Daniel Villahermosa Domínguez
dvilla@isys.dia.fi.upm.es

1 de Julio del 2002

1. WorkSpaces y Proyectos

Los proyectos están incluidos dentro de workspaces, que es algo así como un "espacio de trabajo", un lugar en el cual vamos a ir incluyendo proyectos, los cuales podrán estar o no relacionados.

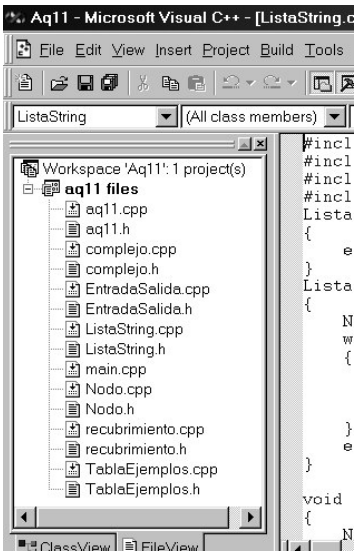
Los proyectos pueden ser de varios tipos y Visual C, al crear un nuevo proyecto, nos da varias opciones, donde las más usuales son:

1. **Win32 console aplication:** Con este tipo de proyecto se podrán crear aplicaciones en modo consola, es decir, el típico programa de MS-DOS:

```
void main()
{
        int i;
        .....
        .....
}
```

Este tipo de proyectos se suelen utilizar para crear los "valida" que son ejecutables que validan una librería o un conjunto de clases (programas de prueba) que hemos creado con anterioridad. (Con este tipo de proyecto se creará un fichero *.EXE ejecutable)

Cuando creamos un proyecto de este tipo, siempre nos suelen salir varios folders (contendedores de ficheros) en el proyecto, en este caso concreto es mejor eliminarlos e introducir todos los ficheros *.cpp y *.h en el raíz de este modo.



2. **Win32 Static Library:** Con este tipo de proyecto crearemos librerías estáticas. Generará en lugar de un ejecutable un fichero [NOMBRE_PROYECTO].LIB el cual podrá ser utilizado en cualquier otro programa siempre y cuando, en el mismo directorio, estén todas las cabeceras que vayamos a utilizar (ficheros *.h) y además la librería *.LIB que utilicemos.

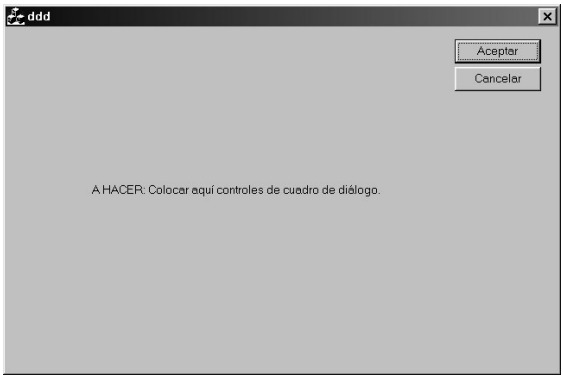
Es muy importante también, para poder utilizar una librería de este tipo en un proyecto, el incluirla en los settings del proyecto como librería adicional. En los folders de source introduciremos los *.cpp y en los de include los *.h.

3. **MFC AppWizzard(exe):** Creará un proyecto para poder generar una aplicación Windows con las librerías MFCs, las cuales, son una librerías propias de microsoft hechas para facilitar la creación de ventanas, botones, y otros controles propios de windows.

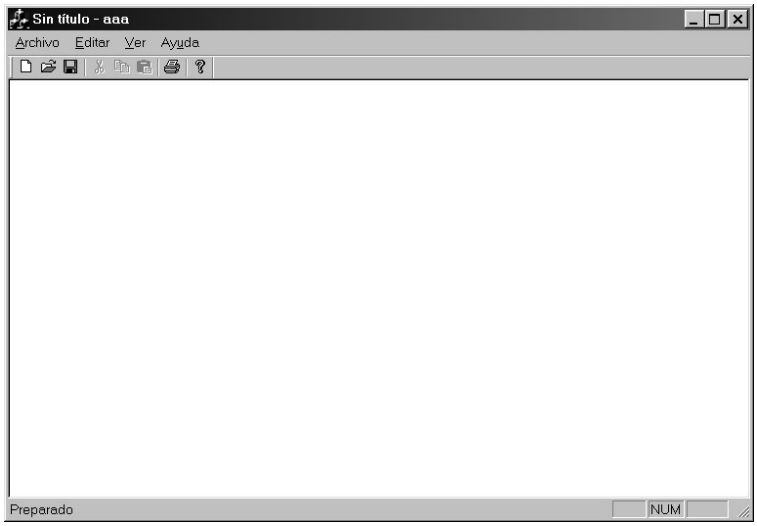
Con las MFC se crea un archivo ejecutable que necesita unas librerías DLL para poder ser ejecutado en cualquier otra máquina, estas son:

Con las MFC se pueden crear tres tipos de aplicaciones:

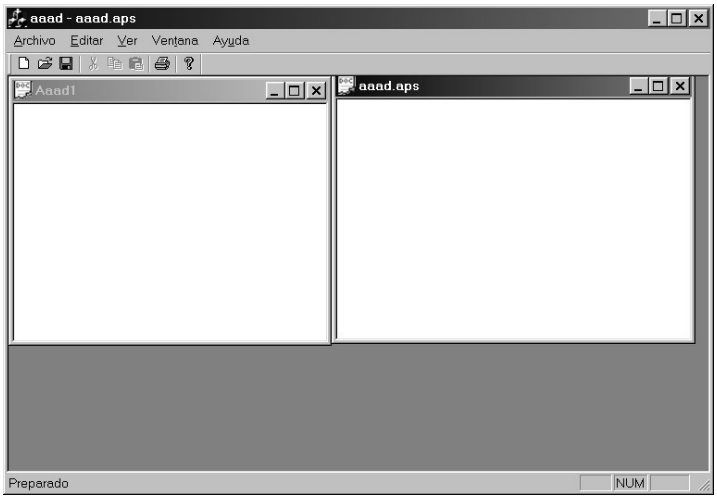
a) Orientadas a Diálogo: Utilizaremos diálogos (o formularios) para crear nuestros interfaces.



b) De único documento (single document) : Estas incluyen un menú (aunque los menúes pueden insertarse también en los diálogos) y un objeto de entrada y salida llamado vista.



c) Multidocumento (multidocument): Contienen varias vistas seleccionables.



Nos vamos a centrar en las aplicaciones basadas en diálogo o formularios como se suelen llamar en Visual Basic que engloban todo lo que básicamente se debe conocer a la hora de realizar una aplicación en Visual C.

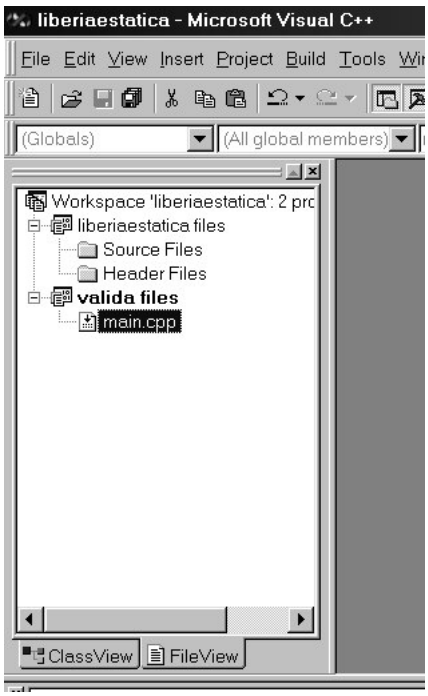
2. Dependencias entre proyectos

Dentro de un mismo workspace como veíamos antes podemos tener varios tipos de proyectos ¿ cuándo utilizar uno y no otro?

1. Utilización de sólo un proyeto en modo consola(Win32 console).

Se utiliza cuando vamos a crear aplicaciones MS-DOS. Hay que recordar eliminar los folders, ya que pueden dar problemas durante la compilación.
También se pueden utilizar librerías estáticas en estos proyectos (en general en todos), para ello habría que tener las cabeceras y el/los fichero/s *.LIB en el directorio donde se encuentra el proyecto e incluir en los settings dicha/s librería/s.

2. Utilización de una librería estática con un proyecto window32 en modo consola



Normalmente se utiliza para crear las **pruebas de una librería** que hemos creado. Por ejemplo, MOREA (una aplicación del laboratorio) es un conjunto de clases que quedan en una librería, para poder validarla creamos proyectos win32 console en el que probamos las clases.

Se debe:
1) El proyecto consola debe tener como dependencia la librería
   (Project →dependecies)
2) Introduciremos las cabecera de la librería en el directorio donde se encuentre el valida.
3) Poner el valida como proyecto activo.

3. Utilización de sólo un proyecto con MFCs.

Se utiliza para crear interfaces. Se pueden añadir librerías *.LIB y sus cabeceras (es lo más normal cuando tenemos la aplicación final)

4. Utilización de un proyecto MFC con una librería estática.

Se suele usar sobretodo durante el desarrollo de una aplicación.

En definitiva, en un Workspace, puede convivir todo tipo de proyectos y todo depende de lo que queramos, lo que hemos visto sólo son posibles sugerencias para un buen desarrollo.

La creación de librerías y clases es igual que en cualquier compilador de C++ estándar, sólo hay que tener en cuenta lo siguiente:

Visual C, por convenio crea sus clases poniendo "C[consonante]_ _ _ ... ", si nosotros en C++ estándar creamos una clase que se ajuste a es formato de nombre, interpretará que es una clase creada con Visual C y no compilará bien. Las opciones son:

a) crear las clases con Visual C (sólo podrán ser utilizadas en Visual C)

b) No crear clases con ese formato(serán utilizables en cualquier compilador de C++)

3. Aplicaciones y conceptos básicos en Visual C

3.1. Hola Mundo

Lo más fácil

AfxMessageBox(cadena de texto,OPCIONES)

Este método sacará por pantalla un diálogo en el que pone el texto que nosotros le hemos introducido. Se suele usar para proporcionar mensajes de alerta y errores.

Ejemplo: *AfxMessageBox("HOLA MUNDO");*

Las opciones se pueden mirar en las MSDN, pero básicamente lo que hacen es elegir el tipo de icono que queremos que salgan,los botones, etc...

Escribir "Hola mundo" en un Edit Box:

Usaremos el método **GetDlgItem(IDENTIFICADOR)** para poder obtener un puntero a la ventana del objeto identifiado con un ID_.....

Si suponemos que el identificador de nuestro Edit Box (el recuadro blanco) es IDC_PANTALLA.

```
CEdit * pantalla;

pantalla = (CEdit *) GetDlgItem(IDC_PANTALLA);
pantalla->SetWindowText("HOLA MUNDO");
```

GetDlgItem devuelve un puntero a un objeto, como no sabe a que tipo pertenece y todos los objetos derivan de la clase CWnd (ventana) devolverá un puntero a CWnd, para poder utilizar los métodos de Cedit tenemos que tiparlo (poner (CEdit *)). En este caso no haría falta hacer todo eso ya que el método SetWindowText lo tienen CWnd, por eso, bastaría con haber puesto lo siguiente:

GetDlgItem(IDC_PANTALLA)->SetWindowText("HOLA MUNDO");

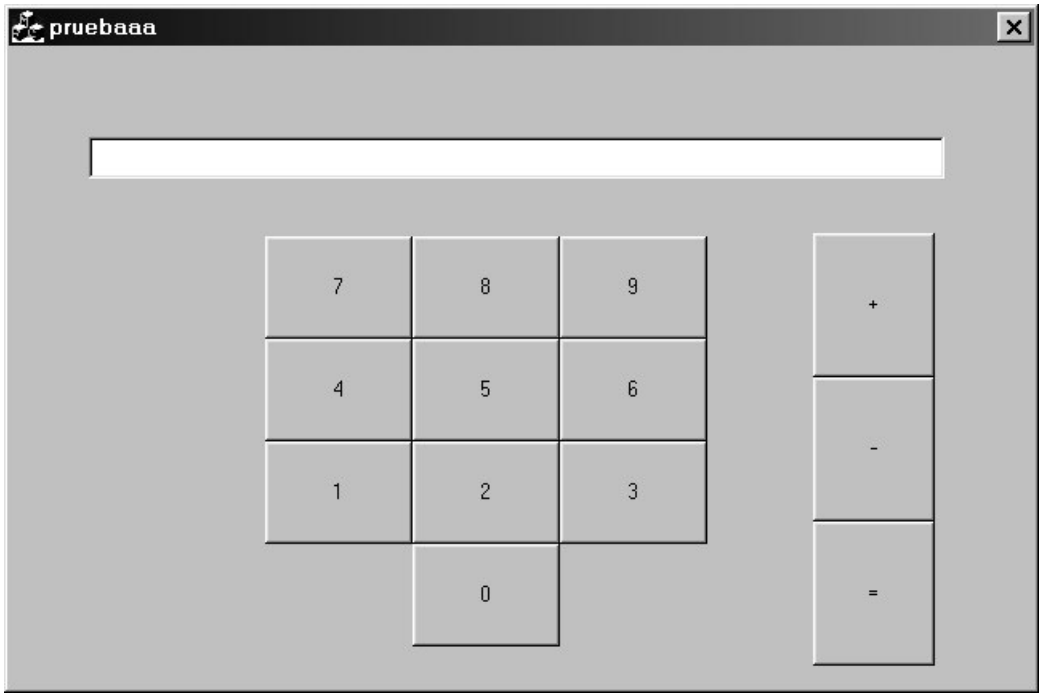
Existe un método que hace la operación contraria que es obtener un texto de una ventana, este es:

GetWindowText(BUFFER, NBUFFER)

Por ejemplo: Este código sacaría por pantalla lo que hemos escrito en el Edit Box.

```
char buffer[100];
GetWindowText(buffer,100);
AfxMessageBox(buffer);
```

3.2. Crear una Calculadora básica en Windows



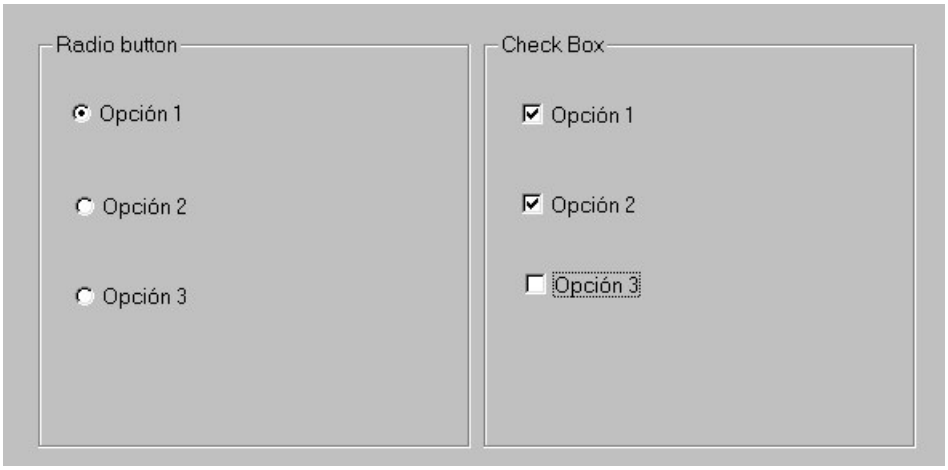
Creando un diálogo de este aspecto en un proyecto al que vamos a llamar Calculadora, nos saldrán en nuestros forders ficheros llamados:

CalculadoraDlg.cpp
CalculadoraDlg.h

Éste será el objeto Diálogo que hemos creado, el cual podremos incluir atributos,métodos, etc, como con cualquiero otra clase en C++. Utilizando lo visto hasta ahora, se puede realizar una pequeña calculadora como esta.
Se propone como ejercicio

En recuerdo a David Bravo, exbecario del ISYS y fundador del ejemplo "Calculadora"

3.3. Botones de selección u opción (radio button y check box)



Estos objetos derivan de la clase CButton (boton normal y corriente), de hecho, existe métodos que son capaces de convertir uno en otro durante la ejecución del programa. La diferencia básica es que en el caso de Radio button, cuando se pulsa sobre una opción de un conjunto de opciones las otras quedan descartadas automáticamente.
En el caso de Check box, se pueden marcar todas las opciones que queramos.

Ambas utilizan los mismos métodos y la única diferencia radica en lo antes mencionado.

Para poder utilizar los métodos correspondientes a una opción usando GetDlgItem, habrá que tipar la salida de la función de esta forma:

((CButton *)GetDlgItem(ID_identificador del botón))→METODO A UTILIZAR

Los métodos más comunes son

- GetCheck() : devuelve true si está seleccionado
- SetCheck(n) : pone como seleccionada la opción { 0 → quitar check ; 1 → poner check }

(MSDN ofrece más información sobre los métodos que se puede utilizar)

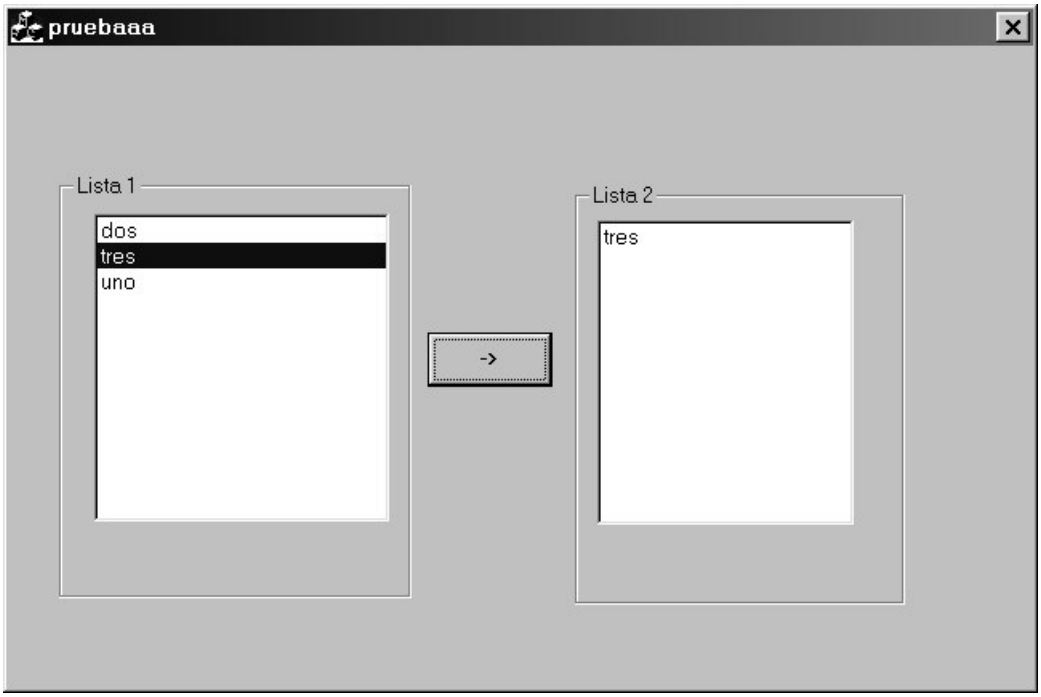
3.4. El control ListBox (lista de elementos)

La clase CListBox se utiliza para almacenar listas de Strings y mostrarlas de forma ordenada por pantalla. Para utilizarlos seguiremos la misma metodología que hemos utilizado hasta ahora con Radio burron, Check box, EditBox, etc.

1) Obtendremos un puntero CWnd con GetDlgItem
2) Lo tiparemos
3) Utilizaremos sus métodos.

Como métodos importantes tenemos:

- **AddString**(STRING): añade un nuevo String
- **ResetContent**() : vacia el contenedor
- **GetCoun**t() : obtiene el total de elementos
- **GetCurSel**(): obtiene la posición del objeto seleccionado
- **GetText**(int,CSTRING): obtiene el texto de la posición int y lo introduce en CSTRING

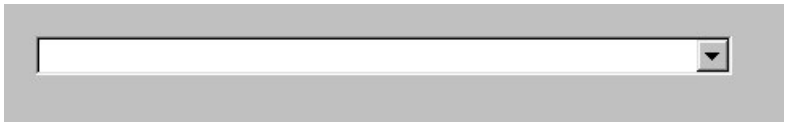


Por ejemplo, para copiar Strings de la lista 1 a la lista 2:

```
CListBox *a;
CListBox *b;
CString buffer;
a=(CListBox *)GetDlgItem(IDC_LIST1);
b=(CListBox *)GetDlgItem(IDC_LIST2);
int i= a->GetCurSel();
a->GetText(i,buffer);
b->AddString(buffer);
```

Hay otro que se llama CComboBox, es parecido (Mirar MSDN).

3.5. El control CComboBox



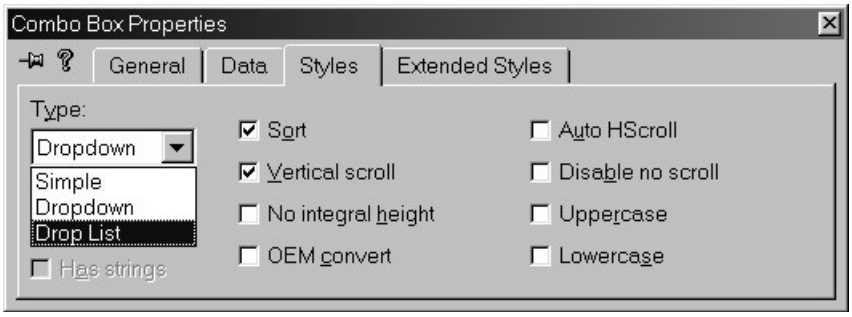
El combobox funciona de forma parecida al ClistBox, utilizando GetCurSel(), GetText() .... es fácil de utilizar si se comprende como funciona ClistBox, pero hay que tener en cuenta algunas cosas importantes y problemas que pueden surgir al utlizarlo.

- Para meter datos en el campo DATA del menu de propiedades, hay que hacer lo siguiente:

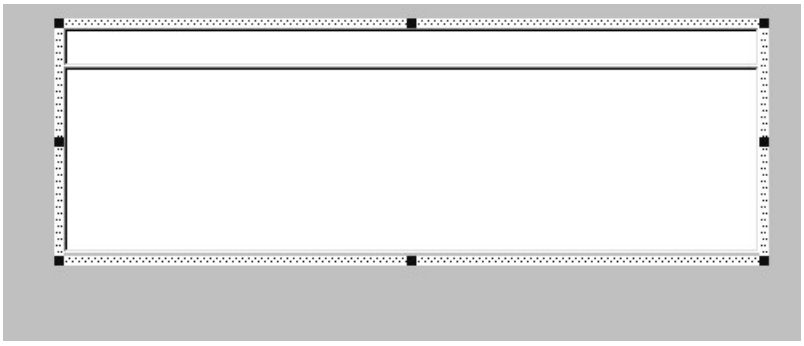
  meter un String y pulsar Control+Return para introducir otro

- Si metemos el ComboBox y compilamos, podemos observar que no se suele abrir, esto es debido a que debemos ponerle el tamaño de la lista que muestra, el truco consiste en seguir estos pasos:

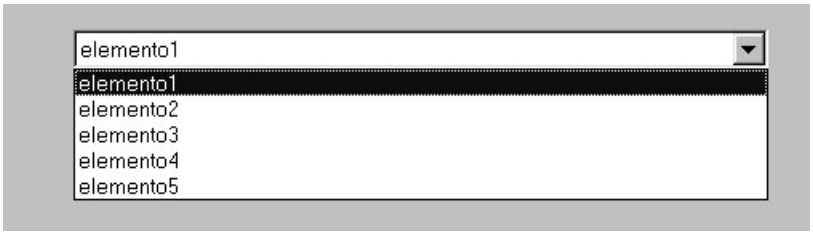
  - Nos metemos en propiedades en el menu de Styles y lo ponemos como simple.



  - Estiramos el ComboBox hasta que se vea la lista de elementos:



  - Lo volvemos a poner en modo DropList y ya podemos acceder a los elementos al compilar.



3.6. Tipos de Diálogos (Modales y no modales)

Existen dos tipos de Diálogos:

- **Modales:** se trata de aquellos dialogos que se cargan y no dejan acceder a la aplicación que lo ha creado (padre). Se crean siempre con el método doModal()

  Ejemplo: CMiDialogo dialogo;
  dialogo.doModal();

- **No modales:** Al cargar uno podemos acceder a la pantalla que lo creo y además podremos tener varios a la vez. Estos se crearán simplementen haciendo new y realizando unos cambios que ahora veremos en la clase del dialogo.

DIALOGOS NO MODALES

Un diálogo no modal se creará simplemente haciendo un :

CMiDialogo *dialogo = new CmiDialogo(this);

Y se destruirá desde donde se ha creado haciendo simplemente:
delete dialogo;
Pero para que el diálogo se cree al hacer simplemente un new, debemos realizar un pequeño cambio en el constructor del diálogo no modal.El cambio consiste en añadir una simple línea:

```
CMiDialogo::CmiDialogo(CWnd* pParent /*=NULL*/)
      : CDialog(CAaaDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CAaaDlg)
      //}}AFX_DATA_INIT
      if(Create(CmiDialogo::IDD,pParente)) ShowWindow(SW_SHOW);

}
```

Básicamente lo que le estamos diciendo es que cuando se cree y si no hay problemas se muestre la ventana.

ACCESO AL PADRE

La ventana desde la cual se crea el diálogo (sea modal o no modal) se llama ventana padre, a ella podemos acceder a través del diálogo ya que se la podemos pasar como parámetro al crearse con el puntero a this.

MODAL	NO MODAL
`CMiDialogo dialogo(this);` Dialogo.Modal();	CMiDialogo *dialogo=new CMiDialogo(this);

Si no se le pasa nada, se toma por defecto que el padre es NULL.

Una forma de poder acceder a los métodos y atributos públicos del padre sería tan sencillo como incluir en el hijo un atributo que sea un puntero al padre:

CclasePadre *padre;

Luego inicializarlo en el constructor de la siguiente forma:

```
CMiDialogo::CmiDialogo(CWnd* pParent /*=NULL*/)
      : CDialog(CAaaDlg::IDD, pParent)
{
```

```
      //{{AFX_DATA_INIT(CAaaDlg)
      //}}AFX_DATA_INIT
      if(Create(CmiDialogo::IDD,pParente)) ShowWindow(SW_SHOW); (*1)
      padre = (CclasePadre *) pParent;
}

(*1) Para el caso de un diálogo modal, esta línea sobraría.
```

3.7. Uso de Menus

INCLUSIÓN DE UN MENÚ EN UN DIÁLOGO

Para incluir un menú dentro de un diálogo basta con seguir los siguiente pasos:

1) Crear un menú en nuestro proyecto(tendrá un identificador al igual que los diálogos,los iconos,etc)
2) En el diálogo en el cual queremos introducir dicho menú lo insertaremos en propiedades→Menu (donde incluiremos el idenficador de nuestro menu)

Con sólo estos dos pasos ya tendremos un menú dentro del diálogo.

Si queremos que por ejemplo, cuando pulsemos uno, el programa haga algo, bastará con ir al menú en nuestros recursos, pulsar con el botón derecho sobre "uno" y aplicar ClassWizzard buscando el evento "click".

NOTA: *Cuando abrimos sobre un menú el class wizzard, nos preguntará a qué clase queremos asociar el diálogo (se trata de donde va a residir el código asociado). Normalmente se asocia a una clase que ya existe. Si el menú esta insertado a un diálogo, lo más natural es elegir la clase de este diálogo para que resida el código de nuestro menú. Si el menú no pertenece a un solo diálogo, o es flotante se suele asociar a la aplicación (el objeto Nombreapp.h).*

CREAR UN MENÚ QUE APAREZCA AL PULSAR SOBRE UN BOTÓN SOBRE LA PANTALLA

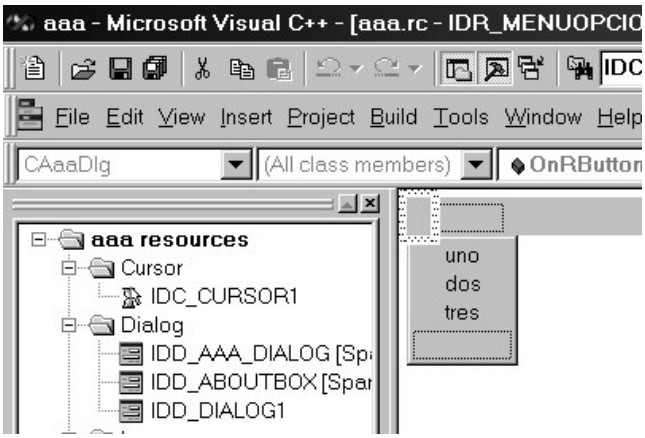
Los pasos a seguir son los siguientes:

1) Creamos un menú que tendrá un identificador, por ejemplo **IDR_MENUOPCIONES**

2) Sobre el diálogo buscamos la zona de código asociada a la captura del evento pulsar botón **(WR_RBUTTONDOWN)**

3) Incluimos el siguiente código:

NOTA: el menu creado debe ser un POP_UP creado en el submenu 0 cuya cabecera no tendrá nombre:



// tenemos como variable global o como atributo de la clase
//Cmenu *menu =NULL;

```
CMenu *menu = NULL;
void CAaaDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
	// TODO: Add your message handler code here and/or call default
	CMenu aux;
	aux.LoadMenu(IDR_MENUOPCIONES);
	menu = aux.GetSubMenu(0);
	menu->TrackPopupMenu(nFlags,point.x,point.y ,this);
	CDialog::OnRButtonDown(nFlags, point);
}
```

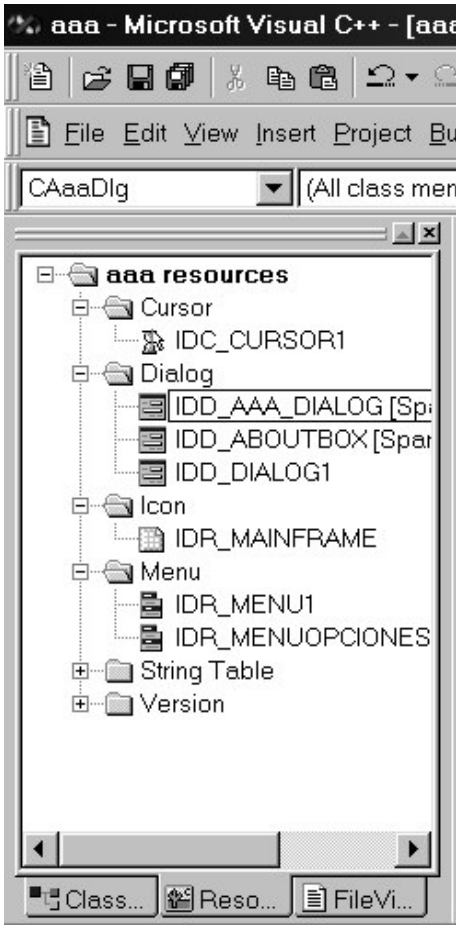
Con este código, no saldrá el menú justo en el puntero del ratón si no que aparecerá en otro sitio, para ajustar esto bastará con incluir lo siguiente:

```
CMenu *menu = NULL;
void CAaaDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
	// TODO: Add your message handler code here and/or call default
	CMenu aux;
	CRect form;
	this->GetWindowRect(&form);
	aux.LoadMenu(IDR_MENUOPCIONES);
	menu = aux.GetSubMenu(0);
	menu->TrackPopupMenu(nFlags,point.x+form.left,point.y+ form.top +50  ,this);
	CDialog::OnRButtonDown(nFlags, point);
}
```

4. Programación gráfica con Diálogos

4.1. El fichero de recursos de Visual C

l



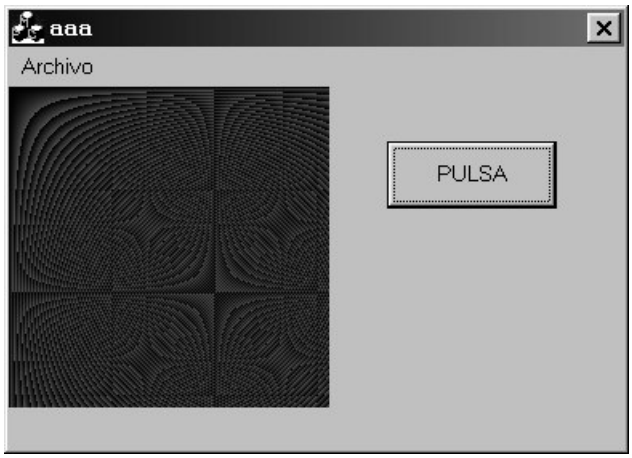
El fichero de recursos, es un fichero *.RC en el cual podemos encontrar todos los recursos que vamos a utilizar en nuestro proyecto. Están divididos en folders. Uno para los cursores,otro para los dialogos,iconos,mapas de bits,menus,etc.

4.2. Dibujar sobre un diálogo puntos

Los diálogos,los botones, los EditBox, en definitiva, las ventanas disponen del CDC o contexto de dispositivo. Se trata de la superficie de ventana que vemos por la pantalla. Sobre el CDC podemos dibujar. Una clase derivada de CDC, es CclientDC. Ahora vamos a ver como podemos utilizarla sobre un diálogo.

EJEMPLO: Sobre un dialogo ponemos un botón y ponemos el siguiente código

```
CClientDC pantalla(this);
int x,y;
for(x=0;x<200;x++)
   for(y=0;y<200;y++)
          pantalla.Setixel(x,y,RGB(x*y,0,0));
```



SetPixel(x,y,COLOR) es un método que lo que hace es situar en la posición (x,y) del dialogo un punto del color indicado. El color se obtiene con la función **RGB(ROJO,VERDE,AZUL).**
ROJO,VERDE y AZUL son valores de 0..255 que indican la intensidad de esos colores que tienen.

Otra forma de conseguir el CDC del propio diálogo es:

CDC *pDC;
pDC=this->GetWindowDC();

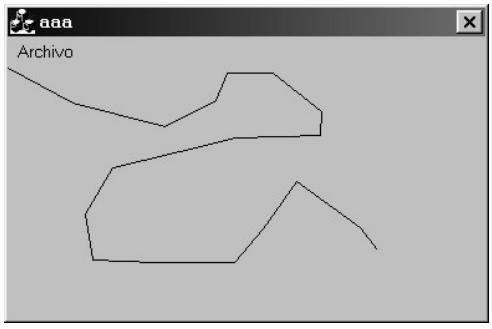
La diferencia entre los dos métodos es que con CclientDC obtenemos la zona de trabajo del diálogo (la zona donde insertaremos los botones, y todo) y con GetWindowDC obtendremos toda la ventana completa.

4.3. Dibujar sobre un diálogo líneas,Rectángulos,elipses

Para ver como dibujar líneas vamos a realizar un pequeño programa, que capture la pulsación del puntero del ratón sobre el diálogo para luego dibujar una línea desde el último punto pulsado, para ello llevaremos como atributos en el diálogo el último punto pulsado (inicialmente es el (0,0))
El código asociado a la captura del evento pulsar es:

Cpoint p; //inicialmente (0,0)

```
void CAaaDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
        CClientDC pDC(this);
        pDC.MoveTo(p); //movemos el puntero de comienzo
        p.x=point.x;
        p.y = point.y;
        pDC.LineTo(point);//dibujamos una linea
        CDialog::OnLButtonDown(nFlags, point);
}
```



Para dibujar un rectángulo:

Rectangle(x1,y1,x2,y2);

Una elipse:

Ellipse(x1,y1,x2,y2);

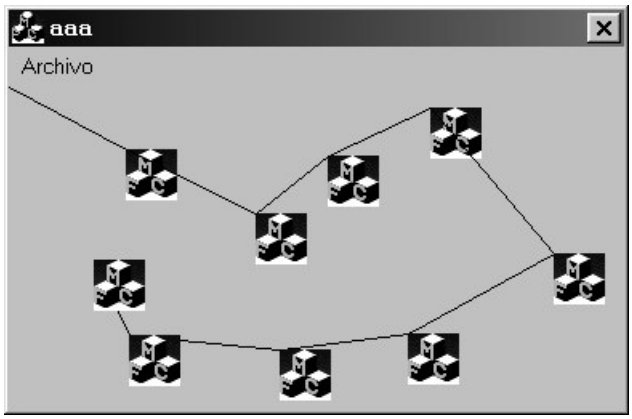
Hay otros pero son igual de sencillos (ver MSDN)

4.4. Dibujar iconos sobre un diálogo

Lo primero que hay que hacer es insertar en nuestro fichero de recursos un icono, el que queramos. Posteriormente le pondremos un identificador.
La forma de dibujar un icono sobre el contexto de dispositivo de un diálogo es:

CClientDC pDC(this);
HICON icono;//un icono
CWinApp *pAplicacion=AfxGetApp();//cargamos la aplicación principal
icono=pAplicacion->LoadICon(IDI_nombreIcono);
pDC->DrawIcon(posicionx,posiciony,icono);

Podemos coger la aplicación de antes, la de las líneas y ahora cada vez que dibujemos una línea podemos dibujar un icono:



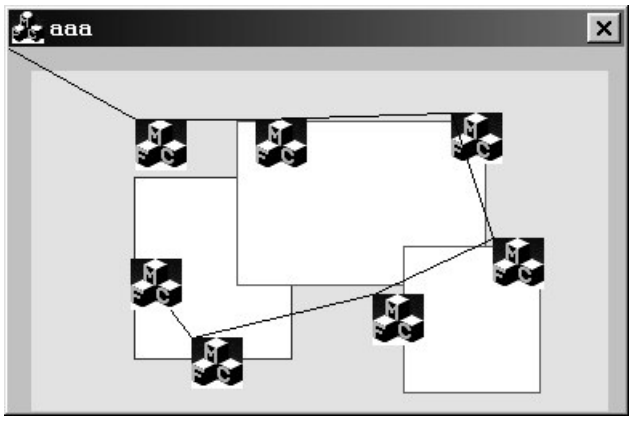

```
void CAaaDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
	CClientDC pDC(this);
	pDC.MoveTo(p);
	p.x=point.x;
	p.y = point.y;
	pDC.LineTo(point);
	HICON icono;
	CWinApp *pAplicacion=AfxGetApp();
	icono=pAplicacion->LoadIcon(IDR_MAINFRAME);
	pDC.DrawIcon(point.x,point.y,icono);
	CDialog::OnLButtonDown(nFlags, point);
}
```

4.5. Dibujar un mapa de bits sobre un diálogo

Pasos a seguir:

ponemos un control Picture sobre el diálogo

1. introducimos un bitmap en nuestros recursos
2. añadimos en propiedades del picture el bitmap que hemos insertado



4.6. CPen y CBrush

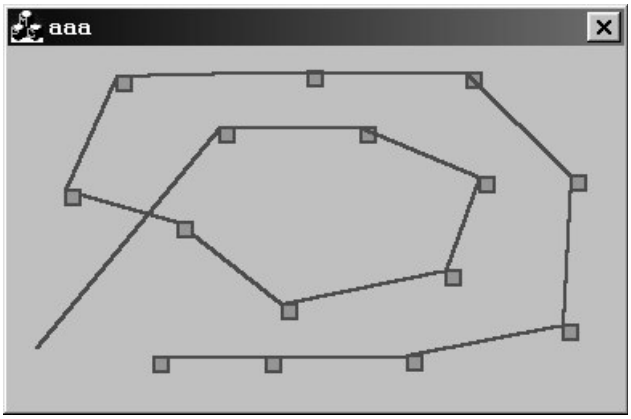
Las clases para los pinceles y fondos son:

CPen *lapiz;
CBrush *brush;

Con ellos podemos cambiar el grosor del lápiz,el tipo y los colores que vamos a utilizar para pintar. La forma de cambiarlo es mediante **SelectObject(OBJETO)**

Vamos a ver como quedaría si cambiamos el color a las lineas que dibujabamos en la aplicación anterior, y cambiamos el fondo,ahora en lugar de iconos,pondremos cuadrados.

```
void CAaaDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
        CClientDC pDC(this);
        pDC.MoveTo(p);
        p.x=point.x;
        p.y = point.y;
        CPen lapiz;
        CBrush fondo;
        fondo.CreateSolidBrush(RGB(0,255,0));
        lapiz.CreatePen(PS_SOLID,2,RGB(255,0,0));
        pDC.SelectObject(&lapiz);
        pDC.SelectObject(&fondo);
        pDC.LineTo(point);
        pDC.Rectangle(point.x,point.y,point.x +10,point.y+10);
        CDialog::OnLButtonDown(nFlags, point);
}
```



Un problema con esto es que cada vez que una pantalla, se ponga en medio de nuestra aplicación y volvamos a poner la nuestra,todo lo dibujado se borrará. Para que esto no ocurra existe el mensaje **WM_PAINT**.Aquí podremos restaurar la imagen, es el repintado de pantalla.

Si quisieramos que nuestros gráficos fueran persistentes,tendriamos que haber ido guardando cada uno de los puntos pulsados, de forma que si minimizamos la pantalla y la volvemos a maximizar, se llamará a la función **OnPaint().** Aquí será donde tendremos que volver a repintar la pantalla tal y como estaba.

4.7. La clase CDib (Dibujar mapas de bits sobre un CDC)

En el libro Visual C 5.0 aparece una clase CDib en inglés, aquí nosotros nos vamos a referir a una versión que existe en español,no creo que varien demasiado.
Es tan fácil utilizarla como esto:

```
CDib dib();
dib.LeerBm(PATHCOMPLETO);
```

Y luego para visualizarlo:

```
dib.VisualizarDib(pDC,PUNTODEORIGEN)
```

Si queremos obtener el ancho y el alto del BMP cargado: GetAncho() y GetAlto()
Tiene otras funciones, mirando su cabecera basta para saber que hacen.

Vamos a ver como quedaría en nuestra aplicación dibujando el dibujo de antes con CDib

NOTA: *La clase CDib debe estar en nuestro WorkSpace.*

Al iniciar el diálogo:

```
//CDib dib; //variable global o atributo de la clase
CClientDC dc(this);
dib.LeerBm("c:\\dibujo.bmp");
dib.VisualizarDib(&dc,CPoint(0,0));
```

Recordar poner en OnPaint lo siguiente:

```
CClientDC dc(this);
dib.VisualizarDib(&dc,CPoint(0,0));
```

Nosotros además disponemos para las aplicaciones de un clase que se llama **mapadinámico** la cual permite introducir puntos con forma de cuadrado,triángulo,círculo. Esto es lo que usa el Saida y el editor Emma. Estos puntos son sensibles al ratón, de tal forma que si se pincha sobre ellos se pueden producir acciones. Lo malo que tiene esta clase es que, esos puntos no se pueden definir dinámicamente, sino que están predefinidos al igual que el mapa en un fichero de recursos.
Esto es mejorable en tiempo de ejecución simplemente utilizando la clase CDib (en la que se basa MapaDinámico) y una clase que sea una lista de puntos, o de iconos, o de lo que queramos pintar. Así funciona Morea.

5. Los controles Active X

Un control active x (también llamado fichero OCX control OLE) es un control creado en Visual C,Visual J o Visual Basic y que puede utilizarse en cualquiera de los lenguajes.
Los controles active X , para que funcionen,tienen que estar registrados en el registro de windows. Existen dos formas de registrarlo:

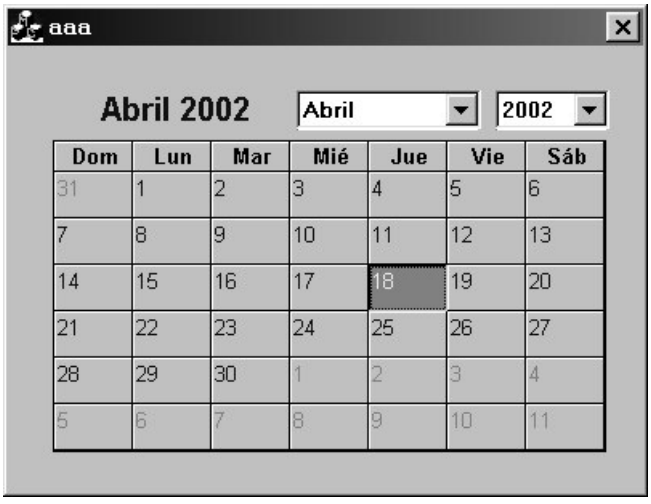
1) a mano en el registro de Windows
2) utilizando regsvr32.exe para registrar el archivo ocx (lo más usual)
   Ejemplo: regsvr32.exe mapa.ocx

Una vez hecho esto,podemos insertarlo en cualquier proyecto de Visual C,Visual J y visual Basic.

Al introducirlo en nuestro diálogo (pulsando botón derecho del ratón sobre el diálogo , insert active x) y crear una variable para él, podremos acceder a todas sus funciones públicas.

"Un active X es en realidad un programa compilado, que se ejecuta "pegado" sobre nuestra aplicación y con el cual nos podemos comunicar y decirle que haga cosas"

EJEMPLO:
1. Vamos a insertar un active X que se llama **Control Calendar 8.0**
2. Si pulsamos sobre él y llamamos a classwizard e intentamos crear una variable,se creará automáticamente una clase, con ella se producirá la comunicación entre el ocx y Visual C.



5.1. Ejemplo de creación de tu propio active X en Visual Basic

1. Creamos un proyecto en Visual Basic de tipo Control Active X
2. Le insertamos un pictureBox al que llamamos pantalla
3. Ajustamos pantalla a 360 x 1 de escala
4. Introducimos las siguientes funciones publicas

```
Dim yb As Double
Dim xb As Double

Public Sub punto(x As Double, y As Double)
   pantalla.Line (xb, yb)-(x, y), RGB(0, 0, 0)
   xb = x
   yb = y
End Sub
Public Sub borra(x As Double, y As Double)
   pantalla.Cls
End Sub

Private Sub UserControl_Initialize()
xb = 0
yb = 0
End Sub

registramos el active x después de generar el ocx
```

regsvr32 prueba.ocx

5. lo insertamos en nuestro proyecto y creamos una variable, observamos que esas funciones públicas se transforman en las siguientes (en la cabecera que se crea automáticamente)

void punto(double* x, double* y);
void borra(double* x, double* y);

```
Creamos un botón que inserte los puntos del seno.
```

6. Creamos otro botón que borre la pantalla

6. Filosofía SDI y MDI

6.1. Introducción SDI y MDI

Al crear un nuevo proyecto en Visual C con las MFCs, no da la posiblidad de crear aplicaciones multidocumento. Principalmente tenemos dos opciones:

- SDI (Single Document Interface)
- MDI(Multi-Document Interface)

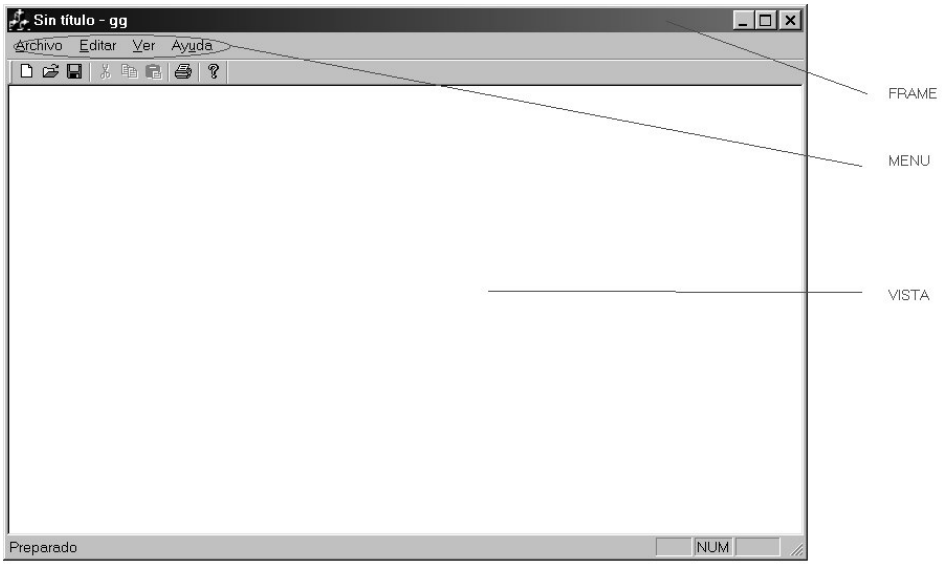
Se programa exactamente igual que lo que hemos visto hasta ahora, salvo que hay que tener en cuenta que ahora no tenemos un único diálogo si no que tenemos una arquitectura especial.

Nos va a crear los siguientes ficheros en el WorkSpace:

NombreProyectoView.cpp
NombreProyectoView.h ← VISTA

NombreProyectoDoc.cpp
NombreProyectoDoc.h ←DOCUMENTO

MainFrame.cpp
MainFrame.h ← MARCO O FRAME



Todas las aplicaciones tienen un menú (y si una barra de herramientas), un marco (que es el contenedor de todo) y una vista (que es donde se va a producir las salidas).
En cualquiera de las clases, se puede conseguir el marco principal con:

(MainFrame *)AfxGetApp()->m_pMainWnd

y a partir del marco se puede llegar a cualquiera de las cosas que contiene: el menu, el documento, la vista....

Lo más importante de la vista es la función Ondraw(CDC *pDC) que es básicamente lo mismo que la función OnPaint() del diálogo, excepto por que lo único que contiene el pDC que se le pasa es el contexto de la zona blanca (vista).

6.2. Posibilidad de tener varios formularios como vistas

Una de las cosas que nos ofrece la filosofía SDI es la posibilidad de cambiar de vista cuando nos convenga. De todos los tipos de vistas que nos ofrece Visual C CFormView, es el equivalente al Diálogo de las primeras lecciones "CDialog". Con un CFormView se puede hacer lo mismo que con un CDialog (pegar botones, pegar Edit Boxes, etc), lo único que cambia es la presentación, es decir, un Diálogo aparecerá como un objeto serparado de nuestra aplicación (menu, marco,vista) y un CformView aparecerá dentro del marco (como una vista). Además de CFormView existen otros tipos de vistas, que por ahora no vamos a ver. Todas estas vistas, heredan o derivan de la clase CView, así que todas tienen los mismos métodos que tiene CView.

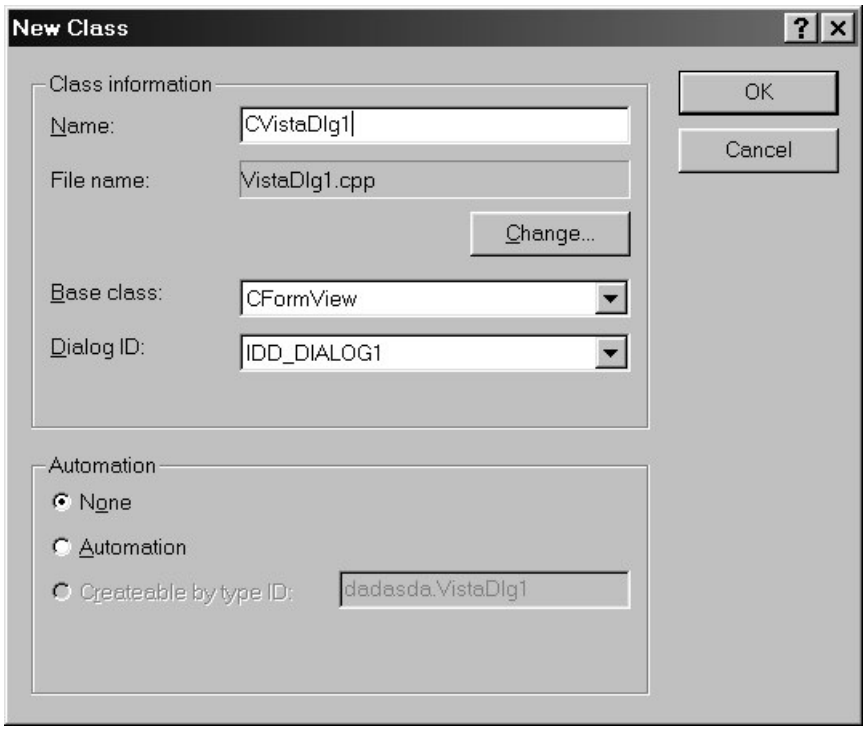
Una de las cosas que tenemos que tener en cuenta es ¿dónde se asigna la vista y cómo se asigna la vista de nuestra aplicación?. El código de cargar la primera vista (y el documento) está en el fichero NombreProyecto.cpp y es el siguiente:

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
      IDR_MAINFRAME,
      RUNTIME_CLASS(CNombreDocumento),
      RUNTIME_CLASS(CMainFrame),       // main SDI frame window
      RUNTIME_CLASS(CNombreVista));
```

Siempre se pone lo mismo (si fuera multidocumento tendría más documentos, pero ese caso no lo vamos a ver en este tema).
Como podemos observar básicamente lo que hace es crear un pDocTemplate que contiene el marco, el documento y la vista, y utiliza la macro RUNTIME_CLASS para crear un objeto de esa clase, por ejemplo en **`RUNTIME_CLASS(CNombreVista)`** un objeto de la clase CNombreVista.
Esto no hace falta cambiarlo para nada, pero es conveniente saber que existe.

Para nosotros introducir como vista un formulario propio que hayamos creado, tenemos que seguir los siguientes pasos:

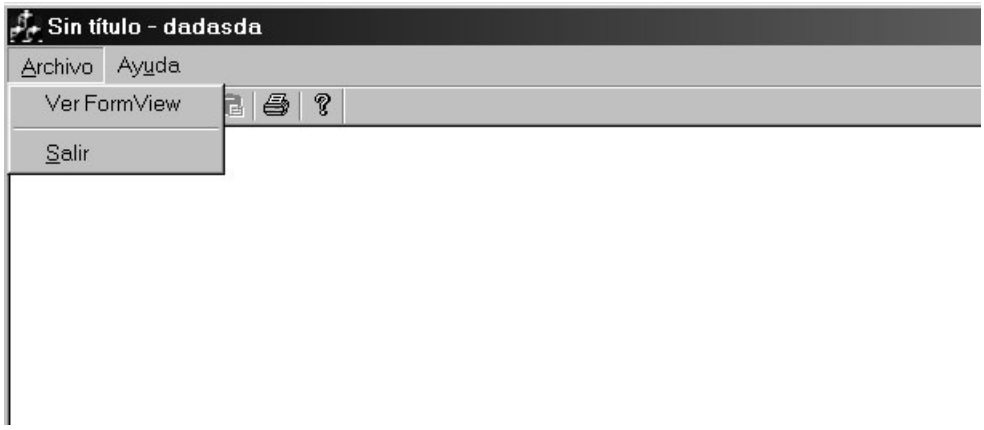
1. Insertar en los recursos un nuevo diálogo: por ejemplo IDD_DIALOG1
2. Le debemos poner como propiedades (properties) que sea:style= CHILD y en border = NONE
3. Le insertamos los controles que creamos conveniente y creamos una clase (con classwizzard) para este diálogo. En lugar de elegir que sea de tipo CDialog, le pondremos que sea de tipo CFormView.



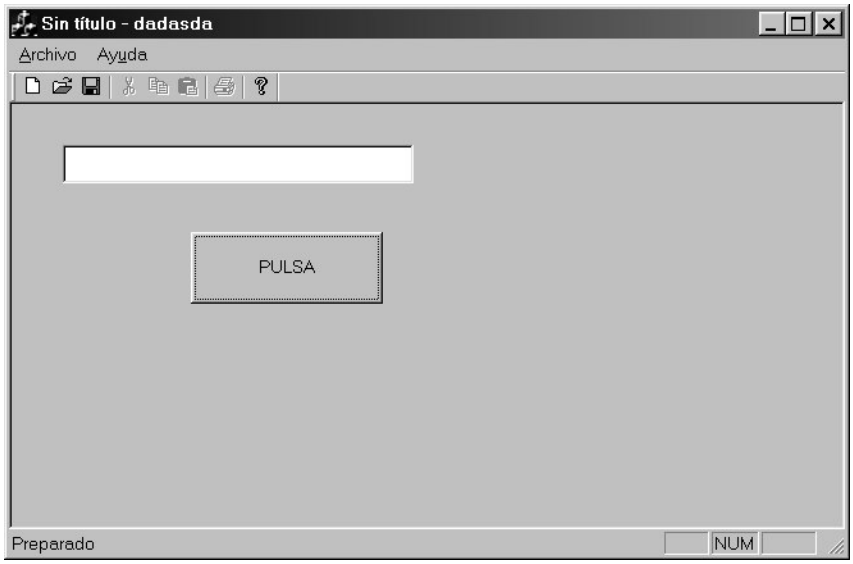
4. Para insertarlo en la aplicación basta con poner esto:
(este código es para cargarlo dando a un botón del menú)

```
CCreateContext context;
CDocument *pDoc = this->GetActiveDocument(); //obtenemos el documento
// movidas con el documento
pDoc->m_bAutoDelete=FALSE;
this->GetActiveView()->DestroyWindow();
pDoc->m_bAutoDelete=TRUE;
// fin de movidas con el documento
context.m_pNewDocTemplate=NULL;
context.m_pLastView = NULL;
// en este atributo le metemos nuestra nueva vista
context.m_pNewViewClass=RUNTIME_CLASS(CDialogo1); context.m_pCurrentDoc = pDoc;
// le metemos la nueva vista
this->SetActiveView((CView *)CreateView(&context));
// actualizamos
RecalcLayout();
```

Por ejemplo:



Introduciendo el código antes visto al pulsar en el menú sobre "Ver FormView" saldria algo así:



Una de las cosas que tendremos en cuenta es que el formulario se extenderá al tamaño de la vista, además la ventana por defecto se puede cambiar de tamaño.

Una vez creado el CFormView, se podrá hacer lo mismo con él que lo que haciamos con los Cdialogs de las lecciones primeras.

7. Código Útil y resolución de problemas

7.1. Lanzar una aplicación desde Visual C

En modo consola basta con utilizar :

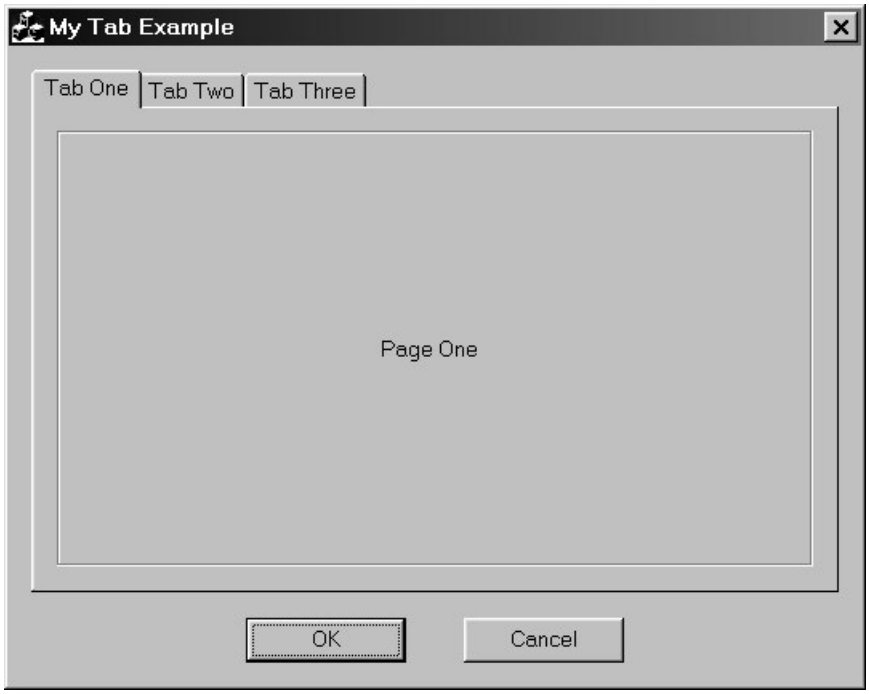
system("t :\\.....\\aplicación.exe");

esto es un poco engorroso, sobre todo si lo que queremos es lanzar una aplicación windows como el notepad o algo así. Con esta solución nos saldria la tipica ventanita de MS-DOS.
La solución a esto es el siguiente código:

```
 STARTUPINFO         StartupInfo;
PROCESS_INFORMATION ProcessInfo;
 memset(&StartupInfo, 0, sizeof(StartupInfo));
 memset(&ProcessInfo, 0, sizeof(ProcessInfo));
 // Set size
 StartupInfo.cb = sizeof(StartupInfo);
 if(CreateProcess("c:\\notepad.exe",NULL,
    // Application parameters
    NULL,
    NULL,
    FALSE ,
    NORMAL_PRIORITY_CLASS,
    NULL,
    NULL,
    &StartupInfo,
    &ProcessInfo) == FALSE) ; // Could not launch application
```

7.2. El control CTabCtrl

CtabCtrl es un control muy útil,sin embargo un poco difícil de utilizar en Visual C ya que hay que modificar una clase derivada para poder utilizarlo.



Pasos a seguir:

1. Insertamos un Control Tab en nuestro dialogo
2. Creamos una nueva clase con classwizard que derive de CTabCtrl, por ejemplo que se llame CMiTabCtrl.
3. Añadamos a esta nueva clase un nuevo atributo

   CDialog *paginas[NUMERO_TABS]

   El numero de tabs, será el numero de paginas que le tenemos pensado introducir

4. En el constructor hacemos los news para cada una de las tabs(para cada una de las tabs tendremos que tener a parte creados dialogos correspondientes con lo que querramos que salga)
5. En el destructor los deletes oportunos
6. Creamos un nuevo método

   void Inicializacion()

   que contendrá el siguiente código

```
paginas[0]->Create(IDD_IDENTIFICADOR del dialogo correspondiente,
                   this);

paginas[1]->Create(IDD_IDENTIFICADOR del dialogo correspondiente,
                   this);
paginas[3]->Create(IDD_IDENTIFICADOR del dialogo correspondiente,
                   this);

..... para todas
```

7. También habrá que ajustar los tamaños de los diálogos, lo usual es crear otro método y llamarlo tras la inicialización.

```
CRect tabRect;
CRect itemRect;
int nx,ny,nxc,nyc;
GetClientRect(&tabRect);
GetItemRect(0,&itemRect);
nx=itemRect.left;
ny=itemRect.bottom +1;
nxc=tabRect.right –itemRect.left-1;
nyc=tabRect.bottom-ny-1;
paginas[0]->SetWindowPos(&wndTop,nx,nymnxc,nyc,SWP_SHOWWINDOW);
paginas[1]->SetWindowPos(&wndTop,nx,nymnxc,nyc,SWP_HIDEWINDOW);
paginas[2]->SetWindowPos(&wndTop,nx,nymnxc,nyc,SWP_HIDEWINDOW);
.......
........
```

NOTA: Uno de los problemas que tiene este tipo de tab control es que no funciona en Windows 2000 y NT si lo introducimos en un CFormView (en windows 98 funciona perfectamente). Si se quiere usar un control tab en un CFormView lo mejor es usar otro control tab, como por ejemplo: CXTabCtrl (código y ejemplos disponibles en www.codeguru.com).

7.3. La barra de progreso (CProgressCtrl)

```
CprogressCtrl *barra;
```
barra=(CprogressCtrl *)GetDlgItem(IDC_NOMBRE);

barra->SetRange(nlower,nupper) = introduce el rango de valores
barra->GetRange(nlower,nupper) = obtiene el rango de valores
barra->GetPos() = obtiene la posición actual
barra->SetPos(n) = introduce la posición actual
barra->SetStep(n) = introduce el incremento según el cual queremos que se produzca el progreso
barra->StepIt() = se produce un aumento según el tamaño de paso que le hemos puesto

7.4. Botones que cambian de forma en tiempo de ejecución

CButton *boton = (CButton *)GetDlgITem(IDC_BOTON);
boton->SetButtonStyle(n);

Con n= 0 boton normal
Con n= 2 check box
Con n= 4 radio button

7.5. Hacer que cualquier control aparezca y desaparezca(incluso los active x)

Obtenemos el objeto y utilizamos **ShowWindow(N);**

N= 0 → desaparece
N=1 →aparece

7.6. Es posible insertar un formulario de Visual Basic en un proyecto Visual C

Una forma de crear diálogos es creandolos en Visual Basic y depues insertandolos con Visual C. Visual C los transforma automáticamente en diálogos en C. Lo malo es que el código asociado en Visual Basic se pierde, lo único que copia es la estructura visual del formulario.

7.7. Las cabeceras que se crean con un Active X ¿cuál es su significado y por qué no se crean nuevas cuando modifico el Active X?

Una vez hemos insertado un active X en un proyecto y se ha creado unas cabeceras, si en nuestro active X decidimos incluir nuevas funciones públicas, no es posible que desde el mismo proyecto se creen automáticamente nuevas cabeceras, siempre se crearán las mismas ¿por qué? Ni idea.
Hay 3 tipos de solución:

1) A capón: nos cargamos el proyecto y hacemos uno nuevo, insertamos el active X y tenemos nuevas cabeceras.
2) Creamos un Workspace nuevo con un diálogo,introducimos el active X, se crean los nuevos ficheros de cabecera y cpp del active X. Los copiamos a nuestro Workspace.
3) Modificamos las cabeceras a pelo... no es tan dificil

Para modificar las cabeceras sólo hay que fijarse un poco en como se realiza la comunicación con el OCX.

EJEMPLO: Supongamos que tenemos un control OLE punto.ocx que tiene una función pública que se llama setx(x) cuya cabecera y contenido es:

```
_punto::setx(double *x)
{
      static BYTE parms[]=VTS_PR8;
```

```
        InvokeHelper(0x60030000,DISPATCH_METHOD,VT_EMPTY,NULL,parms,x);
}
```

Lo señalado en negrita indica las cosas que variarán de una función pública a otra.
Si lo que hemos es creado otra función en el ocx que se llama sety(y) pondremos lo siguiente:

```
_punto::sety(double *y)
{
        static BYTE parms[]=VTS_PR8;
        InvokeHelper(0x60030001,DISPATCH_METHOD,VT_EMPTY,NULL,parms,y);
}
```

como vemos el primer parámetro no se ha modificado, se trata del identificador de tipo (al ser el mismo tipo no se cambia), por ejemplo si fuera un entero sería VTS_PI2. Una forma de verlo es creando distintas funciones con distintos parámetros y ver que es lo que sale en las cabecereras.

El siguente parámetro, se incrementa (siguiente función pública).
El último parámetro es el puntero a la variable de entrada.

7.8. Operaciones sobre Menus

En una aplicación de diálogo, con un menú asociado podemos realizar operaciones sobre los items de ese menú. Esta operaciones pueden ser:

1) Obtención del menú

   CMenu menu =GetMenu();

2) Añadir una nueva entrada a ese menú

   menu->AppendMenu(MF_STRING,NUMERO_ID,STRING)

donde NUMERO_ID será el valor numérico con el que identificaremos el item y STRING será el nombre que aparecerá en el menú.

3) Habilitar,Deshabilitar,poner en gris

   aux=menu->GetSubMenu(n); // cogemos el submenu n (0,1,2,3....)
   aux->EnableMenuItem(ID_ITEM,OPCION);

el ID_ITEM será el identificador del ITEM que queremos modificar y la opción puede ser una o una combinación de las siguientes:

   MF_ENABLED = lo pone activado
   MF_GRAYED = lo pone en gris
   MF_DISABLED = lo desactiva

4) Marcar y desmarcar
   aux=menu->GetSubMenu(n);
   aux->CheckMenuItem(ID_ITEM,OPCION);

OPCIONES:
   MF_CHECKED = marca el item
   MF_UNCHECKED = desmarca el item

5) Ver el estado del Item:

OPCION GetMenuState(ID_ITEM,MF_BYCOMMAND)

Donde opción puede ser cualquiera de las antes vistas y ademas MF_SEPARATOR (es una raya separadora) y alguno más

Existe un método que nos proporciona el menú de sistema (aquel en el que podemos restaurar,minimizar,maximizar,cerrar... la ventana) este es: GetSystemMenu() que nos devuelve un puntero a dicho menú.

7.9. Cosas a tener en cuenta con el menu que viene por defecto en una aplicación SDI o MDI

Para poder marcar los menus, en una aplicación SDI es necesario tener en cuenta una variable, esta es:

m_bAutoMenuEnable = false;

Cada vez que queramos acceder a un menu y modificar uno de los items, debemos ponerlo a false, de esta forma evitaremos que despues de cabiar por ejemplo un item a MF_GRAYED, Windows lo vuelva a poner en MF_ENABLED .
Esta variable se encuentra en CMainFrame, por lo que si queremos acceder a ella desde otro sitio que no sea el marco de ventana tendremos que hacer algo como esto:

```
CMainFrame *main=(CMainFrame *)AfxGetApp()->m_pMainWnd;
main->m_bAutoMenuEnable = false;
... operaciones con menus....
```

7.10. Botones con iconos dentro.

Una cosas muy usual es introducir botones con iconos dentro, como por ejemplo un botón con una carpeta abierta que indique que al pulsar sobre el se va a buscar un archivo. El proceso a seguir es mecánico:

1. Introducir un botón en el diálogo
2. Introducir el icono en los recursos y darle un nombre
3. En propiedades del botón, ponerle que es de estilo Icon (Styles)
4. Cuando se inicie el diálogo en el que se encuentra el botón o si se trata de un CformView, cuando se cree o muestre, introducir lo siguiente:

```
CButton *boton = (CButton *) GetDlgItem(IDC_BUTTON);
HICON icono;
icono = AfxGetApp()->LoadIcon(IDI_carpeta);
boton->SetIcon(icono);
```

donde IDC_BUTTON es el identificador del botón e IDI_carpeta es el identificador del icono que vamos a meter. Quedaria algo asi:

7.11. Conseguir que una ventana SDI o MDI tenga un tamaño máximo del que no pueda pasar.

Para ello se usa una variable que se encuentra en CMainFrame. Si queremos que tenga unas dimensiones máximas de 100 x 300 bastaria con escribir:

```
lpMMI->ptMaxTrackSize.x =100;
lpMMI->ptMaxTrackSize.y =300;
```

lpMMI es una variable que tiene otras propiedades interesantes y útiles a parte de ésta (VER MSDN).

7.12. CFormView, problemas.

En ocasiones un CformView puede dar problemas, aquí recogemos algunos problemas y algunas soluciones:

- **Al cambiar de vista se produce una excepción**: es conveniente introducir en recursos, en strings el identificador del formulario y un comentario (cualquiera), a veces evitamos esas excepciones que no tienen mucho sentido.
- **En Windows 2000 y NT al utilizar un CTabCtrl en un CFormView se produce una excepción**:por algún motivo, esto en Windows 9x funciona, pero en Windows NT o 2000 no funciona, lo mejor es probar a utilizar otro control de pestañas, como por ejemplo CXTabCtrl (buscar en www.codeguru.com) o en el apartado

7.13. Gestión de archivos en Windows.

En ocasiones necesitamos copiar, borrar, buscar archivos o recorrer directorios en Windows ¿cómo podemos hacer este tipo de cosas?. La respuesta se encuentra en utilizar las primitivas de Windows.

- CopyFile("path del fichero que queremos copiar","fichero destino"): copia un fichero
- MoveFile"path del fichero que queremos copiar","fichero destino"): mueve un fichero
- DeleteFile("path"): borra un fichero
- Recorrer un directorio: es un poco más complicado:

```
WIN32_FIND_DATA data;
HANDLE handle;
bool ok;
handle =FindFirstFile(_TEXT("directorio que queremos recorrer"!),& data);
// por ejemplo el directorio podemos poner: "c:\*.*" o "c:\*.txt" ...
ok=(handle!=INVALID_HANDLE_VALUE);
while(ok)
{
        lista->AddString(data.cFileName);
        ok=FindNextFile(handle,&data);
}
```

Con FindFirstFile nos devuelve un puntero al primer fichero que encuentra dentro del directorio que hemos especificado, si nos devuelve INVALID_HANDLE_VALUE es que lo hemos recorrido entero:
En data se encontrará el nombre del fichero y algunos atributos más que podemos utilizar, por ejemplo en el ejemplo que mostramos estamos introduciendo los nombres en una lista.Para más información buscar en MSDN.

7.14. Obtención del directorio en el cual se ha ejecutado la aplicación.

```
char buffer[100];
GetCurrentDirectory(100,buffer);
```

Con esta función, al inicializarse la aplicación obtenemos el directorio en el cual se ha ejecutado:

7.15. Uso de la librería fstream.h para leer y escribir en ficheros.

Fácil de utilizar. Debemos incluir fstream.h y luego:

- lectura:

```
ifstream in;
char buffer[100];
in.open("archivo");
in.getline(buffer,100);
.... cada vez que hacemos getline leemos una linea
in.close(); // cerramos el fichero

si lo que queremos es detectar si es fin de fichero se puede utilizar in.EOF();
(para más funcionalidad utilizar MSDN)
```

- escritura:

```
ofstream ou;
ou.open("fichero");
ou<< variable_o_valor<<'\n';
.....
ou.close();
(para más funcionalidad utilizar MSDN)
```

7.16. Conversiones de Strings a números y de números a Strings

Tipo origen	Tipo destino	Función	Ejemplo
char *	int	int atoi(string)	int a; A= atoi("1200");
Char *	float o double	float atof(string)	Float a; A= atoi("12.00");
int	Char *	Char*itoa(valor,buffer,base);	Char buffer[100]; CString res; Base=10; res= itoa(4,buffer,10);
float	Char *	sprintf(string,"%f",valor)	Char buffer[100]; sprintf(buffer,"%f",1.5);

7.17. Visualizar un fichero en un CEditBox.

```
char *buffer;
CFile *f = new CFile();
f->open(fichero, modo);
buffer=(char ) malloc((f->leng+1)char);
buffer[f->GetLenght() + 1]= 0; // importante
// los modos : CFile::modeCreate     CFile::modeNoTruncate       CFile::modeRead .....
f->Read(buffer,f->GetLength);
<<puntero al edit box>>->SetWindowText(buffer);
free(buffer);
```

T

V

W