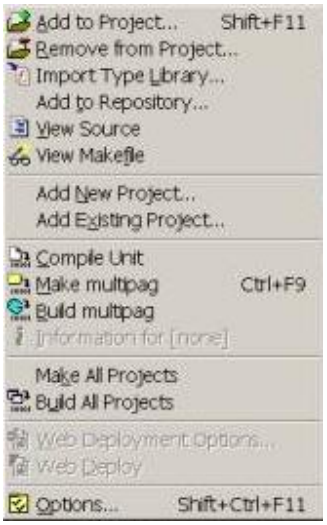


Edit

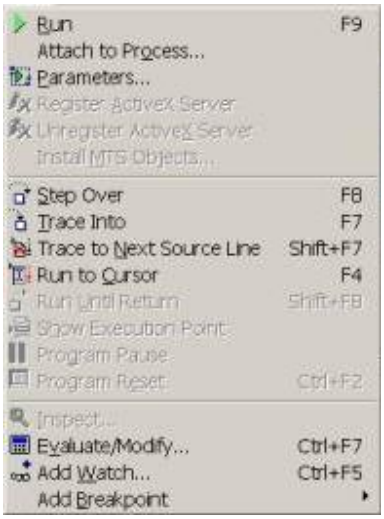
Search

View

Project



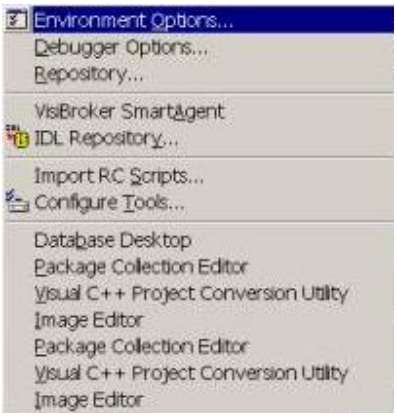
Run



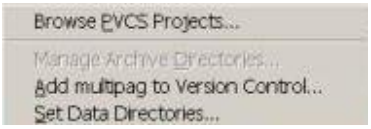
Component

Database

Tools



Workgroups



Help