

Autor: Salvador Pozo Coronado e-mail: salvador@conclase.net

Trascripción del curso alojado en la página: <http://winapi.conclase.net/>

Nota: existen varios enlaces a páginas que no existen, poco a poco se irán completando todas las páginas, disculpad las molestias.

Tabla de contenido

Tabla de contenido.....	2
Independencia de la máquina.....	14
Recursos.....	14
Ventanas.....	15
Eventos.....	15
Proyectos.....	16
Convenciones.....	16
Controles.....	17
Capítulo 1 Componentes de una ventana.....	18
El borde de la ventana.....	18
Barra de título.....	18
Caja de minimizar.....	18
Caja de maximizar.....	18
Caja de cerrar.....	18
Caja de control de menú.....	18
Menú.....	18
Barra de menú.....	19
Barra de scroll horizontal.....	19
Barra de scroll vertical.....	19
El área de cliente.....	19
Capítulo 2 Notación Húngara.....	20
Ejemplos:.....	20
Capítulo 3 La función "WinMain".....	21
Parámetros de entrada de "WinMain".....	21
Función WinMain típica.....	21
Declaración.....	22
Inicialización.....	22
Bucle de mensajes.....	23
Capítulo 4 El procedimiento de ventana.....	25
Sintaxis.....	25
Prototipo de procedimiento de ventana.....	25
Implementación de procedimiento de ventana simple.....	25
Primer ejemplo de programa Windows.....	26
Capítulo 5 Menús 1.....	27
Usando las funciones para inserción ítem a ítem:.....	27
Uso básico de MessageBox:.....	29
Respondiendo a los mensajes del menú:.....	29
Ejemplo de programa Windows con menú.....	30
Ficheros de recursos:.....	30
Cómo usar los recursos de menú:.....	32
Capítulo 6 Diálogo básico.....	34

Ficheros de recursos:	34
Procedimiento de diálogo:	37
Sintaxis.....	37
Prototipo de procedimiento de diálogo	38
Implementación de procedimiento de diálogo para nuestro ejemplo	38
Capítulo 7 Control básico Edit.....	41
Fichero de recursos	41
El procedimiento de diálogo y los controles edit.....	42
Variables a editar en los cuadros de diálogo.....	43
Iniciar controles edit	44
Devolver valores a la aplicación	44
Añadir la opción de cancelar	45
Fichero de recursos para editar enteros.....	46
Variables a editar en los cuadros de diálogo.....	47
Iniciar controles edit de enteros	47
Devolver valores a la aplicación	48
Capítulo 8 Control básico ListBox	49
Ficheros de recursos.....	49
Iniciar controles listbox	50
Devolver valores a la aplicación	51
Capítulo 9 Control básico Button	53
Ficheros de recursos.....	53
Iniciar controles button	54
Tratamiento de acciones de los controles button	55
Capítulo 10 Control básico Static	56
Ficheros de recursos.....	56
Iniciar controles static	58
Tratamiento de acciones de los controles static.....	58
Capítulo 11 Control básico ComboBox.....	59
Ficheros de recursos.....	59
Iniciar controles ComboBox	60
Devolver valores a la aplicación	62
Capítulo 12 Control básico Scrollbar.....	66
Ficheros de recursos.....	67
Iniciar controles Scrollbar.....	68
Iniciar controles scrollbar: estructura SCROLLINFO.....	69
Procesar los mensajes procedentes de controles Scrollbar	70
Procesar mensajes de scrollbar usando SCROLLINFO	72
Devolver valores a la aplicación	73
Button styles:	77
Combo box styles:	80
Dialog box styles:	83
Edit styles:.....	86
List box styles:	89
Valores de nCmdShow:	93
Scroll bar styles:.....	95
Static styles:	97
Window ex_styles:.....	101
Window styles:	103

Estructura CREATESTRUCT	108
Definición:	108
Descripción:	108
Observaciones:	109
Estructura MINMAXINFO	110
Definición:	110
Descripción:	110
Estructura MSG	111
Definición:	111
Descripción:	111
Estructura SCROLLINFO	112
Definición:	112
Descripción:	112
Estructura STARTUPINFO	114
Definición:	114
Descripción:	114
Observaciones:	118
Estructura WNDCLASS	119
Definición:	119
Descripción:	119
AppendMenu:	123
Sintaxis:	123
Parámetros:	123
Valor de retorno:	124
Observaciones:	124
CreateMenu:	126
Sintaxis:	126
Parámetros:	126
Valor de retorno:	126
Observaciones:	126
CreateWindow:	127
Sintaxis:	127
Parámetros:	127
Valor de retorno:	129
Observaciones:	129
DefWindowProc:	133
Sintaxis:	133
Parámetros:	133
Valor de retorno:	133
DestroyMenu:	134
Sintaxis:	134
Parámetros:	134
Valor de retorno:	134
Observaciones:	134
DialogBox:	135
Sintaxis:	135
Parámetros:	135
Valor de retorno:	135
Observaciones:	135

DialogProc:	137
Sintaxis:	137
Parámetros:	137
Valor de retorno:	137
Observaciones:	137
DispatchMessage:	139
Sintaxis:	139
Parámetros:	139
Valor de retorno:	139
Observaciones:	139
DrawMenuBar:	140
Sintaxis:	140
Parámetros:	140
Valor de retorno:	140
EndDialog:	141
Sintaxis:	141
Parámetros:	141
Valor de retorno:	141
Observaciones:	141
GetDlgItem:	142
Sintaxis:	142
Parámetros:	142
Valor de retorno:	142
Observaciones:	142
GetDlgItemInt:	143
Sintaxis:	143
Parámetros:	143
Valor de retorno:	143
Observaciones:	144
GetDlgItemText:	145
Sintaxis:	145
Parámetros:	145
Valor de retorno:	145
Observaciones:	145
GetMessage:	146
Sintaxis:	146
Parametros:	146
Valor de retorno:	146
Observaciones:	146
GetScrollInfo:	148
Sintaxis:	148
Parámetros:	148
Valor de retorno:	149
Observaciones:	149
GetScrollPos:	150
Sintaxis:	150
Parámetros:	150
Valor de retorno:	150
Observaciones:	150

GetScrollRange:	152
Sintaxis:	152
Parámetros:	152
Valor de retorno:	152
Observaciones:	153
InsertMenu:	154
Sintaxis:	154
Parámetros:	154
Valor de retorno:	155
Observaciones:	155
LoadMenu:	158
Sintaxis:	158
Parámetros:	158
Valor de retorno:	158
Observaciones:	158
MessageBox:	159
Sintaxis:	159
Parámetros:	159
Valor de retorno:	163
Observaciones:	163
PostMessage:	164
Sintaxis:	164
Parámetros:	164
Valor de retorno:	164
Observaciones:	164
PostQuitMessage:	166
Sintaxis:	166
Parámetros:	166
Valor de retorno:	166
Observaciones:	166
RegisterClass:	167
Sintaxis:	167
Parámetros:	167
Valor de retorno:	167
Observaciones:	167
SendDlgItemMessage:	168
Sintaxis:	168
Parámetros:	168
Valor de retorno:	168
Observaciones:	168
SendMessage:	169
Sintaxis:	169
Parámetros:	169
Valor de retorno:	169
Observaciones:	169
SetDlgItemInt:	170
Sintaxis:	170
Parámetros:	170
Valor de retorno:	170

Observaciones:.....	170
SetDlgItemText:.....	171
Sintaxis:	171
Parámetros:	171
Valor de retorno:	171
Observaciones:.....	171
SetFocus:.....	172
Sintaxis:	172
Parámetros:	172
Valor de retorno:.....	172
Observaciones:.....	172
SetMenu:.....	173
Sintaxis:	173
Parámetros:	173
Valor de retorno:.....	173
Observaciones:.....	173
SetScrollInfo:	174
Sintaxis:	174
Parámetros:	174
Valor de retorno:.....	175
Observaciones:.....	175
SetScrollPos:.....	176
Sintaxis:	176
Parámetros:	176
Valor de retorno:.....	176
Observaciones:.....	177
SetScrollRange:	178
Sintaxis:	178
Parámetros:	178
Valor de retorno:.....	178
Observaciones:.....	179
ShowWindow:	180
Sintaxis:	180
Parámetros:	180
Valor de retorno:.....	180
Observaciones:.....	180
TranslateMessage:	181
Sintaxis:	181
Parámetros:	181
Valor de retorno:.....	181
Observaciones:.....	181
WindowProc:	182
Sintaxis:	182
Parámetros:	182
Valor de retorno:.....	182
Observaciones:.....	182
WinMain:	183
Sintaxis:	183
Parámetros:	183

Valor de retorno:	183
Observaciones:	183
LOWORD	185
Definición:	185
Descripción:	185
Valor de retorno:	185
Observaciones:	185
MAKEINTRESOURCE	186
Definición:	186
Descripción:	186
Valor de retorno:	186
Observaciones:	186
Mensaje CB_ADDSTRING	188
Definición:	188
Descripción:	188
Valor de retorno:	188
Observaciones:	188
Mensaje CB_FINDSTRING	189
Definición:	189
Descripción:	189
Valor de retorno:	189
Observaciones:	189
Mensaje CB_FINDSTRINGEXACT	190
Definición:	190
Descripción:	190
Valor de retorno:	190
Observaciones:	190
Mensaje CB_GETCURSEL	191
Definición:	191
Descripción:	191
Valor de retorno:	191
Mensaje CB_GETLBTEXT	192
Definición:	192
Descripción:	192
Valor de retorno:	192
Observaciones:	192
Mensaje CB_GETLBTEXTLEN	193
Definición:	193
Descripción:	193
Valor de retorno:	193
Observaciones:	193
Mensaje CB_SELECTSTRING	194
Definición:	194
Descripción:	194
Valor de retorno:	194
Observaciones:	194
Mensaje EM_LIMITTEXT	195
Definición:	195
Descripción:	195

Valor de retorno:	195
Observaciones:	195
Mensaje LB_ADDSTRING	196
Definición:	196
Descripción:	196
Valor de retorno:	196
Observaciones:	196
Mensaje LB_GETCURSEL	197
Definición:	197
Descripción:	197
Valor de retorno:	197
Observaciones:	197
Mensaje LB_GETTEXT	198
Definición:	198
Descripción:	198
Valor de retorno:	198
Observaciones:	198
Mensaje LB_GETTEXTLEN	199
Definición:	199
Descripción:	199
Valor de retorno:	199
Observaciones:	199
Mensaje LB_SELECTSTRING	200
Definición:	200
Descripción:	200
Valor de retorno:	200
Observaciones:	200
Mensaje SBM_GETPOS	202
Definición:	202
Descripción:	202
Valor de retorno:	202
Mensaje SBM_GETRANGE	203
Definición:	203
Descripción:	203
Valor de retorno:	203
Mensaje SBM_GETSCROLLINFO	204
Definición:	204
Descripción:	204
Valor de retorno:	204
Mensaje SBM_SETPOS	205
Definición:	205
Descripción:	205
Valor de retorno:	205
Observaciones:	205
Mensaje SBM_SETRANGE	206
Definición:	206
Descripción:	206
Valor de retorno:	206
Observaciones:	206

Mensaje SBM_SETSCROLLINFO.....	207
Definición:	207
Descripción:	207
Valor de retorno:	207
Mensaje WM_CHAR	208
Definición:	208
Descripción:	208
Valor de retorno:	208
Observaciones:	209
Mensaje WM_COMMAND	210
Definición:	210
Descripción:	210
Valor de retorno:	210
Observaciones:	210
Mensaje WM_CREATE	211
Definición:	211
Descripción:	211
Valor de retorno:	211
Mensaje WM_DESTROY	212
Definición:	212
Descripción:	212
Valor de retorno:	212
Observaciones:	212
Mensaje WM_GETMINMAXINFO	213
Definición:	213
Descripción:	213
Valor de retorno:	213
Observaciones:	213
Mensaje WM_GETTEXT	214
Definición:	214
Descripción:	214
Valor de retorno:	214
Acción por defecto:	214
Observaciones:	214
Mensaje WM_GETTEXTLENGTH.....	215
Definición:	215
Descripción:	215
Valor de retorno:	215
Acción por defecto:	215
Observaciones:	215
Mensaje WM_HSCROLL	217
Definición:	217
Descripción:	217
Valor de retorno:	217
Observaciones:	218
Mensaje WM_INITDIALOG	219
Definición:	219
Descripción:	219
Valor de retorno:	219

Comentarios:	219
Mensaje WM_KEYDOWN	220
Definición:	220
Descripción:	220
Valor de retorno:	220
Acción por defecto:	221
Observaciones:	221
Mensaje WM_KEYUP	222
Definición:	222
Descripción:	222
Valor de retorno:	223
Acción por defecto:	223
Observaciones:	223
Mensaje WM_NCCREATE	224
Definición:	224
Descripción:	224
Valor de retorno:	224
Acción por defecto:	224
Mensaje WM_NCPAINT	225
Definición:	225
Descripción:	225
Valor de retorno:	225
Acción por defecto:	225
Observaciones:	225
Mensaje WM_PAINT	226
Definición:	226
Descripción:	226
Valor de retorno:	226
Acción por defecto:	226
Observaciones:	226
Mensaje WM_QUIT	228
Definición:	228
Descripción:	228
Valor de retorno:	228
Mensaje WM_SETTEXT	229
Definición:	229
Descripción:	229
Valor de retorno:	229
Acción por defecto:	229
Observaciones:	229
Mensaje WM_SYSCHAR	230
Definición:	230
Descripción:	230
Valor de retorno:	230
Observaciones:	231
Mensaje WM_SYSCOMMAND	232
Definición:	232
Descripción:	232
Valor de retorno:	233

Observaciones:.....	233
Mensaje WM_SYSDEADCHAR	235
Definición:	235
Descripción:	235
Valor de retorno:	236
Observaciones:.....	236
Mensaje WM_SYSKEYDOWN.....	237
Definición:	237
Descripción:	237
Valor de retorno:	238
Acción por defecto:.....	238
Observaciones:.....	238
Mensaje WM_SYSKEYUP	239
Definición:	239
Descripción:	239
Valor de retorno:	240
Acción por defecto:.....	240
Observaciones:.....	240
Mensaje WM_TIMER	241
Definición:	241
Descripción:	241
Valor de retorno:	241
Observaciones:.....	241
Mensaje WM_VSCROLL	242
Definición:	242
Descripción:	242
Valor de retorno:	242
Observaciones:.....	243
Atributos comunes de recursos	245
Atributos de carga	245
Atributos de memoria	245
Sentencia CAPTION.....	247
Parametro:	247
Sentencia CHARACTERISTICS	248
Parametro:	248
Sentencia CLASS	249
Parámetros:	249
Observaciones:.....	249
CONTROL: Controles Generales.....	250
Parámetros:	250
Control de clase Button.....	251
Control de clase Combobox.....	251
Control de clase Edit.....	252
Control de clase Listbox	252
Control de clase Scrollbar.....	252
Control de clase Static	253
DIALOG	254
Definición:	254
Descripción:	254

Observaciones:.....	255
Sentencia EXSTYLE	256
Parámetros:	256
Sentencia LANGUAGE.....	257
Parámetros:	257
MENU.....	258
Definición:	258
Descripción:	258
Sentencia MENUITEM	259
Parámetros:	259
Parámetros Comunes de Sentencias	260
Parámetros de los controles comunes	260
Parámetros:	260
Sentencia POPUP	262
Parámetros:	262
Sentencia STYLE	263
Parámetro:.....	263
Comentarios:.....	265
Sentencia VERSION.....	266
Parámetro:.....	266
Glosario.....	267
API (Application Programming Interface).	267
OWL y MFC	267
GDI (Graphics Device Interface).....	267
SDK (Software Development Kit).....	267
MAPI (Messaging Application Programming Inerface).	267
MDI (Multiple Document Interface).	268
SDI (Single Document Interface).	268
GUI (Graphic User Interface)	268
OEM (original equipment manufacturer)	268

Introducción

Requisitos previos:

Para el presente curso supondré que estás familiarizado con la programación en C y C++ y también con las aplicaciones y el entorno Windows, al menos al nivel de usuario. Pero no se requerirán muchos más conocimientos.

El curso pretende ser una explicación de la forma en que se realizan los programas en Windows usando el API. Las explicaciones de las funciones y los mensajes del API son meras traducciones del fichero de ayuda de WIN32 de Microsoft, y sólo se incluyen como complemento.

Vamos a ponernos en antecedentes. Primero veamos algunas características especiales de la programación en Windows.

Independencia de la máquina

Los programas Windows son independientes de la máquina en la que se ejecutan (o deberían serlo), el acceso a los dispositivos físicos se hace a través de interfaces, y nunca se accede directamente a dispositivos físicos. Esta es una de las principales ventajas para el programador, no hay que preocuparse por el modelo de tarjeta gráfica o de impresora, la aplicación funcionará con todas, y será el sistema operativo el que se encargue de que así sea.

Recursos

Un concepto importante es el de recurso. Desde el punto de vista de Windows, un recurso es todo aquello que puede ser usado por una o varias aplicaciones. Existen recursos físicos, que son los dispositivos que componen el ordenador, como la memoria, la impresora, el teclado o el ratón y recursos virtuales o lógicos, como los gráficos, los iconos o las cadenas de caracteres.

Por ejemplo, si nuestra aplicación requiere el uso de un puerto serie, primero debe averiguar si está disponible, es decir, si existe y si no lo está usando otra aplicación; y después lo reservará para su uso. Esto es porque este tipo de recurso no puede ser compartido.

Lo mismo pasa con la memoria o con la tarjeta de sonido, aunque son casos diferentes. Por ejemplo, la memoria puede ser compartida, pero de una forma general, cada porción de memoria no puede compartirse, y se trata de un recurso finito. Las tarjetas de sonido, dependiendo del modelo, podrán o no compartirse por varias aplicaciones. Otros recursos como el ratón y el teclado también se comparten, pero se asigna su uso automáticamente a la aplicación activa, a la que normalmente nos referiremos como la que tiene el "foco", es decir, la que mantiene contacto con el usuario.

Desde nuestro punto de vista, como programadores, también consideramos recursos varios componentes como los menús, los iconos, los cuadros de diálogo, las cadenas de caracteres, los mapas de bits, los cursores, etc. En sus programas, el Windows almacena separados el código y los recursos, dentro del mismo fichero, y estos últimos pueden ser editados por separado, permitiendo por ejemplo, hacer versiones de los programas en distintos idiomas sin tener acceso a los ficheros fuente de la aplicación.

Ventanas

La forma en que se presentan las aplicaciones Windows (al menos las interactivas) ante el usuario es la ventana, supongo que todos sabemos qué es una ventana: un área rectangular de la pantalla que se usa de interfaz entre la aplicación y el usuario.

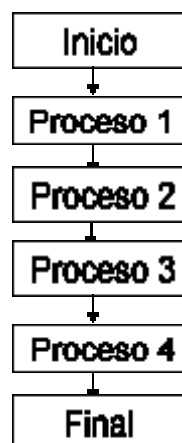
Cada aplicación tiene al menos una ventana, la ventana principal, y todas las comunicaciones entre usuario y aplicación se canalizan a través de una ventana. Cada ventana comparte el espacio de la pantalla con otras ventanas, incluso de otras aplicaciones, aunque sólo una puede estar activa, es decir, sólo una puede recibir información del usuario.

Eventos

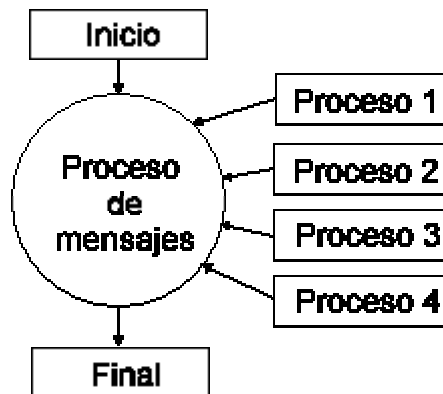
Los programas en Windows están orientados a eventos, esto significa que normalmente los programas están esperando a que se produzca un acontecimiento que les incumba, y mientras tanto permanecen aletargados o dormidos.

Un evento puede ser por ejemplo, el movimiento del ratón, la activación de un menú, la llegada de información desde el puerto serie, una pulsación de una tecla...

Esto es así, porque Windows es un sistema operativo multitarea, y el tiempo del microprocesador ha de repartirse entre todos los programas que se estén ejecutando. Si los programas fueran secuenciales puros, esto no sería posible, ya que hasta que una aplicación finalizara, el sistema no podría atender al resto.



Ejemplo de programa secuencial:



Ejemplo de programa por eventos:

Proyectos

Debido a la complejidad de los programas Windows, normalmente los dividiremos en varios ficheros fuente, que compilaremos por separado y enlazaremos juntos.

Cada compilador puede tener diferencias, más o menos grandes, para trabajar con proyectos. Sin embargo no deberías tener grandes dificultades para adaptarte a cada uno.

En el presente curso trabajaremos con el compilador de "Bloodshed", que es público y gratuito, y puede descargarse de Internet en la siguiente URL:

<http://www.bloodshed.net/>.

Para crear un proyecto Windows usando este compilador elijeremos el menú "File/New Project". Se abrirá un cuadro de diálogo donde podremos elegir el tipo de proyecto. Elegiremos "Windows Application" y "C++ Project". A continuación pulsamos "Aceptar".

El compilador crea un proyecto con un fichero C++, con el esqueleto de una aplicación para una ventana, a partir de ahí empieza nuestro trabajo.

Convenciones

En parte para que no te resulte muy difícil adaptarte a la terminología de Windows, y a la documentación existente, y en parte para seguir mi propia costumbre, en la mayoría de los casos me referiré a componentes y propiedades de Windows con sus nombres en inglés. Por ejemplo, hablaremos de "button", "check box", "radio button", "list box", "combo box" o "property sheet", aunque algunas veces traduzca sus nombre a castellano, por ejemplo, "dialog box" se nombrará a menudo como "cuadro de diálogo".

Además hablaremos a menudo de ventanas "overlapped" o superponibles, que son las ventanas corrientes. Para el término "pop-up" he desistido de buscar una traducción.

También se usaran a menudo, con relación a "check boxes", términos ingleses como checked, unchecked o grayed, en lugar de marcado, no marcado o gris.

Owner-draw, es un estilo que indica que una ventana o control no es la encargada de actualizarse en pantalla, esa responsabilidad es transferida a la ventana dueña del control o ventana.

Para "bitmap" se usará a menudo la expresión "mapa de bits".

Controles

Los controles son la forma en que las aplicaciones windows se comunican con el usuario. Normalmente se usan dentro de los cuadros de diálogo, pero en realidad pueden usarse en cualquier ventana.

Existen bastantes, y los iremos viendo poco a poco, al mismo tiempo que aprendemos a manejarlos.

Los más importantes y conocidos son:

- control estatic: son etiquetas, marcos, iconos o dibujos.
- control edit: permiten que el usuario introduzca datos en la aplicación.
- control list box: el usuario puede escoger entre varias opciones de una lista.
- control combo box: es una mezcla entre un edit y un list box.
- control scroll bar: barras de desplazamiento, para la introducción de valores entre márgenes definidos.
- control button: realizan acciones o comandos, de button de derivan otros dos controles muy comunes:
 - control check box: permite leer variables de dos estados "checked" o "unchecked"
 - control radio button: se usa en grupos, dentro de cada grupo sólo uno puede ser activado.

Capítulo 1 Componentes de una ventana

Veamos ahora los elementos que componen una ventana, aunque más adelante veremos que no todos tienen por qué estar presentes en todas las ventanas.

El borde de la ventana.

Hay varios tipos, dependiendo de que estén o no activas las opciones de cambiar el tamaño de la ventana. Se trata de un área estrecha alrededor de la ventana que permite cambiar su tamaño.

Barra de título.

Zona en la parte superior de la ventana que contiene el icono y el título de la ventana, esta zona también se usa para mover la ventana a través de la pantalla.

Caja de minimizar.

Pequeña área cuadrada situada en la parte derecha de la barra de título que sirve para disminuir el tamaño de la ventana. Antes de la aparición del Windows 95 la ventana se convertía a su forma icónica, pero desde la aparición del Windows 95 se elimina la ventana y sólo permanece el botón en la barra de estado.

Caja de maximizar.

Pequeña área cuadrada situada en la parte derecha de la barra de título que sirve para agrandar la ventana para que ocupe toda la pantalla. Cuando la ventana está maximizada, se sustituye por la caja de restaurar.

Caja de cerrar.

Pequeña área cuadrada situada en la parte derecha de la barra de título que sirve para cerrar la ventana.

Caja de control de menú.

Pequeña área cuadrada situada en la parte izquierda de la barra de título, normalmente contiene el icono de la ventana, y sirve para desplegar el menú del sistema.

Menú.

O menú del sistema. Se trata de una ventana especial que contiene las funciones comunes a todas las ventanas, también accesibles desde las cajas y el borde, como minimizar, restaurar, maximizar, mover, cambiar tamaño y cerrar.

Barra de menú.

Zona situada debajo de la barra de título, contiene los menús de la aplicación.

Barra de scroll horizontal.

Barra situada en la parte inferior de la ventana, permite desplazar horizontalmente la vista del área de cliente.

Barra de scroll vertical.

Barra situada en la parte derecha de la ventana, permite desplazar verticalmente la vista del área de cliente.

El área de cliente.

Es la zona donde el programador sitúa los controles, y los datos para el usuario. En general es toda la superficie de la ventana lo que no está ocupada por las zonas anteriores.

Capítulo 2 Notación Húngara

La notación húngara es un sistema usado normalmente para crear los nombres de variables cuando se programa en Windows. Es el sistema usado en la programación del sistema operativo, y también por la mayoría de los programadores. También será el sistema que usemos en este curso.

Consiste en prefijos en minúsculas que se añaden a los nombres de las variables, y que indican su tipo. El resto del nombre indica, lo más claramente posible, la función que realiza la variable.

Prefijo	Significado
b	Booleano
c	Carácter (un byte)
dw	Entero largo de 32 bits sin signo (DOBLE WORD)
f	Flags empaquetados en un entero de 16 bits
h	Manipulador de 16 bits (HANDLE)
l	Entero largo de 32 bits
lp	Puntero a entero largo de 32 bits
lpfn	Puntero largo a una función que devuelve un entero
lpsz	Puntero largo a una cadena terminada con cero
n	Entero de 16 bits
p	Puntero a entero de 16 bits
pt	Coordenadas (x, y) empaquetadas en un entero de 32 bits
rgb	Valor de color RGB empaquetado en un entero de 32 bits
sz	Cadena terminada en cero
w	Entero corto de 16 bits sin signo (WORD)

Ejemplos:

nContador: la variable es un entero que se usará como contador.

szNombre: una cadena terminada con cero que almacena un nombre.

bRespuesta: una variable booleana que almacena una respuesta.

Capítulo 3 La función "WinMain"

La función de entrada de un programa Windows es "[WinMain](#)", en lugar de la conocida "main". Normalmente, la definición de esta función cambia muy poco de una aplicaciones a otras. Se divide en tres partes claramente diferenciadas: declaración, inicialización y bucle de mensajes.

Parámetros de entrada de "WinMain"

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
```

La función WinMain tiene cuatro parámetros de entrada:

- hInstance es un manipulador para la instancia del programa que estamos ejecutando. Cada vez que se ejecuta una aplicación, Windows crea una Instancia para ella, y le pasa un manipulador de dicha instancia a la aplicación.
- hPrevInstance es un manipulador a instancias previas de la misma aplicación. Como Windows es multitarea, pueden existir varias versiones de la misma aplicación ejecutándose, varias instancias. En Windows 3.1, este parámetro nos servía para saber si nuestra aplicación ya se estaba ejecutando, y de ese modo se podían compartir los datos comunes a todas las instancias. Pero eso era antes, ya que en Win32 usa un segmento distinto para cada instancia y este parámetro es siempre NULL, sólo se conserva por motivos de compatibilidad.
- lpszCmdParam, esta cadena contiene los argumentos de entrada del comando de línea.
- nCmdShow, este parámetro especifica cómo se mostrará la ventana. Para ver sus posibles valores consultar [valores de nCmdShow](#). Se recomienda no usar este parámetro en la función [ShowWindow](#) la primera vez que se ésta es llamada. En su lugar debe usarse el valor SW_SHOWDEFAULT.

Función WinMain típica

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpszCmdParam, int nCmdShow)
{
    HWND hWnd;
    MSG Message;
    WNDCLASS WndClass;

    WndClass.style = CS_HREDRAW | CS_VREDRAW;
    WndClass.lpszWndProc = WindowProcedure;
```

```

WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = 0;
WndClass.hbrBackground = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
WndClass.hInstance = hInstance;
WndClass.lpszClassName = "NUESTRA_CLASE";
WndClass.lpszMenuName = NULL;

RegisterClass(&WndClass);

hWnd = CreateWindow(
    "NUESTRA_CLASE",
    "Ventana de Ejemplo",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    320,
    200,
    HWND_DESKTOP,
    NULL,
    hInstance,
    NULL
);

ShowWindow(hWnd, SW_SHOWDEFAULT);
while(TRUE == GetMessage(&Message, 0, 0, 0))
{
    TranslateMessage(&Message);
    DispatchMessage(&Message);
}

return Message.wParam;
}

```

Declaración

En la primera zona declararemos las variables que necesitamos para nuestra función WinMain, que como mínimo serán tres:

- **HWND** hWnd, un manipulador para la ventana principal de la aplicación. Ya sabemos que nuestra aplicación necesitará al menos una ventana.
- **MSG** Message, una variable para manipular los mensajes que lleguen a nuestra aplicación.
- **WNDCLASS** WndClass, una estructura que se usará para registrar la clase particular de ventana que usaremos en nuestra aplicación.

Inicialización

Esta zona se encargará de registrar la clase, crear la ventana y visualizarla en pantalla.

Para registrar la clase primero hay que rellenar adecuadamente la estructura [WNDCLASS](#), que define algunas características que serán comunes a todas las ventanas de una misma clase, como color de fondo, icono, menú por defecto, el procedimiento de ventana, etc. Después hay que llamar a la función [RegisterClass](#).

A continuación se crea la ventana usando la función [CreateWindow](#), que nos devuelve un manipulador de ventana.

Pero esto no muestra la ventana en la pantalla. Para que la ventana sea visible hay que llamar a la función [ShowWindow](#). La primera vez que se llama a ésta función, después de crear la ventana, se puede usar el parámetro `nCmdShow` de `WinMain` como parámetro o mejor aún, como se recomienda por Windows, el valor `SW_SHOWDEFAULT`.

Bucle de mensajes

Este es el núcleo de la aplicación, como se ve en el ejemplo el programa permanece en este bucle mientras la función [GetMessage](#) retorne con un valor `TRUE`.

```
while(GetMessage(&Message, 0, 0, 0)) {  
    TranslateMessage(&Message);  
    DispatchMessage(&Message);  
}
```

Este bucle no es recomendable, aunque se usa muy habitualmente. La razón es que la función `GetMessage` puede retornar tres valores: `TRUE`, `FALSE` o `-1`. El valor `-1` indica un error, así en este caso se debería abandonar el bucle.

Sería más apropiado un bucle como este:

```
while(TRUE == GetMessage(&Message, 0, 0, 0)) {  
    TranslateMessage(&Message);  
    DispatchMessage(&Message);  
}
```

La función [TranslateMessage](#) se usa para traducir los mensajes de teclas virtuales a mensajes de carácter. Las teclas virtuales son las teclas de función, las teclas de las flechas, y las combinaciones de teclas de caracteres o números con las teclas `ALT` y `CONTROL`.

Los mensajes traducidos se reenvían a la lista de mensajes del proceso, y se recuperarán con las siguientes llamadas a [GetMessage](#).

La función [DispatchMessage](#) envía el mensaje al procedimiento de ventana, donde será tratado adecuadamente. El próximo capítulo está dedicado al procedimiento de ventana, y al final de él estaremos en disposición de crear nuestro primer programa Windows.

Capítulo 4 El procedimiento de ventana

Cada ventana tiene una función asociada, ésta función se conoce como procedimiento de ventana, y es la encargada de procesar adecuadamente todos los mensajes enviados a una determinada clase de ventana. Es la responsable de todo lo relativo al aspecto y al comportamiento de una ventana.

Normalmente, estas funciones están basadas en una estructura "switch" donde cada "case" corresponde aun determinado tipo de mensaje.

Sintaxis

```
LRESULT CALLBACK WindowProcedure(  
    HWND hwnd,    // Manipulador de ventana  
    UINT msg,     // Mensaje  
    WPARAM wParam, // Parámetro palabra, varía  
    LPARAM lParam // Parámetro doble palabra, varía  
);
```

- hwnd es el manipulador de la ventana a la que está destinado el mensaje.
- msg es el código del mensaje.
- wParam es el parámetro de tipo palabra asociado al mensaje.
- lParam es el parámetro de tipo doble palabra asociado al mensaje.

Para más detalles sobre la función de procedimiento de ventana, consultar [WindowProc](#).

Prototipo de procedimiento de ventana

```
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM,  
LPARAM);
```

Implementación de procedimiento de ventana simple

```
/* Esta función es llamada por la función del API DispatchMessage() */  
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg,  
WPARAM wParam, LPARAM lParam)  
{  
    switch (msg)          /* manipulador del mensaje */  
    {  
        case WM_DESTROY:
```

```

        PostQuitMessage(0); /* envía un mensaje WM_QUIT a la cola de
mensajes */
        break;
        default: /* para los mensajes de los que no nos ocupamos */
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

```

En general, habrá tantos procedimientos de ventana como programas diferentes y todos serán distintos, pero también tendrán algo en común: todos ellos procesarán los mensajes que lleguen a una ventana.

En este ejemplo sólo procesamos un tipo de mensaje, se trata de [WM_DESTROY](#) que es el mensaje que se envía a una ventana cuando se recibe un comando de cerrar, ya sea por menú o mediante el icono de aspa en la esquina superior derecha de la ventana.

Este mensaje sólo sirve para informar a la aplicación de que el usuario tiene la intención de abandonar la aplicación, y le da una oportunidad de dejar las cosas en su sitio: cerrar ficheros, liberar memoria, guardar variables, etc. Incluso, la aplicación puede decidir que aún no es el momento adecuado para abandonar la aplicación. En el caso del ejemplo, efectivamente cierra la aplicación, y lo hace enviándole un mensaje [WM_QUIT](#), mediante la función [PostQuitMessage](#).



El resto de los mensajes se procesan en el caso "default", y simplemente se cede su tratamiento a la función del API que hace el proceso por defecto para cada mensaje, [DefWindowProc](#).

Este es el camino que sigue el mensaje [WM_QUIT](#) cuando llega, ya que el proceso por defecto para este mensaje es cerrar la aplicación.

En posteriores capítulos veremos como se complica paulatinamente esta función, añadiendo más y más mensajes.

Primer ejemplo de programa Windows

Ya estamos en condiciones de crear nuestro primer programa Windows, que sólo mostrará una ventana en pantalla.

Ejemplo 1: ejemplo1.c 7/12/2000 (1.779 bytes)  (Alternativo: )

Capítulo 5 Menús 1

Ahora que ya sabemos hacer el esqueleto de una aplicación Windows, veamos el primer medio para comunicarnos con ella.

Supongo que todos sabemos lo que es un menú: se trata de una ventana un tanto especial, del tipo pop-up, que contiene una lista de comandos u opciones entre las cuales el usuario puede elegir.

Cuando se usan en una aplicación, normalmente se agrupan varios menús bajo una barra horizontal, (que no es otra cosa que un menú), dividida en varias zonas o ítems.

Cada ítem de un menú que tenga asociado un comando o un valor, es decir todos menos los separadores y aquellos que despliegan nuevos menús, tiene asociado un identificador. Este valor se usa por la aplicación para saber qué opción se activó por el usuario, y decidir las acciones a tomar en consecuencia.

Existen varias formas de añadir un menú a una ventana, veremos cada una de ellas por separado.

Es posible desactivar algunas opciones para que no estén disponibles para el usuario.

Usando las funciones para inserción ítem a ítem:

Este es el sistema más rudimentario, pero como ya veremos en el futuro, en ocasiones puede ser muy útil. Empezaremos viendo éste sistema porque ilustra mucho mejor la estructura de los menús.

Tomemos el ejemplo del capítulo anterior y definamos algunas constantes:

```
#define CM_PRUEBA 100
#define CM_SALIR 101
```

Y añadamos la declaración de una función en la zona de prototipos:

```
void InsertarMenu(HWND);
```

Al final del programa añadimos la definición de esta función:

```
void InsertarMenu(HWND hWnd)
{
    HMENU hMenu1, hMenu2;

    hMenu1 = CreateMenu(); // Manipulador de la barra de menú
```

```

hMenu2 = CreateMenu(); // Manipulador para el primer menú pop-up
AppendMenu(hMenu2, MF_STRING, CM_PRUEBA, "&Prueba"); // Item 1 del
menú
AppendMenu(hMenu2, MF_SEPARATOR, 0, NULL); // Item 2 del menú
(separador)
AppendMenu(hMenu2, MF_STRING, CM_SALIR, "&Salir"); // Item 3 del
menú
AppendMenu(hMenu1, MF_STRING | MF_POPUP, (UINT)hMenu2,
"&Principal"); // Inserción del menú pop-up
SetMenu (hWnd, hMenu1); // Asigna el menú a la ventana hWnd
}

```

Y por último, sólo nos queda llamar a nuestra función, insertaremos ésta llamada justo antes de visualizar la ventana.

```

...
InsertarMenu(hWnd);
ShowWindow(hWnd, SW_SHOWDEFAULT);
...

```

Veamos cómo funciona "InsertarMenu".

La primera novedad son las variables del tipo HMENU. HMENU es un tipo de manipulador especial para menús. Necesitamos dos variables de este tipo, una para manipular la barra de menú, hMenu1. La otra para manipular cada uno de los menús pop-up, en este caso sólo uno, hMenu2.

De momento haremos una barra de menú con un único elemento que será un menú pop-up. Después veremos como implementar menús más complejos.

Para crear un menú usaremos la función [CreateMenu](#), que crea un menú vacío.

Para ir añadiendo ítems a cada menú usaremos la función [AppendMenu](#). Esta función tiene varios argumentos, el primero es el menú donde queremos insertar el nuevo ítem.

El segundo son las opciones o atributos del nuevo ítem, por ejemplo MF_STRING, indica que se trata de un ítem de tipo texto, MF_SEPARATOR, es un ítem separador y MF_POPUP, indica que se trata de un menú que desplegará un nuevo menú pop-up.

El siguiente parámetro puede tener distintos significados:

- Puede ser un identificador de comando, éste identificador se usará para comunicar a la aplicación si el usuario seleccionó un determinado ítem.
- Un manipulador de menú, si el ítem tiene el flag MF_POPUP, en éste caso hay que hacer un casting a (UINT).
- Opuede ser cero, si se trata de un separador.

El último parámetro es el texto del ítem, cuando se ha especificado el flag MF_STRING, más adelante veremos que los ítems pueden ser también bitmaps. Normalmente se trata de una cadena de texto. Pero hay una peculiaridad interesante, para indicar la tecla que activa un determinado ítem de un menú se muestra la letra correspondiente subrayada. Esto se consigue insertando un '&' justo antes de la letra que se quiere usar como atajo, por ejemplo, en el ítem "&Prueba" esta letra será la 'P'.

Por último [SetMenu](#), asigna un menú a una ventana determinada. El primer parámetro es el manipulador de la ventana, y el segundo el del menú.

Prueba estas funciones y juega un rato con ellas. En la siguiente página veremos cómo hacer que nuestra aplicación responda a los mensajes del menú.

Antes de complicar más nuestros menús, veamos cómo hacer que nuestra aplicación se de cuenta de que el usuario ha activado una opción del menú. Y para que nos lo notifique a nosotros usaremos un cuadro de mensaje.

Uso básico de MessageBox:

Antes de aprender a visualizar texto en la ventana, usaremos un mecanismo más simple para informar al usuario de cualquier cosa que pase en nuestra aplicación. Este mecanismo no es otro que el cuadro de mensaje (message box), que consiste en una pequeña ventana con un mensaje para el usuario y uno o varios botones, según el tipo de cuadro de mensaje que usemos. En nuestros primeros ejemplos, el cuadro de mensaje sólo incluirá el botón de "Aceptar".

Para visualizar un cuadro de mensaje simple, usaremos la función [MessageBox](#). En nuestros ejemplos bastará con la siguiente forma:

```
MessageBox(hWnd, "Texto de mensaje", "Texto de título", MB_OK);
```

Esto mostrará un pequeño cuadro de diálogo con el texto y el título especificados y un botón de "Aceptar". El cuadro se cerrará al pulsar el botón o al pulsar la tecla de Retorno.

Respondiendo a los mensajes del menú:

Las activaciones de los menús se reciben mediante un mensaje [WM_COMMAND](#).

Para procesar estos mensajes, cuando sólo podemos recibir mensajes desde un menú, únicamente nos interesa la palabra de menor peso del parámetro wParam del mensaje.

Modifiquemos el procedimiento de ventana para procesar los mensajes de nuestro menú:

```

LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam)
{
    switch (msg)          /* manipulador del mensaje */
    {
        case WM_COMMAND:
            switch(LOWORD(wParam)) {
                case CM_PRUEBA:
                    MessageBox(hwnd, "Comando: Prueba", "Mensaje de menú",
MB_OK);
                    break;
                case CM_SALIR:
                    MessageBox(hwnd, "Comando: Salir", "Mensaje de menú", MB_OK);
                    PostQuitMessage(0); /* envía un mensaje WM_QUIT a la cola de
mensajes */
                    break;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0); /* envía un mensaje WM_QUIT a la cola de
mensajes */
            break;
        default:          /* para los mensajes de los que no nos ocupamos */
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

```

Sencillo, ¿no?.

Observa que hemos usado la macro [LOWORD](#) para extraer el identificador del ítem del parámetro wParam. Después de eso, todo es más fácil.

También se puede ver que hemos usado la misma función para salir de la aplicación que para el mensaje [WM_DESTROY](#): [PostQuitMessage](#).

Ejemplo de programa Windows con menú

Este ejemplo contiene todo lo que hemos visto sobre los menús hasta ahora.

Ejemplo 2: ejemplo2.c 18/12/2000 (2.750 bytes)



(Alternativo:



Veamos ahora una forma más sencilla y más habitual de implementar menús.

Ficheros de recursos:

Lo normal es implementar los menús desde un fichero de recursos, el sistema que hemos visto sólo se usa en algunas ocasiones, para crear o modificar menús durante la ejecución de la aplicación.

Es importante adquirir algunas buenas costumbres cuando se trabaja con ficheros de recursos.

1. Usaremos siempre etiquetas como identificadores para los ítems de los menús.
2. Crearemos un fichero de cabecera con las definiciones de los identificadores, en nuestro ejemplo se llamará "ids.h".
3. Incluiremos éste fichero en el fichero de recursos y en el código fuente de nuestra aplicación.

Partimos de un proyecto nuevo: win3. Pero usaremos el código modificado del ejemplo1.

Para ello creamos un nuevo proyecto de tipo GUI, al que llamaremos Win3, y copiamos el contenido de "ejemplo1.c" en el fichero "untitled1", al que renombraremos como "ejemplo3.c".

A continuación crearemos el fichero de identificadores.

Añadimos el fichero de cabecera a nuestro proyecto. Si estás usando Dev-C++, ésto se hace pulsando con el botón derecho del ratón sobre el nodo del proyecto y eligiendo el ítem de "new item in project" en el menú pop-up que se despliega.

Después lo renombramos, el mecanismo es similar, pulsamos con el botón derecho sobre el ítem "untitled2" y elegimos la opción de "rename file" del menú pop-up que se despliega. Como nuevo nombre elegimos: "ids.h".

Pinchando sobre el ítem del nuevo fichero éste se abrirá. Introducimos los identificadores:

```
#define CM_PRUEBA 100  
#define CM_SALIR 101
```

En el fichero "ejemplo3.c" añadimos la línea:

```
#include "ids.h"
```

Justo después de la línea "#include <windows.h>".

Ahora editaremos el fichero de recursos. Para ello pulsamos con el botón derecho del ratón sobre el ítem del proyecto y elegimos el ítem de menú "edit resource file".

En la primera línea, antes de la que comienza con "500 ICON...", introducimos la línea:

```
#include "ids.h"
```

Movemos el cursor a la línea 3 y escribimos:

```
Menu MENU
BEGIN
  POPUP "&Principal"
  BEGIN
    MENUITEM "&Prueba", ID_PRUEBA
    MENUITEM SEPARATOR
    MENUITEM "&Salir", ID_SALIR
  END
END
```

Ya podemos cerrar el cuadro de edición del fichero de recursos.

Para ver más detalles sobre el uso de este recurso puedes consultar las claves: [MENU](#), [POPUP](#) y [MENUITEM](#).

Cómo usar los recursos de menú:

Ahora tenemos varias opciones para usar el menú que acabamos de crear.

Primero veremos cómo cargarlo y asignarlo a nuestra ventana, ésta es la forma que más se parece a la del ejemplo del capítulo anterior. Para ello basta con insertar éste código antes de llamar a la función [ShowWindow](#):

```
HMENU hMenu;
...
hMenu = LoadMenu(hInstance, "Menu");
SetMenu (hWnd, hMenu);
```

O simplemente:

```
SetMenu (hWnd, LoadMenu(hInstance, "Menu"));
```

La función [LoadMenu](#) se encarga de cargar el recurso de menú, para ello hay que proporcionarle un manipulador de la instancia a la que pertenece el recurso y el nombre del menú..

Otro sistema, más sencillo todavía, es asignarlo como menú por defecto de la clase. Para esto basta con la siguiente asignación:

```
WndClass.lpszMenuName = "Menu";
```


Y por último, también podemos asignar un menú cuando creamos la ventana, especificándolo en la llamada a [CreateWindow](#):

```
hWnd = CreateWindow(  
    "NUESTRA_CLASE",  
    "Ventana de Ejemplo",  
    WS_OVERLAPPEDWINDOW,  
    CW_USEDEFAULT,  
    CW_USEDEFAULT,  
    320,  
    200,  
    HWND_DESKTOP,  
    LoadMenu(hInstance, "Menu"), // Carga y asignación de menú  
    hInstance,  
    NULL  
);
```

El tratamiento de los comandos procedentes del menú es igual que en el apartado anterior.

Ejemplo 3:

ejemplo3.c 24/12/2000 (1.927 bytes)

ids.h 24/12/2000 (47 bytes)

rsrc.rc 24/12/2000 (275 bytes)

win3.dev 24/12/2000 (406 bytes)



(Alternativo:



)

Capítulo 6 Diálogo básico

Los cuadros de diálogo son la forma más habitual de comunicación entre una aplicación Windows y el usuario. Para facilitar la tarea del usuario existen varios tipos de controles, cada uno de ellos diseñado para un tipo específico de información. Los más comunes son los "static", "edit", "button", "listbox", "scroll", "combobox", "group", "checkboxbutton" y "radiobutton". A partir de Windows 95 se introdujeron varios controles nuevos: "updown", "listview", "treeview", "gauge", "tab" y "trackbar".

En realidad, un cuadro de diálogo es una ventana normal, aunque con algunas peculiaridades, tiene su procedimiento de ventana, pero puede devolver un valor a la ventana que lo invoque.

Igual que los menús, los cuadros de diálogo se pueden construir durante la ejecución o a partir de un fichero de recursos.

Ficheros de recursos:

La mayoría de los compiladores de C/C++ que incluyen soporte para Windows poseen herramientas para la edición de recursos: menús, diálogos, bitmaps, etc. Sin embargo considero que es interesante que aprendamos a construir nuestros recursos con un editor de textos, cada compilador tiene sus propios editores de recursos, y no tendría sentido explicar cada uno de ellos. El compilador que usamos "Dev C++" tiene un editor muy limitado y no aconsejo su uso.

De modo que aprenderemos a hacer cuadros de diálogo igual que hemos aprendido a hacer menús, usando sólo texto.

Para el primer programa de ejemplo de programa con diálogos, que será el ejemplo 4, partiremos de nuevo del programa del ejemplo 1. Nuestro primer diálogo será muy sencillo: un simple cuadro con un texto y un botón de "Aceptar".

Este es el código del fichero de recursos:

```
#include <windows.h>
#include "IDS.H"
```

```
Menu MENU
BEGIN
  POPUP "&Principal"
  BEGUIN
    MENUITEM "&Diálogo", CM_DIALOGO
  END
END
```

```
DialogoPrueba DIALOG 0, 0, 118, 48
```

```

STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
BEGIN
CONTROL "Mensaje de prueba", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 8, 9, 84, 8
CONTROL "Aceptar", IDOK, "button", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 56, 26, 50, 14
END

```

Necesitamos incluir el fichero "windows.h" ya que en él se definen muchas constantes, como por ejemplo "IDOK" que es el identificador que se usa para el botón de "Aceptar".

También necesitaremos el fichero "ids.h", para definir los identificadores que usaremos en nuestro programa, por ejemplo el identificador del diálogo: "DialogoPrueba".

```
/* Identificadores */
```

```
/* Identificadores de comandos */
#define CM_DIALOGO 101
```

Lo primero que hemos definido es un menú para poder comunicarle a nuestra aplicación que queremos abrir un cuadro de diálogo.

A continuación está la definición del diálogo, que se compone de varias líneas. Puedes ver más detalles en el apartado de recursos dedicado a [DIALOG](#).

De momento bastará con un identificador, como el que usabamos para los menús; y además las coordenadas y dimensiones del diálogo.

En cuanto a los estilos, las constantes para definir los estilos de ventana, que comienzan con "WS_", puedes verlos con detalle en la sección de constantes "[estilos de ventana](#)". Y los estilos de diálogos, que comienzan con "DS_", en "[estilos de dialogo](#)".

Para empezar, hemos definido los siguientes estilos:

- DS_MODALFRAME: indica que se creará un cuadro de diálogo con un marco de dialog-box modal que puede combinarse con una barra de título y un menú de sistema.
- WS_POPUP: crea una ventana "pop-up".
- WS_VISIBLE: crea una ventana inicialmente visible.
- WS_CAPTION: crea una ventana con una barra de título, (incluye el estilo WS_BORDER).

La siguiente línea es la de CAPTION, en ello especificaremos el texto que aparecerá en la barra de título del diálogo.

La línea de FONT sirve para especificar el tamaño y el tipo de fuente de caracteres que usará nuestro diálogo.

Después está la zona de [controles](#), en nuestro ejemplo sólo hemos incluido un texto estático y un botón.

Un control estático (static) nos sirve para mostrar textos o rectángulos, que podemos usar para informar al usuario de algo, como etiquetas o como adorno. Para más detalles ver [controles static](#).

```
CONTROL "Mensaje de prueba", -1, "static", SS_LEFT | WS_CHILD |  
WS_VISIBLE, 8, 9, 84, 8
```

- CONTROL es una palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, introducimos el texto que se mostrará.
- id es el identificador del control. Como los controles static no se suelen manejar por las aplicaciones no necesitamos un identificador, así que ponemos -1.
- class es la clase de control, en nuestro caso "static".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo static](#) y varios [de ventana](#):
 - SS_LEFT: indica un simple rectángulo y el texto suministrado se alinea en su interior a la izquierda.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

El control button nos sirve para comunicarnos con el diálogo, podemos darle comandos del mismo tipo que los que proporciona un menú. Para más detalles ver [controles button](#).

```
CONTROL "Aceptar", IDOK, "button", BS_PUSHBUTTON | BS_CENTER |  
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 56, 26, 50, 14
```

- CONTROL es una palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, introducimos el texto que se mostrará en su interior.
- id es el identificador del control. Nuestra aplicación recibirá este identificador junto con el mensaje [WM_COMMAND](#) cuando el usuario active el botón. La etiqueta IDOK está definida en el fichero Windows.h.
- class es la clase de control, en nuestro caso "button".
- style es el estilo de control que queremos. En nuestro caso es una combinación de varios [estilos button](#) y varios [de ventana](#):

- BS_PUSHBUTTON: crea un botón corriente que envía un mensaje WM_COMMAND a su ventana padre cuando el usuario selecciona el botón.
 - BS_CENTER: centra el texto horizontalmente en el área del botón.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
 - WS_TABSTOP: define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
- coordenada x del control.
 - coordenada y del control.
 - width: anchura del control.
 - height: altura del control.

Procedimiento de diálogo:

Como ya hemos dicho, un diálogo es básicamente una ventana, y al igual que aquella, necesita un procedimiento asociado que procese los mensajes que le sean enviados, en este caso, un procedimiento de diálogo.

Sintaxis

```

BOOL CALLBACK DialogProc(
    HWND hwndDlg,    // manipulador del cuadro de diálogo
    UINT uMsg,       // mensaje
    WPARAM wParam,   // primer parámetro del mensaje
    LPARAM lParam    // segundo parámetro del mensaje
);
  
```

- hwndDlg identifica el cuadro de diálogo y es el manipulador de la ventana a la que está destinado el mensaje.
- msg es el código del mensaje.
- wParam es el parámetro de tipo palabra asociado al mensaje.
- lParam es el parámetro de tipo doble palabra asociado al mensaje.

La diferencia con el procedimiento de ventana que ya hemos visto está en el tipo de valor de retorno, que es el caso del procedimiento de diálogo es de tipo booleano.

Excepto en la respuesta al mensaje [WM_INITDIALOG](#), el procedimiento de diálogo debe retornar con un valor no nulo si procesa el mensaje y cero si no lo hace. Cuando responde a un mensaje WM_INITDIALOG, el procedimiento debe retornar cero si llama a la función [SetFocus](#) para poner el foco a uno de los controles del cuadro de diálogo. En otro caso, debe retornar un valor distinto de cero, y en ese caso el sistema pondrá el foco en el primer control del diálogo que pueda recibirlo.

Prototipo de procedimiento de diálogo

```
LRESULT CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);
```

Implementación de procedimiento de diálogo para nuestro ejemplo

Nuestro ejemplo es muy sencillo, ya que nuestro diálogo sólo puede proporcionar un comando, así que sólo debemos responder a un tipo de mensaje [WM_COMMAND](#) y al mensaje [WM_INITDIALOG](#).

Según hemos explicado un poco más arriba, del mensaje WM_INITDIALOG debemos retornar con un valor distinto de cero si no llamamos a SetFocus, como es nuestro caso.

Este mensaje lo usaremos para inicializar nuestro diálogo antes de que sea visible para el usuario, cuando haya algo que inicializar, claro.

Cuando procesemos el mensaje WM_COMMAND, que será siempre el que procede del único botón del diálogo, cerraremos el diálogo llamando a la función [EndDialog](#) y retornaremos con un valor distinto de cero.

En cualquier otro caso retornamos con FALSE, ya que no estaremos procesando el mensaje.

Nuestra función queda así:

```
BOOL CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM wParam,
LPARAM lParam)
{
    WORD status;

    switch (msg)          /* manipulador del mensaje */
    {
        case WM_INITDIALOG:
            return TRUE;
        case WM_COMMAND:
            EndDialog(hDlg, FALSE);
            return TRUE;
    }
    return FALSE;
}
```

Bueno, sólo nos falta saber como creamos un cuadro de diálogo. Para ello usaremos un comando de menú, por lo tanto, el diálogo se activará desde el procedimiento de ventana.

```

LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg,
WPARAM wParam, LPARAM lParam)
{
    static HINSTANCE hInstance;

    switch (msg)          /* manipulador del mensaje */
    {
        case WM_CREATE:
            hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
            return 0;
            break;
        case WM_COMMAND:
            if(LOWORD(wParam) == CM_DIALOGO)
                DialogBox(hInstance, "DialogoPrueba", hwnd, &DlgProc);
            break;
        case WM_DESTROY:
            PostQuitMessage(0); /* envía un mensaje WM_QUIT a la cola de
mensajes */
            break;
        default:          /* para los mensajes de los que no nos ocupamos */
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

```

En este procedimiento hay varias novedades.

Primero hemos declarado una variable estática "hInstance" para tener siempre a mano un manipulador de la instancia actual.

Para inicializar este valor hacemos uso del mensaje [WM_CREATE](#), que se envía a una ventana cuando es creada, antes de que se visualice por primera vez. Aprovechamos es hecho de que nuestro procedimiento de ventana sólo recibe una vez este mensaje y de que lo hace antes de poder recibir ningún comando. En el futuro veremos que se usa para toda clase de inicializaciones.

El mensaje WM_CREATE tiene como parámetro en lParam un puntero a una estructura [CREATESTRUCT](#) que contiene información sobre la ventana, en nuestro caso sólo nos interesa el campo hInstance.

La otra novedad es la llamada a la función [DialogBox](#), que es la que crea el cuadro de diálogo.

Esta función necesita varios parámetros:

1. Un manipulador a la instancia de la aplicación, que hemos obtenido al procesar el mensaje WM_CREATE.
2. Un identificador de recurso de diálogo, este es el nombre que utilizamos para el diálogo al crear el recurso, entre comillas.

3. Un manipulador a la ventana a la que pertenece el diálogo.
4. Y la dirección del procedimiento de ventana que hará el tratamiento del diálogo.

Y ya tenemos nuestro primer ejemplo del uso de diálogos, en capítulos siguientes empezaremos a conocer más detenidamente cómo usar cada uno de los controles básicos: Edit, List Box, Scroll Bar, Static, Button, Combo Box, Group Box, Check Button y Radio Button. Le dedicaremos un capítulo a cada uno de ellos.

Ejemplo 4: 16/01/2001

ejemplo4.c 16/01/2001 (2.569 bytes)

ids.h 7/01/2001 (86 bytes)

rsrc.rc 16/01/2001 (481 bytes)

win4.dev 16/01/2001 (366 bytes)



(Alternativo:



)

Capítulo 7 Control básico Edit

Tal como hemos definido nuestro diálogo en el capítulo 6, no tiene mucha utilidad. Los diálogos se usan para intercambiar información entre la aplicación y el usuario, en ambas direcciones. El ejemplo 4 sólo lo hace en una de ellas.

En el capítulo anterior hemos usado dos controles (un texto estático y un botón), aunque sin saber exactamente cómo funcionan. En éste capítulo veremos el uso del control de edición.

Un control edit es una ventana de control rectangular que permite al usuario introducir y editar texto desde el teclado.

Cuando está seleccionado muestra el texto que contiene y un cursor intermitente que indica el punto de inserción de texto. Para seleccionarlo el usuario puede hacer un click con el ratón en su interior o usar la tecla TAB. El usuario podrá entonces introducir texto, cambiar el punto de inserción, o seleccionar texto para ser borrado o movido usando el teclado o el ratón. Un control de este tipo puede enviar mensajes a su ventana padre mediante WM_COMMAND, y la ventana padre puede enviar mensajes a un control edit en un cuadro de diálogo llamando a la función [SendDlgItemMessage](#). Veremos algunos de estos mensajes en este capítulo, y el resto en los capítulos más avanzados.

Fichero de recursos

Empezaremos definiendo el control edit en el fichero de recursos, y lo añadiremos a nuestro diálogo de prueba.

```
DialogoPrueba DIALOG 0, 0, 118, 48
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
{
CONTROL "Texto:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 8,
9, 28, 8
CONTROL "", ID_TEXTO, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 36, 9, 76, 12
CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 56, 26, 50, 14
}
```

Hemos hecho algunas modificaciones más, para empezar, el control static se ha convertido en una etiqueta para el control edit, que indica al usuario qué tipo de información debe suministrar.

Hemos añadido el control edit a continuación del control static. Veremos que el orden en que aparecen los controles dentro del cuadro de diálogo es muy

importante, al menos en aquellos controles que tengan el estilo WS_TABSTOP, ya que ese orden será el mismo en que se activen los controles cuando usemos la tecla TAB. Para más detalles acerca de los controles edit ver [controles edit](#).

Veamos cómo hemos definido nuestro control edit:

```
CONTROL "", ID_TEXTO, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE |  
WS_BORDER | WS_TABSTOP, 36, 9, 76, 12
```

- CONTROL es una palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, introducimos el texto que se mostrará en el interior del control, en este caso, ninguno.
- id es el identificador del control. Los controles edit necesitan un identificador para que la aplicación pueda acceder a ellos. Usaremos un identificador definido en IDS.h.
- class es la clase de control, en nuestro caso "EDIT".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo edit](#) y varios [de ventana](#):
 - ES_LEFT: indica que el texto en el interior del control se alinearán a la izquierda.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
 - WS_BORDER: se crea un control que tiene de borde una línea fina.
 - WS_TABSTOP: define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

El procedimiento de diálogo y los controles edit

Para manejar el control edit desde nuestro procedimiento de diálogo tendremos que hacer algunas modificaciones.

Para empezar, los controles edit también pueden generar mensajes [WM_COMMAND](#), de modo que debemos diferenciar el control que originó dicho mensaje y tratarlo de diferente modo según el caso.

```
case WM_COMMAND:  
    if(LOWORD(wParam) == IDOK) EndDialog(hDlg, FALSE);  
    return TRUE;
```

En nuestro caso sigue siendo sencillo: sólo cerraremos el diálogo si el mensaje WM_COMMAND proviene del botón "Aceptar".

La otra modificación afecta al mensaje [WM_INITDIALOG](#).

```
case WM_INITDIALOG:  
    SetFocus(GetDlgItem(hDlg, ID_TEXTO));  
    return FALSE;
```

De nuevo es una modificación sencilla, tan sólo haremos que el foco del teclado se coloque en el control edit, de modo que el usuario pueda empezar a escribir directamente.

Para hacer eso usaremos la función [SetFocus](#). Pero esta función requiere como parámetro el manipulador de ventana del control que debe recibir el foco, este manipulador lo conseguimos con la función [GetDlgItem](#), que a su vez necesita como parámetros un manipulador del diálogo y el identificador del control.

Quizás has notado que a nuestro programa le falta algo.

Efectivamente, podemos introducir y modificar texto en el cuadro de diálogo, pero no podemos asignar valores iniciales al control de edición ni tampoco podemos hacer que la aplicación tenga acceso al texto introducido por el usuario.

Variables a editar en los cuadros de diálogo

Lo primero que tenemos que tener es algún tipo de variable que puedan compartir los procedimientos de ventana de la aplicación y el del diálogo. En nuestro caso se trata sólo de una cadena, pero según se añadan más parámetros al cuadro de edición, estos datos pueden ser más complejos, así que usaremos un sistema que nos valdrá en todos los casos.

Se trata de crear una estructura con todos los datos que queremos que el procedimiento de diálogo comparta con el procedimiento de ventana:

```
typedef struct stDatos {  
    char Texto[80];  
} DATOS;
```

```
DATOS Datos;
```

Lo más sencillo es que estos datos sean globales, pero no será buena idea cuando nuestros programas estén escritos en C++, ya que en POO no está "bien visto" el uso de variables globales.

Tampoco parece muy buena idea declarar los datos en el procedimiento de ventana, ya que este procedimiento se usa para todas las ventanas de la misma clase, y tendríamos que definir los datos como estáticos.

Para este ejemplo usaremos variables globales, en capítulos más avanzados veremos cómo se pueden pasar parámetros a los procedimientos de diálogo.

Daremos valores iniciales a las variables de la aplicación, dentro de WinMain:

```
// Inicialización de los datos de la aplicación
strcpy(Datos.Texto, "Inicial");
```

Iniciar controles edit

Ahora tenemos que hacer que se actualice el contenido del control edit al abrir el cuadro de diálogo.

El lugar adecuado para hacer esto es en el proceso del mensaje WM_INITDIALOG:

```
case WM_INITDIALOG:
    SendDlgItemMessage(hDlg, ID_TEXTO, EM_LIMITTEXT, 80, 0L);
    SetDlgItemText(hDlg, ID_TEXTO, Datos.Texto);
    SetFocus(GetDlgItem(hDlg, ID_TEXTO));
    return FALSE;
```

Hemos añadido dos llamadas a dos nuevas funciones del API. La primera es a [SendDlgItemMessage](#), que envía un mensaje a un control. En este caso se trata de un mensaje [EM_LIMITTEXT](#), que sirve para limitar la longitud del texto que se puede almacenar y editar en el control. Es necesario que hagamos esto, ya que el texto que puede almacenar nuestra estructura de datos está limitado a 80 caracteres.

También hemos añadido una llamada a la función [SetDlgItemText](#), que hace exactamente lo que pretendemos: cambiar el contenido del texto en el interior de un control edit.

Devolver valores a la aplicación

También queremos que cuando el usuario está satisfecho con los datos que ha introducido, y pulse el botón de aceptar, el dato de nuestra aplicación se actualice con el texto que hay en el control edit.

Esto lo podemos hacer de varios modos. Como veremos en capítulos más avanzados, podemos responder a mensajes que provengan del control cada vez que cambia su contenido.

Pero ahora nos limitaremos a leer ese contenido cuando procesemos el comando generado al pulsar el botón de "Aceptar".

```
case WM_COMMAND:
    if(LOWORD(wParam) == IDOK)
    {
```

```

    GetDlgItemText(hDlg, ID_TEXTO, Datos.Texto, 80);
    EndDialog(hDlg, FALSE);
}
return TRUE;

```

Para eso hemos añadido la llamada a la función [GetDlgItemText](#), que es simétrica a SetDlgItemText.

Ahora puedes comprobar lo que pasa cuando abres varias veces seguidas el cuadro de diálogo modificando el texto cada vez.

Con esto parece que ya controlamos lo básico de los controles edit, pero aún hay algo más.

Añadir la opción de cancelar

Es costumbre dar al usuario la oportunidad de arrepentirse si ha modificado algo en un cuadro de diálogo y, por la razón que sea, cambia de idea.

Para eso se suele añadir un segundo botón de "Cancelar".

Empecemos por añadir dicho botón en el fichero de recursos:

```

DialogoPrueba DIALOG 0, 0, 118, 48
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
{
    CONTROL "Texto:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 9,
    28, 8
    CONTROL "", ID_TEXTO, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE |
    WS_BORDER | WS_TABSTOP, 36, 9, 76, 12
    CONTROL "Aceptar", IDOK, "BUTTON", BS_DEFPUSHBUTTON |
    BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 26, 45, 14
    CONTROL "Cancelar", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
    BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 61, 26, 45, 14
}

```

Hemos cambiado las coordenadas de los botones, para que el de "Aceptar" aparezca a la izquierda. Además, el botón de "Aceptar" lo hemos convertido en el botón por defecto, añadiendo el estilo BS_DEFPUSHBUTTON. Haciendo eso, podemos simular la pulsación del botón de aceptar pulsando la tecla de "intro".

El identificador del botón de "Cancelar" es IDCANCEL, y está definido en Windows.h.

Ahora tenemos que hacer que nuestro procedimiento de diálogo manipule el mensaje del botón de "Cancelar".

```

case WM_COMMAND:
    switch(LOWORD(wParam)) {
        case IDOK:
            GetDlgItemText(hDlg, ID_TEXTO, Datos.Texto, 80);
            EndDialog(hDlg, FALSE);
            break;
        case IDCANCEL:
            EndDialog(hDlg, FALSE);
            break;
    }
    return TRUE;

```

Como puedes ver, sólo leemos el contenido del control edit si se ha pulsado el botón de "Aceptar".

Ejemplo 5: 4/02/2001

ejemplo5.c 4/02/2001 (3.244 bytes)

ids.h 4/02/2001 (108 bytes)

rsrc.rc 4/02/2001 (772 bytes)

win5.dev 4/02/2001 (440 bytes)  (Alternativo: )

En muchas ocasiones necesitaremos editar valores de números enteros en nuestros diálogos.

Para eso, el API tiene previstas algunas constantes y funciones, (no así para números en coma flotante, para los que tendremos que crear nuestros propios controles).

Bien, vamos a modificar nuestro ejemplo para editar valores numéricos en lugar de cadenas de texto.

Fichero de recursos para editar enteros

Empezaremos añadiendo una constante al fichero de identificadores: "IDS.h":

```
#define ED_NUMERO 100
```

Y redefiniendo el control edit en el fichero de recursos, al que añadiremos el flag [ES_NUMBER](#) para que sólo admita caracteres numéricos:

```

DialogoPrueba DIALOG 0, 0, 118, 48
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
{

```

```

CONTROL "Número:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE,
8, 9, 28, 8
CONTROL "", ID_NUMERO, "EDIT", ES_NUMBER | ES_LEFT | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 36, 9, 76, 12
CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 56, 26, 50, 14
}

```

Variables a editar en los cuadros de diálogo

Ahora modificaremos la estructura de los datos para que el dato a editar sea de tipo numérico:

```

typedef struct stDatos {
    int Numero;
} DATOS;

```

```
DATOS Datos;
```

Daremos valores iniciales a las variables de la aplicación, dentro de WinMain:

```

// Inicialización de los datos de la aplicación
Datos.Numero = 123;

```

Iniciar controles edit de enteros

Ahora tenemos que hacer que se actualice el contenido del control edit al abrir el cuadro de diálogo.

El lugar adecuado para hacer esto es en el proceso del mensaje WM_INITDIALOG:

```

case WM_INITDIALOG:
    SetDlgItemInt(hDlg, ID_NUMERO, (UINT)Datos.Numero, FALSE);
    SetFocus(GetDlgItem(hDlg, ID_NUMERO));
    return FALSE;

```

En este caso no es necesario limitar el texto que podemos editar en el control, ya que, como veremos, las propias funciones del API se encargan de capturar y convertir el contenido del control en un número, de modo que no tenemos que preocuparnos de que no quepa en nuestra variable.

También hemos modificado la función a la que llamamos para modificar el contenido del control, ahora usaremos [SetDlgItemInt](#), que cambia el contenido de un control edit con un valor numérico.

Devolver valores a la aplicación

Por último leeremos el contenido cuando procesemos el comando generado al pulsar el botón de "Aceptar".

```
    BOOL NumeroOk;
...
    case WM_COMMAND:
        switch(LOWORD(wParam)) {
            case IDOK:
                Datos.Numero = GetDlgItemInt(hDlg, ID_NUMERO, &NumeroOk,
FALSE);
                if(NumeroOk) EndDialog(hDlg, FALSE);
                else MessageBox(hDlg, "Número no válido", "Error",
MB_ICONEXCLAMATION | MB_OK);
                break;
```

Para eso hemos añadido la llamada a la función [GetDlgItemInt](#), que es simétrica a [SetDlgItemInt](#). El proceso difiere del usado para capturar cadenas, ya que en este caso la función nos devuelve el valor numérico del contenido del control edit.

También devuelve un parámetro que indica si ha habido algún error durante la conversión. Si el valor de ese parámetro es TRUE, significa que la conversión se realizó sin problemas, si es FALSE, es que ha habido un error. Si nuestro programa detecta un error visualizará un mensaje de error y no permitirá abandonar el cuadro de diálogo.

También hemos usado el flag [BM_ICONEXCLAMATION](#), que añade un icono al cuadro de mensaje y el sonido predeterminado para alertar al usuario.

Ejemplo 6: 10/02/2001

ejemplo6.c 10/02/2001 (3.314 bytes)

ids.h 10/02/2001 (111 bytes)

rsrc.rc 10/02/2001 (786 bytes)

win6.dev 10/02/2001 (440 bytes)



(Alternativo:



)

Capítulo 8 Control básico

ListBox

Los controles edit son muy útiles cuando la información a introducir por el usuario es imprevisible o existen muchas opciones. Pero cuando el número de opciones no es muy grande y son todas conocidas es preferible usar un control ListBox.

Ese es el siguiente control básico que veremos. Un ListBox consiste en una ventana rectangular con una lista de cadenas entre las cuales el usuario puede escoger.

El usuario puede seleccionar una cadena apuntandola y haciendo clic con el botón del ratón. Cuando una cadena se selecciona, se resalta y se envía un mensaje de notificación a la ventana padre. También se puede usar una barra de scroll con los listbox para desplazar listas muy largas o demasiado anchas para la ventana.

Ficheros de recursos

Empezaremos definiendo el control listbox en el fichero de recursos, y lo añadiremos a nuestro dialogo de prueba:

```
DialogoPrueba DIALOG 0, 0, 130, 140
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
{
CONTROL "Lista:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 9,
28, 8
CONTROL "", ID_LISTA, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 9, 19, 104, 99
CONTROL "Aceptar", IDOK, "BUTTON", BS_DEFPUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 116, 45, 14
CONTROL "Cancelar", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 61, 116, 45, 14
}
```

Hemos añadido el control listbox a continuación del control static. Para más detalles acerca de los controles listbox ver [controles listbox](#).

Ahora veamos cómo hemos definido nuestro control listbox:

```
CONTROL "", ID_LISTA, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 9, 19, 104, 99
```

- CONTROL es la palabra clave que indica que vamos a definir un control.

- A continuación, en el parámetro text, en el caso de los listbox no tiene ninguna función. Lo dejaremos como cadena vacía.
- id es el identificador del control. Los controles listbox necesitan un identificador para que la aplicación pueda acceder a ellos. Usaremos un identificador definido en IDS.h.
- class es la clase de control, en nuestro caso "LISTBOX".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo listbox](#) y varios [de ventana](#):
 - LBS_STANDARD: ordena alfabéticamente las cadenas en el listbox. La ventana padre recibe in mensaje de entrada cada vez que el usuario hacer click o doble click sobre una cadena. El list box tiene bordes en todos sus lados.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
 - WS_TABSTOP: define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

Iniciar controles listbox

Para este ejemplo también usaremos variables globales para almacenar el valor de la cadena del listbox actualmente seleccionada.

```
// Datos de la aplicación
typedef struct stDatos {
    char Item[80];
} DATOS;
```

```
DATOS Datos;
```

Daremos valores iniciales a las variables de la aplicación, dentro de WinMain:

```
// Inicialización de los datos de la aplicación
strcpy(Datos.Item, "Cadena nº 3");
```

La característica más importante de los listbox es que contienen listas de cadenas. Así que es imprescindible iniciar este tipo de controles, introduciendo las cadenas antes de que se muestre el diálogo. Eso se hace durante el proceso del mensaje [WM_INITDIALOG](#) dentro del procedimiento de diálogo.

```
case WM_INITDIALOG:
    // Añadir cadenas. Mensaje: LB_ADDSTRING
```

```

        SendDlgItemMessage(hDlg, ID_LISTA, LB_ADDSTRING, 0,
(LPARAM)"Cadena nº 1");
        SendDlgItemMessage(hDlg, ID_LISTA, LB_ADDSTRING, 0,
(LPARAM)"Cadena nº 4");
        SendDlgItemMessage(hDlg, ID_LISTA, LB_ADDSTRING, 0,
(LPARAM)"Cadena nº 3");
        SendDlgItemMessage(hDlg, ID_LISTA, LB_ADDSTRING, 0,
(LPARAM)"Cadena nº 2");
        SendDlgItemMessage(hDlg, ID_LISTA, LB_SELECTSTRING, -1,
(LPARAM)Datos.Item);
        SetFocus(GetDlgItem(hDlg, ID_LISTA));
return FALSE;

```

Para añadir cadenas a un listbox se usa el mensaje [LB_ADDSTRING](#) mediante la función [SendDlgItemMessage](#), que envía un mensaje a un control.

También podemos preseleccionar alguna de las cadenas del listbox, aunque esto no es muy frecuente ya que se suele dejar al usuario que seleccione una opción sin sugerirle nada. Para seleccionar una de las cadenas también se usa un mensaje: [LB_SELECTSTRING](#). Usaremos el valor -1 en wParam para indicar que se busque en todo el listbox.

Devolver valores a la aplicación

También queremos que cuando el usuario está satisfecho con los datos que ha introducido, y pulse el botón de aceptar, el dato de nuestra aplicación se actualice con el texto del ítem seleccionado.

De nuevo recurriremos a mensajes para pedirle al listbox el valor de la cadena actualmente seleccionada. En este caso se trata dos mensajes combinados, uno es [LB_GETCURSEL](#), que se usa para averiguar el índice de la cadena actualmente seleccionada. El otro es [LB_GETTEXT](#), que devuelve la cadena del índice que le indiquemos.

Cuando trabajemos con memoria dinámica y con ítems de longitud variable, será interesante saber la longitud de la cadena antes de leerla desde el listbox. Para eso podemos usar el mensaje [LB_GETTEXTLEN](#).

Haremos esa lectura al procesar el comando IDOK, que se genera al pulsar el botón "Aceptar".

```

    UINT indice;
...
    case WM_COMMAND:
        switch(LOWORD(wParam)) {
            case IDOK:
                indice = SendDlgItemMessage(hDlg, ID_LISTA, LB_GETCURSEL, 0,
0);

```

```
        SendDlgItemMessage(hDlg, ID_LISTA, LB_GETTEXT, indice,  
(LPARAM)Datos.Item);  
        EndDialog(hDlg, FALSE);  
        break;  
    case IDCANCEL:  
        EndDialog(hDlg, FALSE);  
        break;  
    }  
    return TRUE;
```

Ejemplo 7: 10/02/2001

ejemplo7.c 10/02/2001 (3.744 bytes)

ids.h 10/02/2001 (110 bytes)

rsrc.rc 10/02/2001 (772 bytes)

win7.dev 10/02/2001 (440 bytes)  (Alternativo: )

Capítulo 9 Control básico Button

Los controles button simulan el comportamiento de un pulsador o un interruptor. Pero sólo cuando se comportan como un pulsador los llamaremos botones, cuando emulen interruptores nos referiremos a ellos como checkbox o radiobutton.

Los botones se usan para que el usuario pueda ejecutar ciertas acciones o para dar órdenes a una aplicación. En muchos aspectos, funcionan igual que los menús, y de hecho, ambos generan mensajes de tipo [WM_COMMAND](#).

Los botones se componen normalmente de una pequeña área rectangular con un texto en su interior que identifica la acción asociada.

En realidad ya hemos usado controles button en todos los ejemplos anteriores, pero los explicaremos ahora con algo más de detalle.

Ficheros de recursos

Empezaremos definiendo el control button en el fichero de recursos, y lo añadiremos a nuestro dialogo de prueba:

```
DialogoPrueba DIALOG 0, 0, 130, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Diálogo de prueba"
FONT 8, "Helv"
{
CONTROL "Botones:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 9,
28, 8
CONTROL "Nuestro botón", ID_BOTON, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 9, 19, 104, 25
CONTROL "Aceptar", IDOK, "BUTTON", BS_DEFPUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 50, 45, 14
CONTROL "Cancelar", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 61, 50, 45, 14
}
```

Hemos añadido un nuevo control button a continuación del control static. Para más detalles acerca de los controles button ver [controles button](#).

Ahora veamos cómo hemos definido nuestro control button, y también los otros dos que hemos usado hasta ahora.:

```
CONTROL "Nuestro botón", ID_BOTON, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 9, 19, 104, 25
```

CONTROL "Aceptar", IDOK, "BUTTON", BS_DEFPUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 50, 45, 14

- CONTROL es la palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, en el caso de los button se trata del texto que aparecerá en su interior.
- id es el identificador del control. Los controles button necesitan un identificador para que la aplicación pueda acceder a ellos y para usarlos como parámetro en los mensajes WM_COMMAND. Usaremos un identificador definido en IDS.h.
- class es la clase de control, en nuestro caso "BUTTON".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo button](#) y varios [de ventana](#):
 - BS_PUSHBUTTON: Crea un botón corriente que envía un mensaje WM_COMMAND a su ventana padre cuando el usuario pulsa el botón.
 - BS_DEFPUSHBUTTON: Crea un botón normal que se comporta como uno del estilo BS_PUSHBUTTON, pero también tiene un borde negro y grueso. Si el botón está en un cuadro de diálogo, el usuario puede pulsar este botón usando la tecla ENTER, aún cuando el botón no tenga el foco de entrada. Este estilo es corriente para permitir al usuario seleccionar rápidamente la opción más frecuente, la opción por defecto. Lo usaremos frecuentemente con el botón "Aceptar".
 - BS_CENTER: Centra el texto horizontalmente en el área del botón.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
 - WS_TABSTOP: define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

Iniciar controles button

Los controles button no se usan para editar o seleccionar información, sólo para que el usuario pueda dar órdenes o indicaciones a la aplicación, así que no requieren inicialización. Lo más que haremos en algunos casos es situar el foco en uno de ellos.

Eso se hace durante el proceso del mensaje [WM_INITDIALOG](#) dentro del procedimiento de diálogo.

```
case WM_INITDIALOG:  
    SetFocus(GetDlgItem(hDlg, ID_BOTON));
```

```
return FALSE;
```

Tratamiento de acciones de los controles button

Nuestro cuadro de diálogo tiene tres botones. Los de "Aceptar" y "Cancelar" tienen una misión clara, validar o ignorar los datos y cerrar el cuadro de diálogo. En el caso de nuestro botón, queremos que se realice otra operación diferente, por ejemplo, mostrar un mensaje.

```
case WM_COMMAND:
    switch(LOWORD(wParam)) {
        case ID_BOTON:
            MessageBox(hDlg, "Se pulsó 'Nuestro botón'", "Acción",
MB_ICONINFORMATION|MB_OK);
            break;
        case IDOK:
            EndDialog(hDlg, FALSE);
            break;
        case IDCANCEL:
            EndDialog(hDlg, FALSE);
            break;
    }
    return TRUE;
```

Ejemplo 8: 19/02/2001

ejemplo8.c 19/02/2001 (3.034 bytes)

ids.h 19/02/2001 (110 bytes)

rsrc.rc 19/02/2001 (796 bytes)

win8.dev 19/02/2001 (362 bytes)



(Alternativo:



)

Capítulo 10 Control básico Static

Los static son el tipo de control menos interactivo, normalmente se usan como información o decoración. Pero son muy importantes. Windows es un entorno gráfico, y la apariencia forma una parte muy importante de él

Existen varios tipos de controles static, o mejor dicho, varios estilos de controles static.

Dependiendo del estilo que elijamos para cada control static, su aspecto será radicalmente distinto, desde una simple línea o cuadro hasta un bitmap, un icono o un texto.

Cuando hablemos de controles static de tipo texto, nos referiremos a ellos normalmente también como etiquetas. Las etiquetas pueden tener también una función añadida, como veremos más adelante: nos servirán para acceder a otros controles usando el teclado.

En realidad ya hemos usado controles static del tipo etiqueta cuando vimos los controles edit, listbox y button, pero de nuevo los explicaremos ahora con más detalle.

Ficheros de recursos

Empezaremos definiendo varios controles static en el fichero de recursos, y los añadiremos a nuestro dialogo de prueba, para obtener un muestrario:

```
DialogoPrueba DIALOG 0, 0, 240, 120
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP |
WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Prueba de static"
FONT 8, "Helv"
{
CONTROL "Frame1", -1, "STATIC", SS_WHITEFRAME | WS_CHILD |
WS_VISIBLE, 8, 5, 52, 34
CONTROL "Frame2", -1, "STATIC", SS_GRAYFRAME | WS_CHILD |
WS_VISIBLE, 12, 9, 52, 34
CONTROL "Frame3", -1, "STATIC", SS_BLACKFRAME | WS_CHILD |
WS_VISIBLE, 16, 13, 52, 34
CONTROL "Rect1", -1, "STATIC", SS_BLACKRECT | WS_CHILD |
WS_VISIBLE, 72, 22, 48, 34
CONTROL "Rect2", -1, "STATIC", SS_GRAYRECT | WS_CHILD |
WS_VISIBLE, 12, 60, 52, 34
CONTROL "Rect3", -1, "STATIC", SS_WHITERECT | WS_CHILD |
WS_VISIBLE, 72, 60, 48, 34
```



```

CONTROL "Bitmap1", -1, "STATIC", SS_BITMAP | WS_CHILD |
WS_VISIBLE, 128, 22, 18, 15
CONTROL "Icono", -1, "STATIC", SS_ICON | WS_CHILD | WS_VISIBLE,
188, 47, 20, 20
CONTROL "Edit &1:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE,
128, 73, 40, 9
CONTROL "", ID_EDIT1, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 180, 73, 20, 12
CONTROL "Edit &2:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE,
128, 95, 28, 8
CONTROL "", ID_EDIT2, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE |
WS_BORDER | WS_TABSTOP, 180, 95, 20, 12
CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 186, 6, 50, 14
}

```

Hemos añadido diez nuevos controles static. Para más detalles acerca de los controles static ver [controles static](#).

Ahora veamos cómo hemos definido nuestros controles static:

```

CONTROL "Frame1", -1, "static", SS_WHITEFRAME | WS_CHILD |
WS_VISIBLE, 8, 5, 52, 34
CONTROL "Bitmap1", -1, "static", SS_BITMAP | WS_CHILD | WS_VISIBLE,
128, 22, 18, 15
CONTROL "Icono", -1, "static", SS_ICON | WS_CHILD | WS_VISIBLE, 188, 47,
20, 20
CONTROL "Edit &1:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 128,
73, 40, 9

```

- CONTROL es la palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, en el caso de los static tiene sentido para las etiquetas, los bitmaps y los iconos. En estos dos últimos casos indicará el nombre del recurso a insertar. En el resto de los casos se incluye como información. Comentaremos algo más sobre los textos de la etiquetas más abajo.
- id es el identificador del control. Los controles static no suelen necesitar un identificador, ya que no suelen tener un comportamiento interactivo. De modo que todos los identificadores de controles static serán -1.
- class es la clase de control, en nuestro caso "STATIC".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo static](#) y varios [de ventana](#):
 - SS_WHITEFRAME, SS_GRAYFRAME, SS_BLACKFRAME: Crea un rectángulo vacío o un marco de color blanco, gris o negro, respectivamente. SS_WHITEREC, SS_GRAYREC, SS_BLACKREC: Crea un rectángulo relleno de color blanco, gris o negro, respectivamente. SS_BITMAP mostrará el bitmap indicado en el campo text. SS_ICON mostrará el icono indicado en el campo text. SS_LEFT, SS_RIGHT, SS_CENTER: indican que es una

etiqueta y ajustará el texto a la izquierda, la derecha o el centro, respectivamente.

◦WS_CHILD: crea el control como una ventana hija.

◦WS_VISIBLE: crea una ventana inicialmente visible.

- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

En el caso de las etiquetas, cuando se incluye el carácter '&', el siguiente carácter de la cadena aparecerá subrayado, indicando que puede ser usado como acelerador, (pulsando la tecla 'ALT' más el carácter subrayado), para acceder al control más cercano, normalmente a su derecha o debajo. Pero, cuidado, en realidad el acelerador situará el foco en el control definido exactamente a continuación del control static en el fichero de recursos, y no al más cercano físicamente en pantalla.

En el ejemplo, verifica lo que sucede al pulsar la tecla ALT más '1' ó '2'. Verás que el foco del teclado se desplaza a los cuadros de edición 1 y 2.

Iniciar controles static

Los controles static normalmente no necesitan iniciarse, para eso son estáticos.

Tratamiento de acciones de los controles static

Los controles static tampoco responderán, normalmente, a acciones del usuario, ni tampoco generarán mensajes. En el caso de las etiquetas, el comportamiento de los aceleradores es automático y no requerirá ninguna acción del programa.

Ejemplo 9: 18/03/2001

ejemplo9.c 18/03/2001 (2.749 bytes)

ids.h 18/03/2001 (155 bytes)

rsrc.rc 18/03/2001 (1.677 bytes)

Smiley.bmp 09/02/1999 (518 bytes)

win9.dev 18/03/2001 (362 bytes)



(Alternativo:



)

Capítulo 11 Control básico ComboBox

Los ComboBoxes son una combinación de un control Edit y un Listbox. Son los controles que suelen recordar las entradas que hemos introducido antes, para que podamos seleccionarlas sin tener que escribirlas, en ese sentido funcionan igual que un Listbox, pero también permiten introducir nuevas entradas.

Hay modalidades de ComboBox en las que el control Edit está inhibido, y no permite introducir nuevos valores. En esos casos, el control se comportará de un modo muy parecido al que lo hace un Listbox, pero, como veremos más adelante, tienen ciertas ventajas.

Existen tres tipos de ComboBoxes:

- Simple: es la forma que muestra siempre el control Edit y el ListBox, aunque ésta esté vacía.
- DropDown: despliegue hacia abajo. Se muestra un pequeño icono a la derecha del control Edit, si el usuario lo pulsa con el ratón, se desplegará el ListBox, mientras no se pulse, la lista permanecerá oculta.
- DropDownList: lo mismo, pero el control Edit se sustituye por un control Static.

Ficheros de recursos

Para nuestro ejemplo incluiremos un control ComboBox de cada tipo, así veremos las peculiaridades de cada uno:

```
DialogoPrueba DIALOG 0, 0, 205, 78
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "Combo boxes"
FONT 8, "Helv"
{
CONTROL "&Simple", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 2,
60, 8
CONTROL "ComboBox1", ID_COMBOBOX1, "COMBOBOX", CBS_SORT |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 13, 60, 43
CONTROL "&Dropdown", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
73, 2, 60, 8
CONTROL "ComboBox2", ID_COMBOBOX2, "COMBOBOX",
CBS_DROPDOWN | CBS_SORT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 72, 13, 60, 103
CONTROL "Dropdown &List", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 138, 2, 60, 8
```

```

CONTROL "ComboBox3", ID_COMBOBOX3, "COMBOBOX",
CBS_DROPDOWNLIST | CBS_SORT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 136, 13, 60, 103
CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 28, 60, 50, 14
CONTROL "Cancelar", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 116, 60, 50, 14
}

```

Hemos añadido los nuevos controles ComboBox. Para más detalles acerca de estos controles ver [controles combobox](#).

Ahora veremos más detalles sobre los estilos de los controles ComboBox:

```

CONTROL "ComboBox1", ID_COMBOBOX1, "COMBOBOX", CBS_SORT |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 13, 60, 43
CONTROL "ComboBox2", ID_COMBOBOX2, "COMBOBOX",
CBS_DROPDOWN | CBS_SORT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 72, 13, 60, 103
CONTROL "ComboBox3", ID_COMBOBOX3, "COMBOBOX",
CBS_DROPDOWNLIST | CBS_SORT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 136, 13, 60, 103

```

- CONTROL es la palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, en el caso de los combobox sólo sirve como información, y no se usa.
- id es el identificador del control. El identificador será necesario para inicializar y leer los contenidos y selecciones del combobox.
- class es la clase de control, en nuestro caso "COMBOBOX".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo combobox](#) y varios [de ventana](#):
 - CBS_SORT: Indica que los valores en la lista se aparecerán por orden alfabético. CBS_DROPDOWN crea un ComboBox del tipo DropDown. CBS_DROPDOWNLIST crea un ComboBox del tipo DropDownList.
 - WS_CHILD: crea el control como una ventana hija.
 - WS_VISIBLE: crea una ventana inicialmente visible.
 - WS_TABSTOP: para que cuando el foco cambie de control al pulsar TAB, pase por este control.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

Iniciar controles ComboBox

Iniciar los controles ComboBox es análogo a iniciar los controles ListBox. En general, necesitaremos introducir una lista de valores en la lista, para que el usuario pueda usarla.

El lugar adecuado para hacerlo también es al procesar el mensaje [WM_INITDIALOG](#) de nuestro cuadro de diálogo, y el mensaje para añadir cadenas es [CB_ADDSTRING](#). Recordar también que para enviar mensajes a un control se usa la función [SendDlgItemMessage](#).

Para hacer más fácil la inicialización de las listas, usaremos los mismos valores en las tres. Para ello definiremos un array de cadenas con los valores que usaremos:

```
char *Lista[] = {  
    "aaaaaaaaaaaa",  
    "bbbbbbbbbbbbbb",  
    "cccccccccccc",  
    "rrrrrrrrrrrr",  
    "ffffffffffff",  
    "dddddddddddd"  
};
```

También usaremos una estructura para almacenar los valores iniciales y de la última selección de los tres comboboxes:

```
/* Datos de la aplicación */  
typedef struct stDatos {  
    char Item[3][80];  
} DATOS;
```

DATOS Datos;

Y asignaremos valores iniciales a las selecciones dentro de "WinMain":

```
strcpy(Datos.Item[0], "a");  
strcpy(Datos.Item[1], "c");  
strcpy(Datos.Item[2], "r");
```

La parte de inicialización de los comboboxes quedaría así:

```
case WM_INITDIALOG:  
    // Añadir cadenas. Mensaje: LB_ADDSTRING  
    for(i = 0; i < 6; i++) {  
        SendDlgItemMessage(hDlg, ID_COMBOBOX1, CB_ADDSTRING, 0,  
(LPARAM)Lista[i]);  
        SendDlgItemMessage(hDlg, ID_COMBOBOX2, CB_ADDSTRING, 0,  
(LPARAM)Lista[i]);  
        SendDlgItemMessage(hDlg, ID_COMBOBOX3, CB_ADDSTRING, 0,  
(LPARAM)Lista[i]);  
    }  
}
```

```

    SendDlgItemMessage(hDlg, ID_COMBOBOX1, CB_SELECTSTRING,
(WPARAM)-1, (LPARAM)Datos.Item[0]);
    SendDlgItemMessage(hDlg, ID_COMBOBOX2, CB_SELECTSTRING,
(WPARAM)-1, (LPARAM)Datos.Item[1]);
    SendDlgItemMessage(hDlg, ID_COMBOBOX3, CB_SELECTSTRING,
(WPARAM)-1, (LPARAM)Datos.Item[2]);
    SetFocus(GetDlgItem(hDlg, ID_COMBOBOX1));
    return FALSE;

```

También podemos preseleccionar alguna de las cadenas de los comboboxes. Para seleccionar una de las cadenas también se usa un mensaje: [CB_SELECTSTRING](#). Usaremos el valor -1 en wParam para indicar que se busque en todo el listbox.

Devolver valores a la aplicación

También queremos que cuando el usuario está satisfecho con los datos que ha introducido, y pulse el botón de aceptar, el dato de nuestra aplicación se actualice con el texto del ítem seleccionado.

De nuevo podemos recurrir a mensajes para pedirle al combobox el valor de la cadena actualmente seleccionada. En este caso se trata dos mensajes combinados, análogos a los usados en los Listbox. Uno es [CB_GETCURSEL](#), que se usa para averiguar el índice de la cadena actualmente seleccionada. El otro es [CB_GETLBTEXT](#), que devuelve la cadena del índice que le indiquemos.

Cuando trabajemos con memoria dinámica y con ítems de longitud variable, será interesante saber la longitud de la cadena antes de leerla desde el listbox. Para eso podemos usar el mensaje [CB_GETLBTEXTLEN](#).

Pero esto es válido para comboboxes DropDownList, que se comportan como un Listbox, sin embargo en las otras modalidades de controles ComboBox, el ítem seleccionado no tiene por qué ser igual que el texto que contiene el control Edit.

Para capturar el contenido del control Edit asociado a un ComboBox se puede usar la función [GetDlgItemText](#). Y también el mensaje [WM_GETTEXT](#) y [WM_GETTEXTLENGTH](#).

Como tenemos tres tipos de ComboBox, usaremos un método diferente con cada uno de ellos. Veamos cómo podría quedar el tratamiento del mensaje WM_COMMAND:

```

case WM_COMMAND:
    switch(LOWORD(wParam)) {
        case IDOK:
            // En el ComboList Simple usaremos:
            GetDlgItemText(hDlg, ID_COMBOBOX1, Datos.Item[0], 80);
            // En el ComboList DropDown usaremos:

```

```

        SendDlgItemMessage(hDlg, ID_COMBOBOX2, WM_GETTEXT, 80,
(LPARAM)Datos.Item[1]);
        // En el ComboList DropDownList usaremos:
        indice = SendDlgItemMessage(hDlg, ID_COMBOBOX3,
CB_GETCURSEL, 0, 0);
        SendDlgItemMessage(hDlg, ID_COMBOBOX3, CB_GETLBTEXT,
indice, (LPARAM)Datos.Item[2]);
        wsprintf(resultado, "%s\n%s\n%s", Datos.Item[0], Datos.Item[1],
Datos.Item[2]);
        MessageBox(hDlg, resultado, "Leido", MB_OK);
        EndDialog(hDlg, FALSE);
        return TRUE;
    case IDCANCEL:
        EndDialog(hDlg, FALSE);
        return FALSE;

```

Pero ahora surge un problema. Si en un combobox introducimos una cadena que no está en nuestra lista, y posteriormente volvemos a entrar en el cuadro de diálogo, no podremos editar el valor inicial, ni siquiera nos será mostrado.

Para evitar eso deberíamos añadir cada nuevo valor introducido a la lista. Nuestro ejemplo es un poco limitado, ya que no tiene previsto que la lista pueda crecer, y desde luego, no guardará los valores de la lista cuando el programa termine, de modo que estén disponibles en sucesivas ejecuciones. Pero de momento nos conformaremos con ciertas modificaciones mínimas que ilustren cómo solventar este error.

Para empezar, reservaremos espacio suficiente para almacenar algunos valores extra:

```

#define MAX_CADENAS 100
...

int nCadenas;
char Lista[MAX_CADENAS][80] = {
    "aaaaaaaaaaaa",
    "bbbbbbbbbbbb",
    "cccccccccccc",
    "rrrrrrrrrr",
    "ffffffffffff",
    "dddddddddddd"
};

```

También deberemos inicializar el valor de nCadenas, dentro de la función "WinMain":

```
nCadenas = 6;
```

Modificaremos la rutina para inicializar los ComboBoxes:

```

// Añadir cadenas. Mensaje: LB_ADDSTRING
for(i = 0; i < nCadenas; i++) {
    SendDlgItemMessage(hDlg, ID_COMBOBOX1, CB_ADDSTRING, 0,
(LPARAM)Lista[i]);
    SendDlgItemMessage(hDlg, ID_COMBOBOX2, CB_ADDSTRING, 0,
(LPARAM)Lista[i]);
    SendDlgItemMessage(hDlg, ID_COMBOBOX3, CB_ADDSTRING, 0,
(LPARAM)Lista[i]);
}
...

```

Y para leer los valores introducidos, y añadirlos a la lista si no están. Para eso usaremos el mensaje [CB_FINDSTRINGEXACT](#), que buscará una cadena entre los valores almacenados en la lista, si se encuentra devolverá un índice, y si no, el valor CB_ERR.

Exite otro mensaje parecido, [CB_FINDSTRING](#), pero no nos vale, porque localizará la primera cadena de la lista que comience con los mismos caracteres que la cadena que buscamos. Por ejemplo, si hay un valor en la lista "valor a", y nosotros buscamos "valor", obtendremos el índice de la cadena "valor a", que no es lo que queremos, al menos en este ejemplo.

El proceso del comando IDOK quedaría así:

```

case IDOK:
    // En el ComboList Simple usaremos:
    GetDlgItemText(hDlg, ID_COMBOBOX1, Datos.Item[0], 80);
    if(SendDlgItemMessage(hDlg, ID_COMBOBOX1,
CB_FINDSTRINGEXACT,
(WPARAM)-1, (LPARAM)Datos.Item[0]) == CB_ERR)
        strcpy(Lista[nCadenas++], Datos.Item[0]);
    // En el ComboList DropDown usaremos:
    SendDlgItemMessage(hDlg, ID_COMBOBOX2, WM_GETTEXT, 80,
(LPARAM)Datos.Item[1]);
    if(SendDlgItemMessage(hDlg, ID_COMBOBOX1,
CB_FINDSTRINGEXACT,
(WPARAM)-1, (LPARAM)Datos.Item[1]) == CB_ERR &&
strcmp(Datos.Item[0], Datos.Item[1]))
        strcpy(Lista[nCadenas++], Datos.Item[1]);
    // En el ComboList DropDownList usaremos:
    indice = SendDlgItemMessage(hDlg, ID_COMBOBOX3,
CB_GETCURSEL, 0, 0);
    SendDlgItemMessage(hDlg, ID_COMBOBOX3, CB_GETLBTEXT,
indice, (LPARAM)Datos.Item[2]);
    wsprintf(resultado, "%s\n%s\n%s", Datos.Item[0], Datos.Item[1],
Datos.Item[2]);
    MessageBox(hDlg, resultado, "Leido", MB_OK);
    EndDialog(hDlg, FALSE);
    return TRUE;

```


Ejemplo 10: 02/04/2001

ejemplo10.c 02/04/2001 (5.209 bytes)

ids.h 02/04/2001 (166 bytes)

rsrc.rc 02/04/2001 (1.217 bytes)

win10.dev 02/04/2001 (369 bytes)  (Alternativo: )

Capítulo 12 Control básico Scrollbar

Veremos ahora el siguiente control básico: la barra de desplazamiento o Scrollbar.

Las ventanas pueden mostrar contenidos que ocupan más espacio del que cabe en su interior, cuando eso sucede se suelen agregar unos controles en forma de barra que permiten desplazar el contenido a través del área de la ventana de modo que el usuario pueda ver las partes ocultas del documento.

Pero las barras de scroll pueden usarse para introducir otros tipos de datos en nuestras aplicaciones, en general, cualquier magnitud de la que sepamos el máximo y el mínimo, y que tenga un rango valores finito. Por ejemplo un control de volumen, de 0 a 10, o un termostato de -15° a 60° .

Las barras de desplazamiento tienen varias partes o zonas diferenciadas, cada una con su función particular. Me imagino que ya las conoces, pero las veremos desde el punto de vista de un programador.

Una barra de desplazamiento consiste en un rectángulo sombreado con un botón de flecha en cada extremo, y una caja en el interior del rectángulo (llamado normalmente thumb). La barra de desplazamiento representa la longitud o anchura completa del documento, y la caja interior la porción visible del documento dentro de la ventana. La posición de la caja cambia cada vez que el usuario desplaza el documento para ver diferentes partes de él. También se modifica el tamaño de la caja para adaptarlo a la proporción del documento que es visible. Cuanta más porción del documento resulte visible, mayor será el tamaño de la caja, y viceversa.

Hay dos modalidades de ScrollBars: horizontales y verticales.

El usuario puede desplazar el contenido de la ventana pulsando uno de los botones de flecha, pulsando en la zona sombreada no ocupada por el thumb, o desplazando el propio thumb. En el primer caso se desplazará el equivalente a una unidad (si es texto, una línea o columna). En el segundo, el contenido se desplazará en la porción equivalente al contenido de una ventana. En el tercer caso, la cantidad de documento desplazado dependerá de la distancia que se desplace el thumb.

Hay que distinguir los controles ScrollBar de las barras de desplazamiento estándar. Aparentemente son iguales, y se comportan igual, los primeros están en el área de cliente de la ventana, pero las segundas no, éstas se crean y se muestran junto con la ventana. Para añadir estas barras a tu ventana, basta con crearla con los estilos `WS_HSCROLL`, `WS_VSCROLL` o ambos. `WS_HSCROLL` añade una barra horizontal y `WS_VSCROLL` una vertical.

Un control scroll bar es una ventana de control de la clase `SCROLLBAR`. Se pueden crear tantas barras de scroll como se quiera, pero el programador es el encargado de especificar el tamaño y la posición de la barra.

Ficheros de recursos

Para nuestro ejemplo incluiremos un control ScrollBar de cada tipo, aunque en realidad son casi idénticos en comportamiento:

```
DialogoPrueba DIALOG 0, 0, 189, 106
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
CAPTION "Scroll bars"
FONT 8, "Helv"
{
CONTROL "ScrollBar1", ID_SCROLLH, "SCROLLBAR", SBS_HORZ |
WS_CHILD | WS_VISIBLE, 7, 3, 172, 9
CONTROL "Scroll 1:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE,
24, 18, 32, 8
CONTROL "Edit1", ID_EDITH, "EDIT", ES_LEFT | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 57, 15, 32, 12
CONTROL "ScrollBar2", ID_SCROLLV, "SCROLLBAR", SBS_VERT |
WS_CHILD | WS_VISIBLE, 7, 15, 9, 86
CONTROL "Scroll 2:", -1, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE,
23, 41, 32, 8
CONTROL "Edit2", ID_EDITV, "EDIT", ES_LEFT | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 23, 51, 32, 12
CONTROL "Aceptar", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 40, 87, 50, 14
}
```

Para ver cómo funcionan las barras de scroll hemos añadido dos controles Edit, que mostrarán los valores seleccionados en cada control ScrollBar. Para más detalles acerca de estos controles ver [controles scrollbar](#).

Ahora veremos más detalles sobre los estilos de los controles ScrollBar:

```
CONTROL "ScrollBar1", ID_SCROLLH, "SCROLLBAR", SBS_HORZ |
WS_CHILD | WS_VISIBLE, 7, 3, 172, 9
CONTROL "ScrollBar2", ID_SCROLLV, "SCROLLBAR", SBS_VERT |
WS_CHILD | WS_VISIBLE, 7, 15, 9, 86
```

- CONTROL es la palabra clave que indica que vamos a definir un control.
- A continuación, en el parámetro text, en el caso de los scrollbar sólo sirve como información, y no se usa.
- id es el identificador del control. El identificador será necesario para inicializar y leer el valor del scrollbar, así como para manipular los mensajes que produzca.
- class es la clase de control, en nuestro caso "SCROLLBAR".
- style es el estilo de control que queremos. En nuestro caso es una combinación de un [estilo scrollbar](#) y varios [de ventana](#):
 - SBS_HORZ: Indica se trata de un scrollbar horizontal.

- SBS_VERT: Indica se trata de un scrollbar vertical.
- WS_CHILD: crea el control como una ventana hija.
- WS_VISIBLE: crea una ventana inicialmente visible.
- coordenada x del control.
- coordenada y del control.
- width: anchura del control.
- height: altura del control.

Iniciar controles Scrollbar

Los controles Scrollbar tienen varios tipos de parámetros que hay que iniciar. Lo primero los límites de valores mínimo y máximo. Y también el valor actual.

El lugar adecuado para hacerlo también es al procesar el mensaje [WM_INITDIALOG](#) de nuestro cuadro de diálogo, y para ajustar los parámetros podemos usar mensajes o funciones. En el caso de hacerlo con mensajes hay que usar la función [SendDlgItemMessage](#).

También usaremos una estructura para almacenar los valores iniciales y de la última selección de las dos barras de desplazamiento:

```
// Datos de la aplicación
typedef struct stDatos {
    int ValorH;
    int ValorV;
} DATOS;
```

```
DATOS Datos;
```

Y asignaremos valores iniciales a los controles dentro de "WinMain":

```
Datos.ValorH = 10;
Datos.ValorV = 32;
```

La parte de inicialización de los scrollbars quedaría así, hemos inicializado el scrollbar horizontal usando las funciones y el vertical usando los mensajes::

```
case WM_INITDIALOG:
    SetScrollRange(GetDlgItem(hDlg, ID_SCROLLH), SB_CTL, 0, 100, true);

    SetScrollPos(GetDlgItem(hDlg, ID_SCROLLH), SB_CTL, Datos.ValorH,
true);
    SetDlgItemInt(hDlg, ID_EDITH, (UINT)Datos.ValorH, FALSE);
    SendDlgItemMessage(hDlg, ID_SCROLLV, SBM_SETRANGE,
(WPARAM)0, (LPARAM)50);
    SendDlgItemMessage(hDlg, ID_SCROLLV, SBM_SETPOS,
(WPARAM)Datos.ValorV, (LPARAM>true);
    SetDlgItemInt(hDlg, ID_EDITV, (UINT)Datos.ValorV, FALSE);
```

```
return FALSE;
```

Para iniciar el rango de valores del scrollbar se usa la función [SetScrollRange](#) o el mensaje [SBM_SETRANGE](#). Para cambiar el valor seleccionado o posición se usa la función [SetScrollPos](#) o el mensaje [SBM_SETPOS](#).

Iniciar controles scrollbar: estructura SCROLLINFO

A partir de la versión 4.0 de Windows existe otro mecanismo para inicializar los scrollbars. También tiene la doble forma de mensaje y función. Su uso se recomienda en lugar de [SetScrollRange](#), que sólo se conserva por compatibilidad con Windows 3.x.

Se trata de la función [SetScrollInfo](#) y del mensaje [SBM_SETSCROLLINFO](#). También es necesaria una estructura que se usará para pasar los parámetros tanto a la función como al mensaje: [SCROLLINFO](#).

Usando esta forma, el ejemplo anterior quedaría así:

```
SCROLLINFO sih = {
    sizeof(SCROLLINFO),
    SIF_POS | SIF_RANGE | SIF_PAGE,
    0, 104,
    5,
    Datos.ValorH,
    0};
SCROLLINFO siv = {
    sizeof(SCROLLINFO),
    SIF_POS | SIF_RANGE | SIF_PAGE,
    0, 54,
    5,
    Datos.ValorV,
    0};

...
case WM_INITDIALOG:
    SetScrollInfo(GetDlgItem(hDlg, ID_SCROLLH), SB_CTL, &sih, true);
    SetDlgItemInt(hDlg, ID_EDITH, (UINT)Datos.ValorH, FALSE);
    SendDlgItemMessage(hDlg, ID_SCROLLV, SBM_SETSCROLLINFO,
        (WPARAM)TRUE, (LPARAM)&siv);
    SetDlgItemInt(hDlg, ID_EDITV, (UINT)Datos.ValorV, FALSE);
    return FALSE;
```

El segundo campo de la estructura SCROLLINFO consiste en varios bits que indican qué parámetros de la estructura se usarán para inicializar los scrollbars. Hemos incluido la posición, el rango y el valor de la página. Sería equivalente haber puesto únicamente SIF_ALL.

El valor de la página no lo incluimos antes, y veremos que será útil al procesar los mensajes que provienen de los controles scrollbar. Además el tamaño de la caja de desplazamiento se ajusta de modo que esté a escala en relación con el tamaño total del control scrollbar. Si hubiéramos definido una página de 50 y un rango de 0 a 100, el tamaño de la caja sería exactamente la mitad del tamaño del scrollbar.

Hay que tener en cuenta que el valor máximo que podremos establecer en un control no es siempre el que nosotros indicamos en el miembro nMax de la estructura SCROLLINFO. Este valor depende del valor de la página (nPage), y será nMax-nPage+1. Así que si queremos que nuestro control pueda devolver 100, y la página tiene un valor de 5, debemos definir nMax como 104. Este funcionamiento está diseñado para scrollbars como los que incluyen las ventanas, donde la caja indica la porción del documento que se muestra en su interior.

Procesar los mensajes procedentes de controles Scrollbar

Cada vez que el usuario realiza alguna acción en un control Scrollbar se envía un mensaje [WM_HSCROLL](#) o [WM_VSCROLL](#), dependiendo del tipo de control, a la ventana donde está insertado. En realidad eso pasa con todos los controles, pero en el caso de los Scrollbars, es imprescindible que el programa procese algunos de esos mensajes adecuadamente.

Estos mensajes entregan distintos valores en la palabra de menor peso del parámetro wParam, según la acción del usuario sobre el control. En la palabra de mayor peso se incluye la posición actual y en lParam el manipulador de ventana del control.

De modo que nuestra rutina para manejar los mensajes de los scrollbars debe ser capaz de distinguir el control del que procede el mensaje y el tipo de acción, para actuar en consecuencia.

En nuestro caso es irrelevante la orientación de la barra de scroll, podemos distinguirlos por el identificador de ventana, de todos modos procesaremos cada uno de los dos mensajes con una rutina distinta.

Para no recargar en exceso el procedimiento de ventana del diálogo, crearemos una función para procesar los mensajes de las barras de scroll. Y la llamaremos al recibir esos mensajes:

```
switch (msg)          /* manipulador del mensaje */
{
...
case WM_HSCROLL:
case WM_VSCROLL:
    ProcesarScroll((HWND)lParam, (int)(LOWORD)wParam);
    return FALSE;
...
}
```

Los códigos que tenemos que procesar son los siguientes:

- SB_BOTTOM desplazamiento hasta el principio de la barra, en verticales arriba y en horizontales a la izquierda.
- SB_TOP: desplazamiento hasta el final de la barra, en verticales abajo y en horizontales a la derecha.
- SB_LINERIGHT y SB_LINEDOWN: desplazamiento una línea a la derecha en horizontales o abajo en verticales.
- SB_LINELEFT y SB_LINEUP: desplazamiento una línea a la izquierda en horizontales o arriba en verticales.
- SB_PAGERIGHT y SB_PAGEDOWN: desplazamiento de una página a la derecha en horizontales y abajo en verticales.
- SB_PAGELEFT y SB_PAGEUP: desplazamiento un párrafo a la izquierda en horizontales o arriba en verticales.
- SB_THUMBPOSITION: se envía cuando el thumb está en su posición final.
- SN_THUMBTRACK: el thumb se está moviendo.
- SB_ENDSCROLL: el usuario ha liberado el thumb en una nueva posición.

En nuestro caso, no haremos que la variable asociada se actualice hasta que pulsemos el botón de aceptar, pero actualizaremos la posición del thumb y el valor del control edit asociado a cada scrollbar.

Veamos por ejemplo la rutina para tratar el scroll horizontal:

```
void ProcesarScrollH(HWND hDlg, HWND Control, int Codigo, int Posicion)
{
    int Pos = GetScrollPos(Control, SB_CTL);

    switch(Codigo) {
        case SB_BOTTOM:
            Pos = 0;
            break;
        case SB_TOP:
            Pos = 100;
            break;
        case SB_LINERIGHT:
            Pos++;
            break;
        case SB_LINELEFT:
            Pos--;
            break;
        case SB_PAGERIGHT:
            Pos += 5;
            break;
        case SB_PAGELEFT:
            Pos -= 5;
            break;
        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:
```

```

        Pos = Posicion;
    case SB_ENDSCROLL:
        break;
    }
    if(Pos < 0) Pos = 0;
    if(Pos > 100) Pos = 100;
    SetDlgItemInt(hDlg, ID_EDITH, (UINT)Pos, FALSE);

    SetScrollPos(Control, SB_CTL, Pos, true);
}

```

Como puede observarse, actualizamos el valor de la posición dependiendo del código recibido. Nos aseguramos de que está dentro de los márgenes permitidos y finalmente actualizamos el contenido del control edit. La función para el scrollbar vertical es análoga, pero cambiando los identificadores de los códigos y los valores de los límites.

Hemos usado una función nueva, [GetScrollPos](#) para leer la posición actual del thumb.

Procesar mensajes de scrollbar usando SCROLLINFO

Como comentamos antes, a partir de la versión 4.0 de Windows existe otro mecanismo para inicializar los scrollbars. También tiene la doble forma de mensaje y función. Su uso se recomienda en lugar de `GetScrollRange`, que sólo se conserva por compatibilidad con Windows 3.x.

Usando [GetScrollInfo](#), la función para procesar el mensaje del scrollbar horizontal quedaría así:

```

void ProcesarScrollH(HWND hDlg, HWND Control, int Codigo, int Posicion)
{
    SCROLLINFO si = {
        sizeof(SCROLLINFO),
        SIF_ALL, 0, 0, 0, 0, 0};

    GetScrollInfo(Control, SB_CTL, &si);

    switch(Codigo) {
        case SB_BOTTOM:
            si.nPos = si.nMin;
            break;
        case SB_TOP:
            si.nPos = si.nMax;
            break;
        case SB_LINERIGHT:
            si.nPos++;
    }
}

```



```

        break;
    case SB_LINELEFT:
        si.nPos--;
        break;
    case SB_PAGERIGHT:
        si.nPos += si.nPage;
        break;
    case SB_PAGELEFT:
        si.nPos -= si.nPage;
        break;
    case SB_THUMBPOSITION:
    case SB_THUMBTRACK:
        si.nPos = Posicion;
    case SB_ENDSCROLL:
        break;
}
if(si.nPos < si.nMin) si.nPos = si.nMin;
if(si.nPos > si.nMax-si.nPage+1) si.nPos = si.nMax-si.nPage+1;

SetScrollInfo(Control, SB_CTL, &si, true);
SetDlgItemInt(hDlg, ID_EDITH, (UINT)si.nPos, FALSE);
}

```

Usamos los valores de la estructura para acotar la posición de la caja y para avanzar y retroceder de página, esto hace que nuestra función sea más independiente y que use menos constantes definidas en el fuente.

También se puede usar el mensaje [SBM_GETSCROLLINFO](#), basta con cambiar la línea:

```
GetScrollInfo(Control, SB_CTL, &si);
```

por esta otra:

```
SendMessage(hDlg, ID_SCROLLH, SBM_GETSCROLLINFO, 0,
(LPARAM)&si);
```

Devolver valores a la aplicación

Cuando el usuario ha decidido que los valores son los adecuados pulsará el botón de Aceptar. En ese momento deberemos capturar los valores de los controles scroll y actualizar la estructura de parámetros.

También en este caso podemos usar un mensaje o una función para leer la posición del thumb del scrollbar. La función ya la hemos visto un poco más arriba, se trata de [GetScrollPos](#). El mensaje es [SBM_GETPOS](#), ambos devuelven el valor de la posición actual del thumb del control.

Usaremos los dos métodos, uno con cada control. El lugar adecuado para leer esos valores sigue siendo el tratamiento del mensaje WM_COMMAND:

```
case WM_COMMAND:
    switch(LOWORD(wParam)) {
        case IDOK:
            Datos.ValorH = GetScrollPos(GetDlgItem(hDlg, ID_SCROLLH),
SB_CTL);
            Datos.ValorV = SendDlgItemMessage(hDlg, ID_SCROLLV,
SBM_GETPOS, 0, 0);
            EndDialog(hDlg, FALSE);
            return TRUE;
        case IDCANCEL:
            EndDialog(hDlg, FALSE);
            return FALSE;
```

Ejemplo 11: 23/04/2001


ejemplo11.c 23/04/2001 (5.807 bytes)

ids.h 22/04/2001 (180 bytes)

rsrc.rc 23/04/2001 (1.055 bytes)

win11.dev 23/04/2001 (428 bytes)



(Alternativo: )

Ejemplo 12 (usando SCROLLINFO): 29/04/2001


ejemplo12.c 23/04/2001 (6.354 bytes)

ids.h 22/04/2001 (180 bytes)

rsrc.rc 29/04/2001 (1.055 bytes)

win12.dev 29/04/2001 (428 bytes)



(Alternativo: )



Constantes

Button styles:

Para la clase BUTTON, pueden especificarse los siguientes estilos de button mediante el parámetro dwStyle:

Estilo	Significado
BS_3STATE	Crea un button que es lo mismo que un check box, salvo que puede ponerse gris (grayed) además de ser marcado (checked) o desmarcado (unchecked). El estado gris se usa para mostrar que el check box está indeterminado.
BS_AUTO3STATE	Crea un button que es igual que el check box de tres estados, salvo que cambia de estado cuando el usuario lo selecciona. El estado cambia alternativamente entre checked, grayed y unchecked.
BS_AUTOCHECKBOX	Crea un button que es igual que un check box, pero que cuyo estado de selección oscila automáticamente entre checked y unchecked cada vez que el usuario selecciona el check box.
BS_AUTORADIOBUTTON	Crea un button que es igual que un radio button, pero que cuando es seleccionado por el usuario, Windows cambia su estado automáticamente a seleccionado y automáticamente deselectiona el resto de los radio buttons del mismo grupo.
BS_CHECKBOX	Crea un pequeño check box vacío con texto. Por defecto, el texto se muestra a la derecha del check box. Para mostrar el texto a la izquierda, hay que combinar este flag con el estilo BS_LEFTTEXT (o sólo para Windows 95, con su equivalente BS_RIGHTBUTTON).
BS_DEFPUSHBUTTON	Crea un botón normal que se comporta como uno del estilo BS_PUSHBUTTON, pero también tiene un borde negro y grueso. Si el botón está en un cuadro de diálogo, el usuario puede pulsar este botón usando la tecla ENTER, aún cuando el botón no tenga el foco de entrada. Este estilo es corriente para permitir al usuario seleccionar rápidamente la opción más frecuente, la opción por defecto.
BS_GROUPBOX	Crea un rectángulo en cuyo interior se pueden agrupar otros controles. Cualquier texto asociado con este estilo se mostrará en la esquina superior izquierda del rectángulo.
BS_LEFTTEXT	Coloca un texto a la izquierda de un radio button o check box cuando se combina con los estilos radio button o check box. Lo mismo que el estilo de

	Windows 95 BS_RIGHTBUTTON.
BS_OWNERDRAW	Crea un botón owner-drawn. La ventana propietaria recibirá un mensaje WM_MEASUREITEM cuando el botón sea creado y un mensaje WM_DRAWITEM cuando algún aspecto visual del botón haya cambiado. No debe combinarse el estilo BS_OWNERDRAW con cualquier otro estilo de botón.
BS_PUSHBUTTON	Crea un botón corriente que envía un mensaje WM_COMMAND a su ventana padre cuando el usuario selecciona el botón.
BS_RADIOBUTTON	Crea un pequeño círculo con texto. Por defecto, el texto se muestra a la derecha del círculo. Para mostrar el texto a la izquierda, hay que combinar este flag con el estilo BS_LEFTTEXT (o sólo para Windows 95, con su equivalente BS_RIGHTBUTTON). Se usan para grupos de opciones relacionadas pero mutuamente exclusivas.
BS_USERBUTTON	Obsoleto, pero se mantiene por compatibilidad con versiones de Windows de 16 bits. Las aplicaciones basadas en Win32 deben usar en su lugar BS_OWNERDRAW.
BS_BITMAP	Sólo en Windows 95: indica que el botón muestra un bitmap.
BS_BOTTOM	Sólo en Windows 95: muestra el texto en la parte de abajo del área del botón.
BS_CENTER	Solo en Windows 95: centra el texto horizontalmente en el área del botón.
BS_ICON	Sólo en Windows 95: indica que el botón muestra un icono.
BS_LEFT	Sólo en Windows 95: justifica a la izquierda el texto del botón. Sin embargo, si el botón es un check box o un radio button que no tiene el estilo BS_RIGHTBUTTON, el texto será justificado a la izquierda, pero al lado derecho del check box o radio button.
BS_MULTILINE	Sólo en Windows 95: divide el texto del botón en varias líneas si es demasiado largo para que quepa en una sólo línea en el área del botón.
BS_NOTIFY	Sólo en Windows 95: permite al botón enviar mensaje de notificación BN_DBLCLK, BN_KILLFOCUS y BN_SETFOCUS a su ventana padre. Observa que los botones envían el mensaje BN_CLICKED a pesar de poseer este estilo.
BS_PUSHLIKE	Sólo en Windows 95: hace un botón (como un check

	box, three-state check box o radio button) que se comporta y tiene el mismo aspecto que un botón normal. El botón se muestra levantado cuando no está pulsado o unchecked y hundido cuando esté pulsado o checked.
BS_RIGHT	Sólo en Windows 95: justifica a la derecha el texto del botón. Sin embargo, si el botón es un check box o un radio button que no tiene el estilo BS_RIGHTBUTTON, el texto será justificado a la derecha, pero al lado derecho del check box o radio button.
BS_RIGHTBUTTON	Sólo en Windows 95: coloca el círculo del radio button o el cuadrado del check box al lado derecho del área del botón. Lo mismo que el estilo BS_LEFTTEXT.
BS_TEXT	Sólo en Windows 95: indica que el botón muestra un texto.
BS_TOP	Sólo en Windows 95: muestra el texto en la parte de arriba del área del botón.
BS_VCENTER	Sólo en Windows 95: muestra el texto en el centro, verticalmente, del área del botón.

Combo box styles:

Para la clase COMBOBOX, pueden especificarse los siguientes estilos mediante el parámetro dwStyle:

Estilo	Significado
CBS_AUTOHSCROLL	Desplaza automáticamente a la derecha el texto en un control edit cuando el usuario escribe al final de la línea. Si este estilo no se selecciona, sólo puede introducirse el texto que cabe en los límites del cuadro de edición.
CBS_DISABLENOSCROLL	Muestra la barra de scroll vertical deshabilitada en el list box cuando no contiene suficientes elementos para desplazarlos. Sin este estilo la barra de scroll se oculta si no hay suficientes elementos en la lista.
CBS_DROPDOWN	Igual que CBS_SIMPLE, salvo que no se muestra el list box si el usuario no selecciona el icono cercano al control edit.
CBS_DROPDOWNLIST	Igual que CBS_DROPDOWN, salvo que el control edit se sustituye por un static text que muestra la selección actual del list box.
CBS_HASSTRINGS	Especifica que un combo box owner-drawn (actualizado por la ventana padre) contiene elementos que son cadenas. El combo box mantiene la memoria y las direcciones de las cadenas, así que la aplicación puede usar el mensaje CB_GETLBTEXT para recuperar el texto de un elemento en particular.

CBS_LOWERCASE	Convierte los caracteres en minúsculas introducidos en el control edit de un combo box a mayúsculas.
CBS_NOINTEGRALHEIGHT	Especifica que el tamaño del combo box es exactamente el indicado por la aplicación cuando se creó el combo box. Normalmente, Windows cambia el tamaño del combo box para que no se muestren trozos de líneas.
CBS_OEMCONVERT	Convierte en texto introducido en control de edición del combo box. El texto se convierte de juego de caracteres de Windows a de OEM y después vuelve al juego de Windows. Esto asegura una correcta conversión cuando la aplicación llame a la función CharToOem para convertir una cadena Windows del combo box a una cadena OEM. Este estilo es comunmente usado en combo boxes que contienen nombres de fichero y se aplica sólo a combo boxes creados con el estilo CBS_SIMPLE o CBS_DROPDOWN.
CBS_OWNERDRAWFIXED	Especifica que el padre del list box es el responsable de actualizar su contenido en pantalla y de que los elementos del list box sean todos de la misma altura. La ventana padre recibirá el mensaje WM_MEASUREITEM cuando el combo box sea creado y el mensaje WM_DRAWITEM cuando algún aspecto visual del combo box haya cambiado.

CBS_OWNERDRAWVARIABLE	<p>Especifica que el propietario del combo box es el responsable de dibujar su contenido y que los elementos del list box tienen altura variable. La ventana padre recibirá un mensaje WM_MEASUREITEM para cada elemento del combo box cuando el combo box sea creado; y un mensaje WM_DRAWITEM cuando algún aspecto visual del combo box haya cambiado.</p>
CBS_SIMPLE	<p>Muestra el list box todo el tiempo. La selección actual del list box se muestra en el control edit.</p>
CBS_SORT	<p>Ordena automáticamente las cadenas introducidas en el list box.</p>
CBS_UPPERCASE	<p>Convierte los caracteres en mayúsculas introducidos en el control edit de un combo box a minúsculas.</p>

Dialog box styles:

Para la Dialog boxes pueden especificarse los siguientes estilos mediante el parámetro dwStyle:

Estilo	Significado
DS_3DLOOK	<p>Sólo para Windows 95: proporciona al dialog box una fuente no enfatizada y dibuja bordes tridimensionales alrededor de las ventanas de control en el cuadro de diálogo.</p> <p>El estilo DS_3DLOOK no se precisa para aplicaciones Win32 que estén marcadas con la versión 4.0 y siguientes; el sistema aplica el aspecto tridimensional automáticamente a los cuadros de diálogo creados por esas aplicaciones.</p>
DS_ABSALIGN	Indica que las coordenadas del cuadro de diálogo son coordenadas de pantalla, de otro modo, Windows asume que se trata de coordenadas de cliente.
DS_CENTER	Sólo para Windows 95: centra el cuadro de diálogo en el área de trabajo; esto es, el área no ocupada por la barra de estado.
DS_CENTERMOUSE	Sólo para Windows 95: centra el cursor del ratón en el cuadro de diálogo.
DS_CONTEXTHELP	Incluye el signo de interrogación en la barra de título del cuadro de diálogo. Cuando el usuario hace click sobre la interrogación, el cursor cambia a una interrogación con un puntero. Si el usuario hace entonces click en un control del cuadro de diálogo, el control recibe el mensaje WM_HELP. El control debe pasar el mensaje al procedimiento del diálogo, el cual a su vez llamará a la función WinHelp usando el comando HELP_WM_HELP. La aplicación Help muestra una ventana pop-up que normalmente contendrá la ayuda

	<p>relativa al control.</p> <p>Observa que DS_CONTEXTHELP es sólo un asa (placeholder). Cuando el cuadro de diálogo es creado, el sistema verifica si hay DS_CONTEXTHELP, si lo hay, añade WS_EX_CONTEXTHELP al estilo extendido del cuadro de diálogo.</p>
DS_CONTROL	<p>Crea un cuadro de diálogo que se comporta como una ventana hija de otro cuadro de diálogo, muy parecido a una página de un property sheet. Este estilo permite al usuario desplazarse entre controles con el tab, usar aceleradores, etc.</p>
DS_FIXEDSYS	<p>Sólo para Windows 95: usa SYSTEM_FIXED_FONT en lugar de SYSTEM_FONT.</p>
DS_LOCALEDIT	<p>Sólo se refiere a aplicaciones de 16 bits. Este estilo hace que los controles edit del dialog box asignen memoria del segmento de datos de la aplicación. De lo contrario los controles edit asignan memoria de la memoria global.</p>
DS_MODALFRAME	<p>Crea un cuadro de diálogo con un marco de dialog-box modal que puede combinarse con una barra de título y un menú de sistema especificando los estilos WS_CAPTION y WS_SYSMENU.</p>
DS_NOFAILCREATE	<p>Sólo para Windows 95: crea el cuadro de diálogo aunque se produzca algún error. Por ejemplo, si una ventana hija no pudo ser creada o el sistema no puede crear un segmento de datos especial para un control edit.</p>
DS_NOIDLEMSG	<p>Suprime los mensajes WM_ENTERIDLE que Windows de otro modo enviará a la ventana padre del cuadro de diálogo mientras éste sea mostrado.</p>
DS_RECURSE	<p>Estilo para cuadros de diálogo parecidos a controles (ontrol-like dialog boxes).</p>

DS_SETFONT	Indica que la plantilla del cuadro de diálogo (la estructura DLGTEMPLATE) contiene dos miembros adicionales que especifican el nombre de una fuente y el tamaño en puntos. La fuente correspondiente será usada para mostrar el texto en el interior del área de cliente del cuadro de diálogo y en sus controles. Windows pasa el manipulador de la fuente al cuadro de diálogo y a cada control enviando el mensaje WM_SETFONT.
DS_SETFOREGROUND	No usar a aplicaciones de Windows de 16 bits. Este estilo lleva el cuadro de diálogo al primer plano. Internamente, Windows llama a la función SetForegroundWindow para el cuadro de diálogo.
DS_SYSMODAL	Crea un cuadro de diálogo system-modal. Este estilo hace que el cuadro de diálogo tenga el estilo WS_EX_TOPMOST, pero por otro lado no tiene ningún otro efecto en el cuadro de diálogo o en el comportamiento de otras ventanas del sistema cuando el cuadro de diálogo es mostrado.

Edit styles:

Para la clase EDIT, pueden especificarse los siguientes estilos mediante el parámetro dwStyle:

Estilo	Significado
ES_AUTOHSCROLL	Desplaza automáticamente 10 caracteres a la derecha el texto cuando el usuario introduce caracteres al final de la línea. cuando el usuario presiona la tecla de ENTER, el control desplaza el texto de nuevo a la posición cero.
ES_AUTOVSCROLL	Desplaza automáticamente el texto una página arriba cuando el usuario presiona la tecla de ENTER en la última línea.
ES_CENTER	Centra el texto en un control edit multilínea.
ES_LEFT	Alinea el texto a la izquierda.
ES_LOWERCASE	Convierte todos los caracteres introducidos en el edit a minúsculas.
ES_MULTILINE	<p>Indica que se trata de un control edit multilínea. Por defecto los controles edit son de una sola línea.</p> <p>Cuando un edit multilínea está en un cuadro de diálogo (dialog box), la respuesta por defecto cuando se presiona la tecla ENTER es activar el botón por defecto. Para usar la tecla ENTER como retorno de línea, hay que usar el estilo ES_WANTRETURN.</p> <p>Cuando un edit multilínea no está en un cuadro de diálogo y se especifica el estilo, el control edit muestra tantas líneas como sea posible y desplaza verticalmente cuando el usuario presiona la tecla de ENTER. Si no se especifica ES_AUTOVSCROLL, el control edit muestra tantas líneas como sea posible y emite un pitido si el usuario presiona la tecla ENTER cuando no pueden mostrarse más líneas.</p> <p>Si se especifica el estilo</p>

	<p>ES_AUTOHSCROLL, el control edit desplaza horizontalmente de forma automática cuando el cursor (caret) pasa sobre el borde derecho del control. Para comenzar una línea nueva, el usuario debe presionar la tecla de ENTER. Si no se especifica el estilo ES_AUTOHSCROLL, el control se desplaza automáticamente a la siguiente línea, sin cortar las palabras, cuando sea necesario. Se comenzará una línea nueva también si el usuario presiona la tecla ENTER. El tamaño de la ventana determina el punto de corte entre palabras. Si el tamaño de la ventana cambia, estas divisiones cambiarán u el texto será mostrado de nuevo.</p> <p>Los edit multilinea pueden tener barras de scroll (scroll bars). Un control edit con scroll bars procesa sus propios mensajes de scroll. Sin embargo, son controles edit, como los descritos en los párrafos anteriores, y procesaran cualquier mensaje de scroll que provenga de la ventana padre.</p>
ES_NOHIDSESEL	<p>Invalida el comportamiento por defecto de un control edit. Por defecto, se oculta una selección cuando el control pierde el foco de entrada e muestra en inverso la selección cuando el control recupera el foco de entrada. Si se especifica el estilo ES_NOHIDSESEL, el texto seleccionado será invertido, aún cuando el control no tenga el foco de entrada.</p>
ES_NUMBER	<p>Solo para Windows 95: sólo permite introducir dígitos en el control edit.</p>
ES_OEMCONVERT	<p>Convierte en texto introducido en el control edit. El texto se convierte de juego de caracteres de Windows a de OEM y después vuelve al juego de Windows. Esto asegura una correcta conversión cuando la aplicación llame a la función CharToOem para convertir una cadena Windows del combo box a una cadena OEM. Este estilo es comunmente usado en combo boxes que contienen nombres de fichero.</p>
ES_PASSWORD	<p>Muestra un asterisco (*) para cada</p>

	caracter introducido en el control edit. Se puede usar el mensaje EM_SETPASSWORDCHAR para cambiar el carácter que será mostrado en lugar del asterisco.
ES_READONLY	Impide que el usuario pueda escribir o editar texto en el control edit.
ES_RIGHT	Alinea el texto a la izquierda en un control edit multilínea.
ES_UPPERCASE	Convierte todos los caracteres introducidos en el edit a mayúsculas.
ES_WANTRETURN	Indica que se insertará un retorno de línea cuando el usuario presione la tecla ENTER mientras introduce texto en un control edit multilínea dentro de un dialog box. Si no se especifica este estilo, presionar la tecla ENTER tendrá el mismo efecto que presionar el botón por defecto del dialog box. Este estilo no afecta a controles edit de una sola línea.

List box styles:

Para la clase LISTBOX, pueden especificarse los siguientes estilos mediante el parámetro dwStyle:

Estilo	Significado
LBS_DISABLENOSCROLL	Muestra una barra de scroll vertical deshabilitada para el list box cuando no contiene suficientes elementos como para desplazarlos. Si no se especifica éste estilo, la barra de scroll se oculta cuando el list box no contiene suficientes elementos.
LBS_EXTENDEDSEL	Permite que varios elementos puedan ser seleccionados usando la tecla SHIFT y el ratón o combinaciones especiales de teclas.
LBS_HASSTRINGS	Indica que el list box contiene elementos que son cadenas. El list box mantiene la memoria y las direcciones de las cadenas, así que la aplicación puede usar el mensaje LB_GETTEXT para recuperar el texto de un elemento en particular. Por defecto, todas los list boxes, menos los owner-drawn, tienen este estilo. Se puede crear un list box de tipo owner-drawn tanto con o sin este estilo.
LBS_MULTICOLUMN	Indica un list box multicolumna que es desplazado horizontalmente. El mensaje LB_SETCOLUMNWIDTH ajusta en ancho de las columnas.
LBS_MULTIPLESEL	Cambia la selección de cada cadena, cada vez que el usuario hace click o doble click sobre una cadena del list box. El usuario puede seleccionar cualquier número de

	elementos.
LBS_NODATA	<p>Indica que es un list box sin datos. Debe especificarse este estilo cuando el número de elementos pueda exceder de mil. Una lista de este tipo debe tener también el estilo LBS_OWNERDRAWFIXED, pero no debe tener los estilos LBS_SORT o LBS_HASSTRINGS.</p> <p>Un list box no-data se parece a un list box owner-drawn excepto que no contiene cadenas o bitmaps para cada elemento. Los comandos para añadir, insertar o borrar un elemento ignorarán cualquier dato del elemento; peticiones para encontrar una cadena en el interior del list box siempre fallarán. Windows envía el mensaje WM_DRAWITEM a la ventana padre cuando un elemento deba ser mostrado.</p> <p>Se pasará el miembro itemID de la estructura DRAWITEMSTRUCT junto con el mensaje WM_DRAWITEM para especificar el número de línea del elemento a mostrar. Un list box no-data no envía el mensaje WM_DELETEITEM.</p>
LBS_NOINTEGRALHEIGHT	<p>Especifica que el tamaño del list box es exactamente el indicado por la aplicación cuando se creó el list box. Normalmente, Windows cambia el tamaño del list box para que no se muestren trozos de líneas.</p>
LBS_NOREDRA	<p>Indica que la apariencia del list box no se actualiza cuando se hagan cambios. Se puede cambiar este estilo en cualquier</p>

	momento enviando un mensaje WM_SETREDRAW.
LBS_NOTIFY	Informa a la ventana padre con un mensaje de entrada cada vez que el usuario hace click o doble click sobre una cadena del list box.
LBS_OWNERDRAWFIXED	Especifica que el padre del list box es el responsable de actualizar su contenido en pantalla y de que los elementos del list box sean todos de la misma altura. La ventana padre recibirá el mensaje WM_MEASUREITEM cuando el list box sea creado y el mensaje WM_DRAWITEM cuando algún aspecto visual del list box haya cambiado.
LBS_OWNERDRAWVARIABLE	Especifica que el propietario del list box es el responsable de dibujar su contenido y que los elementos del list box tienen altura variable. La ventana padre recibirá un mensaje WM_MEASUREITEM para cada elemento del list box cuando el list box sea creado; y un mensaje WM_DRAWITEM cuando algún aspecto visual del list box haya cambiado.
LBS_SORT	Ordena alfabéticamente las cadenas en el list box.
LBS_STANDARD	Ordena alfabéticamente las cadenas en el list box. La ventana padre recibe un mensaje de entrada cada vez que el usuario hace click o doble click sobre una cadena. El list box tiene bordes en todos sus lados.
LBS_USETABSTOPS	Permite al list box reconocer y expandir los caracteres tab cuando muestra cadenas. Por defecto, las posiciones de los tabs son 32 unidades de cuadro de diálogo. Una unidad de

	<p>cuadro de diálogo es una distancia vertical u horizontal. Una unidad de cuadro de diálogo horizontal es un cuarto de la unidad básica de anchura del cuadro de diálogo actual. Windows calcula estas unidades basándose en la altura y anchura de la fuente de caracteres de sistema actual. la función GetDialogBaseUnits devuelve las unidades base actuales del cuadro de diálogo en pixels.</p>
LBS_WANTKEYBOARDINPUT	<p>Indica que la ventana padre del list box recibe mensajes WM_VKEYTOITEM cada vez que el usuario presiona una tecla y el list box tiene el foco de entrada. Esto permite a la aplicación realizar un procesamiento especial del teclado.</p>

Valores de nCmdShow:

Valores posibles para el parámetro **nCmdShow** de [WinMain](#) y de [ShowWindow](#):

Valor	Significado
SW_HIDE	Oculto la ventana y activa una distinta.
SW_MAXIMIZE	Maximiza la ventana especificada.
SW_MINIMIZE	Minimiza la ventana especificada y activa la siguiente de mayor nivel en la lista Z de ventanas.
SW_RESTORE	Activa y muestra la ventana. Si la ventana está minimizada o maximizada, se restaurará a su tamaño y posición original. Una aplicación debe especificar esta opción cuando restaure una ventana minimizada.
SW_SHOW	Activa la ventana y la muestra en su tamaño y posición actuales.
SW_SHOWDEFAULT	Ajusta el modo de mostrar la ventana basado en los flags SW_ especificados en la estructura STARTUPINFO pasada a la función CreateProcess por el programa que lanza la aplicación. Una aplicación debe llamar a ShowWindow con este flag para ajustar en estado inicial de su ventana principal.
SW_SHOWMAXIMIZED	Activa la ventana y la muestra como una ventana maximizada.
SW_SHOWMINIMIZED	Activa la ventana y la muestra como una ventana minimizada.
SW_SHOWMINNOACTIVE	Muestra como una ventana minimizada. La ventana activa permanece activa.
SW_SHOWNA	Muestra la ventana en su estado actual. La ventana activa permanece activa.
SW_SHOWNOACTIVATE	Muestra la ventana en su estado más reciente de tamaño y posición. La ventana activa permanece activa.

SW_SHOWNORMAL	Activa y muestra una ventana. Si la ventana está minimizada o maximizada, se restaurará a su tamaño y posición originales. Una aplicación debe especificar este flag cuando muestre la ventana por primera vez.
---------------	---

Scroll bar styles:

Para la clase SCROLLBAR, pueden especificarse los siguientes estilos mediante el parámetro dwStyle:

Estilo	Significado
SBS_BOTTOMALIGN	Alinea el borde inferior del scroll bar con el borde inferior del rectángulo definido por los parámetros x, y, nWidth y nHeight de la función CreateWindow . El scroll bar tiene la altura por defecto para los scroll bars del sistema. Usar este estilo junto con el estilo SBS_HORZ.
SBS_HORZ	Especifica un scroll bar horizontal. Si no se especifican los estilos SBS_BOTTOMALIGN o SBS_TOPALIGN, el scroll bar tendrá la altura, anchura y posición especificados por los parámetros de la función CreateWindow .
SBS_LEFTALIGN	Alinea el borde izquierdo del scroll bar con el borde izquierdo del rectángulo definido por los parámetros de la función CreateWindow . El scroll bar tiene la anchura por defecto para los scroll bars del sistema. Usar este estilo junto con el estilo SBS_VERT.
SBS_RIGHTALIGN	Alinea el borde derecho del scroll bar con el borde derecho del rectángulo definido por los parámetros de la función CreateWindow . El scroll bar tiene la anchura por defecto para los scroll bars del sistema. Usar este estilo junto con el estilo SBS_VERT.
SBS_SIZEBOX	Especifica una caja de tamaño (size box). Si no se especifican ninguno de los estilos SBS_SIZEBOXBOTTOMRIGHTALIGN o SBS_SIZEBOXTOPLEFTALIGN, el size box tendrá la altura, anchura y posición especificados por los parámetros de CreateWindow .
SBS_SIZEBOXBOTTOMRIGHTALIGN	Alinea la esquina inferior derecha del size box con la esquina inferior derecha del rectángulo definido por los parámetros de la función CreateWindow . El size box tiene el tamaño por defecto para los size

	<p>boxes del sistema. Usar este estilo junto con el estilo SBS_SIZEBOX.</p>
SBS_SIZEBOXTOPLEFTALIGN	<p>Alinea la esquina superior izquierda del size box con la esquina superior izquierda del rectángulo definido por los parámetros de la función CreateWindow. El size box tiene el tamaño por defecto para los size boxes del sistema. Usar este estilo junto con el estilo SBS_SIZEBOX.</p>
SBS_SIZEGRIP	<p>Sólo para Windows 95: lo mismo que SBS_SIZEBOX, pero con un borde realzado.</p>
SBS_TOPALIGN	<p>Alinea el borde superior del scroll bar con el borde superior del rectángulo definido por los parámetros de la función CreateWindow. El scroll bar tiene la altura por defecto para los scroll bars del sistema. Usar este estilo junto con el estilo SBS_HORZ.</p>
SBS_VERT	<p>Especifica un scroll bar vertical. Si no se especifican los estilos SBS_RIGHTALIGN o SBS_LEFTALIGN, el scroll bar tendrá la altura, anchura y posición especificados por los parámetros de la función CreateWindow.</p>

Static styles:

Para la clase `STATIC`, pueden especificarse los siguientes estilos mediante el parámetro `dwStyle`:

Estilo	Significado
<code>SS_BITMAP</code>	Indica que un bitmap será mostrado en el control static. El texto proporcionado será el nombre de un bitmap (no un nombre de fichero) definido en otro lugar en el fichero de recursos. Este estilo ignora los parámetros <code>nWidth</code> y <code>nHeight</code> ; el control se redimensiona automáticamente para acomodarse al bitmap.
<code>SS_BLACKFRAME</code>	Indica una caja con un marco que se pintará del mismo color que los marcos de la ventana. Este color es negro en el esquema de color por defecto en Windows.
<code>SS_BLACKRECT</code>	Indica un rectángulo relleno con el color de frame actual. Este color es negro en el esquema de color por defecto en Windows.
<code>SS_CENTER</code>	Indica un simple rectángulo y centra el texto suministrado en su interior. El texto será formateado antes de mostrarse. Las palabras que excedan el final de la línea se pasarán automáticamente al principio de la siguiente que será centrada también.
<code>SS_CENTERIMAGE</code>	<p>Indica el punto medio del control static con el estilo <code>SS_BITMAP</code> o <code>SS_ICON</code> que permanecerá fijo cuando el control sea redimensionado. Los cuatro lados serán ajustados para adaptarse al nuevo bitmap o icono.</p> <p>Si un control estatic tiene el estilo <code>SS_BITMAP</code> y el bitmap es más pequeño que el área de cliente del control, el área de cliente se rellenará con el color del pixel en la esquina superior izquierda del</p>

	<p>bitmap. Si un control static tiene el estilo SS_ICON, el icono no influye para pintar el área de cliente.</p>
SS_GRAYFRAME	<p>Indica una caja con un marco que se pintará del mismo color que el fondo de pantalla (escritorio). Este color es gris en el esquema de color por defecto en Windows.</p>
SS_GRAYRECT	<p>Indica un rectángulo relleno con el color que el fondo de pantalla. Este color es gris en el esquema de color por defecto en Windows.</p>
SS_ICON	<p>Indica que un icono será mostrado en el dialog box. El texto proporcionado será el nombre de un icono (no un nombre de fichero) definido en otro lugar en el fichero de recursos. Este estilo ignora los parámetros nWidth y nHeight; el icono se dimensiona automáticamente a si mismo.</p>
SS_LEFT	<p>Indica un simple rectángulo y una línea el texto suministrado en su interior a la izquierda. El texto será formateado antes de mostrarse. Las palabras que excedan el final de la línea se pasarán automáticamente al principio de la siguiente que será alineada a la izquierda también.</p>
SS_LEFTNOWORDWRAP	<p>Indica un simple rectángulo y alinea a la izquierda el texto suministrado en su interior. Los caracteres tab se expandirán, pero el texto no será formateado. Las palabras que excedan el final de la línea serán cortadas.</p>
SS_METAPICT	<p>Indica que una imagen metafile será mostrada en el control static. El texto proporcionado será el nombre de la imagen metafile (no un nombre de fichero) definido en otro lugar en el fichero de recursos. Un control metafile static tiene un tamaño fijo; la imagen metafile será escalada para adaptarse al área de cliente del control.</p>

SS_NOPREFIX	<p>Impide la interpretación del cualquier caracter ampersand (&) el en texto del control como carácter de prefijo de acelerador. Los aceleradores serán mostrados eliminando el carácter & y con el siguiente carácter subrayado. Este estilo puede ser incluido con cualquiera de los controles estáticos definidos.</p> <p>Una aplicación puede combinar el estilo SS_NOPREFIX con otros estilos usando el operador de bits OR (). Esto se usa normalmente con nombres de fichero u otras cadenas que puedan contener el carácter & que deba ser mostrado en un control estático de un cuadro de diálogo..</p>
SS_NOTIFY	<p>Envía los mensajes de notificación STN_CLICKED y STN_DBLCLK a la ventana padre cuando el usuario hace click o doble click en el control.</p>
SS_RIGHT	<p>Indica un simple rectángulo y alinea a la derecha el texto suministrado en su interior. El texto será formateado antes de mostrarse. Las palabras que excedan el final de la línea se pasarán automáticamente al principio de la siguiente que será alineada a la derecha también.</p>
SS_RIGHTIMAGE	<p>Indica que la esquina inferior derecha del control static con el estilo SS_BITMAP o SS_ICON permanecerá fijo cuando el control sea redimensionado. Sólo los lados superior e izquierdo serán ajustados para adaptarse al nuevo bitmap o icono.</p>
SS_SIMPLE	<p>Indica un rectángulo sencillo y muestra una línea de texto alineada a la izquierda del rectángulo. La línea de texto no podrá ser acortada o modificada de ninguna manera. La ventana padre del control o el cuadro de diálogo no debe procesar</p>

	el mensaje WM_CTLCOLORSTATIC.
SS_WHITEFRAME	Indica una caja con un marco que se pintará del mismo color que el fondo de la ventana. Este color es blanco en el esquema de color por defecto en Windows.
SS_WHITERECT	Indica un rectángulo relleno con el color que el fondo de la ventana. Este color es blanco en el esquema de color por defecto en Windows.

Window ex_styles:

Para el parámetro dwExStyle se pueden especificar los siguientes estilos de ventana en la función CreateWindowEx.

También pueden usarse en la definición de recursos.

Estilo	Significado
WS_EX_ACCEPTFILES	Especifica que una ventana creada con este estilo acepta el arrastre y pegado de ficheros.
WS_EX_CONTEXTHELP	Incluye un signo de interrogación en la barra de título de la ventana. Cuando el usuario hace clic sobre ella, el cursor cambia a un puntero con una interrogación. Si entonces el usuario hace clic sobre una ventana hija, la ventana hija recibe un mensaje WM_HELP. La ventana hija debe pasar el mensaje al procedimiento de ventana de su ventana padre, el cual debe llamar a la función WinHelp usando el comando HELP_WM_HELP. La aplicación de ayuda muestra una ventana pop-up que normalmente contiene ayuda sobre la ventana hija.
WS_EX_CONTROLPARENT	Permite al usuario navegar a través de las ventanas hija de la ventana usando la tecla TAB.
WS_EX_DLGMODALFRAME	Crea una ventana con un borde doble. A diferencia del estilo WS_DLGFRAME, una aplicación también puede especificar el estilo WS_CAPTION para crear una barra de título para la ventana.
WS_EX_NOPARENTNOTIFY	Especifica que una ventana hija creada con este estilo no pueda enviar el mensaje WM_PARENTNOTIFY a su ventana padre cuando se crea o se destruye.

WS_EX_TOPMOST	Especifica que una ventana creada con este estilo debe ponerse sobre todas las ventanas que no sean topmost y permancer sobre ellas hasta que la ventana sea desactivada.
WS_EX_TOOLWINDOW	Crea una ventana de herramientas, es decir, una ventana diseñada para ser usada como una barra flotante de herramientas. Una ventana de herramientas tiene una barra de título que es más paqueña que una barra normal, y el título se escribe con una fuente más pequeña. Una ventana de este tipo no aparece en la barra de tareas o en la ventana que se muestra si el usuario pulsa ALT+TAB.

Window styles:

Para el parámetro dwStyle se pueden especificar los siguientes estilos de ventana.

Estilo	Significado
WS_BORDER	Crea una ventana que tiene de borde una línea fina.
WS_CAPTION	Crea una ventana con una barra de título, (incluye el estilo WS_BORDER).
WS_CHILD	Crea una ventana hija. Este estilo no puede ser usado junto con el estilo WS_POPUP.
WS_CHILDWINDOW	Lo mismo que WS_CHILD.
WS_CLIPCHILDREN	Excluye el área ocupada por ventanas hija cuando se pinta dentro de la ventana padre. Este estilo se usa cuando se crea la ventana padre.
WS_CLIPSIBLINGS	Descarta las áreas relativas de cada una de las ventanas hijas restantes; esto es, cuando una ventana hija concreta recibe un mensaje WM_PAINT, el estilo WS_CLIPSIBLINGS excluye el área ocupada por todas las otras ventanas hijas superpuestas con la región de la ventana a actualizar. Si WS_CLIPSIBLINGS no se especifica y las ventanas hijas se superponen, es posible, cuando se dibuja en el área de cliente de la ventana hija, que se pinte dentro del área de cliente de la ventana hija colindante.
WS_DISABLED	Crea una ventana que inicialmente está deshabilitada. Una ventana deshabilitada no puede recibir datos del usuario.
WS_DLGFRAME	Crea una ventana con un estilo de borde típico de los cuadros de diálogo. Una ventana con este estilo no puede tener una barra de título.

WS_GROUP	Indica que es el primer control de un grupo de controles. El usuario puede cambiar el foco del teclado de un control de un grupo al siguiente en el mismo grupo usando las teclas de dirección. Todos los controles definidos sin el estilo WS_GROUP después del primer control de grupo pertenecerán al mismo grupo. El siguiente control con el estilo WS_GROUP termina el grupo y empieza el siguiente.
WS_HSCROLL	Crea una ventana que tiene una scroll bar horizontal.
WS_ICONIC	Crea una ventana inicialmente minimizada. El mismo efecto que el estilo WS_MINIMIZE.
WS_MAXIMIZE	Crea una ventana inicialmente maximizada.
WS_MAXIMIZEBOX	Crea una ventana que tiene un botón de Maximizar.
WS_MINIMIZE	Crea una ventana inicialmente minimizada. Lo mismo que el estilo WS_ICONIC.
WS_MINIMIZEBOX	Crea una ventana que tiene un botón de Minimizar.
WS_OVERLAPPED	Crea una ventana "superpuesta". Una ventana "superpuesta" (overlapped) tiene una barra de título y un borde. El mismo efecto que el estilo WS_TILED.
WS_OVERLAPPEDWINDOW	Crea una ventana superpuesta con los estilos: WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX y WS_MAXIMIZEBOX. El mismo efecto que el estilo WS_TILEDWINDOW.
WS_POPUP	Crea una ventana "pop-up". Este estilo no puede usarse junto con el estilo WS_CHILD.

WS_POPUPWINDOW	Crea una ventana "pop-up" con los estilos: WS_BORDER, WS_POPUP y WS_SYSMENU. Los estilos WS_CAPTION y WS_POPUPWINDOW deben combinarse para que el menú de sistema sea visible.
WS_SIZEBOX	Crea una ventana que tiene un borde que permite cambiar su tamaño. El mismo efecto que el estilo WS_THICKFRAME.
WS_SYSMENU	Crea una ventana que contiene un menú de sistema en su barra de título. El estilo WS_CAPTION debe ser especificado también.
WS_TABSTOP	Define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
WS_THICKFRAME	Crea una ventana que tiene un borde que permite cambiar su tamaño. El mismo efecto que el estilo WS_SIZEBOX.
WS_TILED	Crea una ventana "superpuesta". Una ventana "superpuesta" (overlapped) tiene una barra de título y un borde. El mismo efecto que el estilo WS_OVERLAPPED.
WS_TILEDWINDOW	Crea una ventana "superpuesta" con los estilos: WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX y WS_MAXIMIZEBOX. Lo mismo que el estilo WS_OVERLAPPEDWINDOW.
WS_VISIBLE	Crea una ventana inicialmente visible.
WS_VSCROLL	Crea una ventana con un scroll

	bar vertical.
--	---------------



Estructuras

Estructura CREATESTRUCT

Definición:

```
typedef struct tagCREATESTRUCT { // cs
    LPVOID lpCreateParams;
    HINSTANCE hInstance;
    HMENU hMenu;
    HWND hwndParent;
    int cy;
    int cx;
    int y;
    int x;
    LONG style;
    LPCTSTR lpszName;
    LPCTSTR lpszClass;
    DWORD dwExStyle;
} CREATESTRUCT;
```

La estructura CREATESTRUCT define los parámetros de inicialización pasados al procedimiento ventana de una aplicación.

Descripción:

lpCreateParams: puntero a datos a usar para crear una ventana.

Windows NT: este miembro es la dirección de un valor SHORT (16 bits) que especifica el tamaño en bytes de los datos para la creación de la ventana. El valor es seguido a continuación por los datos. Para mayor información, consulta la sección de observaciones.

hInstance: identifica al módulo al que pertenece la nueva ventana.

hMenu: identifica el menú que será usado por la nueva ventana.

hwndParent: identifica la ventana padre, si se trata de una ventana hija. Si la ventana tiene dueño, este miembro identifica a la ventana propietaria. Si no se trata de una ventana hija o con dueño, este miembro es NULL.

cy: especifica la anchura de la ventana en pixels.

cx: especifica la altura de la ventana en pixels.

y: indica la coordenada y de la esquina superior izquierda de la nueva ventana. Si la nueva ventana es una ventana hija, las coordenadas son relativas a la ventana padre. En caso contrario, las coordenadas son relativas al origen de la pantalla.

x: indica la cooredenada x de la esquina superior izquierda de la nueva ventana. Si la nueva ventana es una ventana hija, las coordenadas son relativas a la ventana padre. En caso contrario, las coordenadas son relaticas al origen de la pantalla.

style: especifica el estilo para la nueva ventana.

lpzName: apunta a una cadena (terminada con cero) que especifica el nombre de la nueva ventana.

lpzClass: apunta a una cadena (teminada con cero) que especifica el nombre de la clase de la nueva ventana.

dwExStyle: especifica el estilo ampliado o extendido de la nueva ventana.

Observaciones:

Windows NT: con referencia a l miembro lpCreateParams de la estructura CREATESTRUCT, ya que el puntero muede no estar alineado como DWORD, una aplicación debe acceder a los datos usando un puntero que haya sido declarado usando el tipo UNALIGNED, como se muestra en el siguiente ejemplo:

```
typedef struct tagMyData {  
    ...; // definir los datos de creación aquí  
} MYDATA;
```

```
typedef struct tagMyDlgData {  
    SHORT cbExtra;  
    MYDATA myData;  
} MYDLGDATA, UNALIGNED *PMYDLGDATA;
```

```
PMYDLGDATA pMyDlgdata =  
    (PMYDLGDATA) (((LPCREATESTRUCT) lParam)->lcreateParams);
```

Estructura MINMAXINFO

Definición:

```
typedef struct tagMINMAXINFO { // mmi  
  
    POINT ptReserved;  
    POINT ptMaxSize;  
    POINT ptMaxPosition;  
    POINT ptMinTrackSize;  
    POINT ptMaxTrackSize;  
} MINMAXINFO;
```

Esta estructura contiene la información sobre el tamaño y posición de la ventana maximizada y sobre el tamaño máximo y mínimo del tamaño "tracking".

Descripción:

ptReserved: reservado, no usar.

ptMaxSize: especifica la anchura máxima (point.x) y la altura máxima (point.y) de la ventana.

ptMaxPosition: especifica la posición del borde izquierdo de la ventana maximizada (point.x) y la posición del borde superior de la ventana maximizada (point.y).

ptMinTrackSize: indica la anchura del "tracking" mínimo (point.x) y la altura del "tracking" mínimo (point.y) de la ventana.

ptMaxTrackSize: indica la anchura del "tracking" máximo (point.x) y la altura del "tracking" máximo (point.y) de la ventana.

Estructura MSG

Definición:

```
typedef struct tagMSG { // msg
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

Esta estructura contiene la información sobre el mensaje recibido de la lista de mensajes de proceso.

Descripción:

hwnd: identificador de la ventana cuyo procedimiento de ventana recibirá el mensaje.

message: número de mensaje.

wParam: información adicional sobre el mensaje. El significado exacto depende del número de mensaje.

lParam: información adicional sobre el mensaje. El significado exacto depende del número de mensaje.

time: la hora a la que el mensaje fue enviado.

pt: posición del cursor, en coordenadas de pantalla, en el momento en que se envió en mensaje.

Estructura SCROLLINFO

Definición:

```
typedef struct tagSCROLLINFO { // si
    UINT cbSize;
    UINT fMask;
    int nMin;
    int nMax;
    UINT nPage;
    int nPos;
    int nTrackPos;
} SCROLLINFO;
typedef SCROLLINFO FAR *LPSCROLLINFO;
```

La estructura SCROLLINFO contiene los parámetros de scrollbars para ser modificados por la función [SetScrollInfo](#) (o el mensaje [SBM_SETSCROLLINFO](#)), o recuperados por la función [GetScrollInfo](#) (o el mensaje [SBM_GETSCROLLINFO](#)).

Descripción:

cbSize: especifica el tamaño en bytes de la estructura.

fMask: especifica qué parámetros de scrollbar serán modificados o recuperados. Este miembro puede ser una combinación de los siguientes valores:

Valor	Significado
SIF_ALL	Combinación de SIF_PAGE, SIF_POS y SIF_RANGE.
SIF_DISABLENOSCROLL	Este valor sólo se usa para se modifican parámetros de scrollbars. Si los nuevos parámetros hacen que el scrollbar sea innecesario, deshabilita el scrollbar en lugar de eliminarlo.
SIF_PAGE	El miembro nPage de la estructura SCROLLINFO contiene el tamaño de página para un scrollbar proporcional.
SIF_POS	El miembro nPos contiene la posición de la caja de scroll.
SIF_RANGE	Los miembros nMin y nMax contienen los valores mínimo y máximo del rango de desplazamiento.

nMin: especifica la posición mínima de desplazamiento.

nMax: especifica la posición máxima de desplazamiento.

nPage: especifica el tamaño de la página. Una scrollbar usa este valor para determinar el tamaño apropiado de una caja de desplazamiento (thumb) proporcional.

nPos: especifica la posición de la caja de desplazamiento.

nTrackPos: especifica la posición inmediata de una caja de desplazamiento que el usuario está desplazando. La aplicación puede recuperar este valor procesando el mensaje de notificación SB_THUMBTRACK. La aplicación no puede modificar este valor; la función SetScrollInfo ignora este miembro.

Estructura STARTUPINFO

Definición:

```
typedef struct _STARTUPINFO { // si
    DWORD   cb;
    LPTSTR  lpReserved;
    LPTSTR  lpDesktop;
    LPTSTR  lpTitle;
    DWORD   dwX;
    DWORD   dwY;
    DWORD   dwXSize;
    DWORD   dwYSize;
    DWORD   dwXCountChars;
    DWORD   dwYCountChars;
    DWORD   dwFillAttribute;
    DWORD   dwFlags;
    WORD    wShowWindow;
    WORD    cbReserved2;
    LPBYTE  lpReserved2;
    HANDLE  hStdInput;
    HANDLE  hStdOutput;
    HANDLE  hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

Esta estructura se usa junto con la función [CreateProcess](#) para definir las propiedades de la ventana principal si se crea una nueva ventana para un proceso nuevo. Para procesos con interfaz gráfico de usuario (GUI), esta información afecta a la primera ventana creada con la función [CreateWindow](#). Para procesos de consola, esta información afecta a la ventana de consola si el proceso crea una nueva ventana de consola. Un proceso puede usar la función [GetStartupInfo](#) para recuperar la estructura STARTUPINFO especificada cuando un proceso fue creado.

Descripción:

cb: especifica el tamaño en bytes de la estructura.

lpReserved: reservado. Poner este miembro a NULL antes de pasar la estructura a [CreateProcess](#).

lpDesktop:

Windows 95: ignora este miembro.

Windows NT: apunta a una cadena (terminada con cero) que especifica o bien el nombre del escritorio sólo o bien ambos: el nombre de la estación de ventanas y

del escritorio para éste proceso. Una barra inversa en la cadena indica que ésta incluye ambos nombres: escritorio y estación de ventanas. En otro caso, lpDesktop se interpreta como el nombre de escritorio. Si lpDesktop es NULL, el nuevo proceso heredará la ventana-estación y el escritorio del proceso padre.

lpTitle: para procesos de consola, éste es el título mostrado en la barra de título si se crea una nueva ventana de consola. Si es NULL, se usará en su lugar el nombre del fichero ejecutable. Este parámetro debe ser NULL para GUI o procesos de consola que no creen nuevas ventanas de consola.

dwX, dwY: se ignoran a no ser que dwFlags especifique STARTF_USEPOSITION. Indican un desplazamiento x e y, en pixels, de la esquina superior izquierda de una ventana si se crea una nueva ventana. Los desplazamientos son referentes a la esquina superior izquierda de la pantalla. Para procesos GUI, la posición especificada se usa la primera vez que el nuevo proceso llama a [CreateWindow](#) para crear una ventana superpuesta si el parámetro x de CreateWindow es CW_USEDEFAULT.

dwXSize, dwYSize: se ignoran a no ser que dwFlags especifique STARTF_USESIZE. Indican el ancho (dwXSize) y el alto (dwYSize), en pixels, de la ventana si se crea una nueva ventana. Para procesos GUI, son usados sólo la primera vez que el nuevo proceso llama a [CreateWindow](#) para crear una ventana superpuesta si el parámetro nWidth de CreateWindow es CW_USEDEFAULT.

dwXCountChars, dwYCountChars: se ignoran a no ser que dwFlags especifique STARTF_USECOUNTCHARS. Para procesos de consola, si se crea una nueva ventana de consola, dwXCountChars indica el ancho del buffer de pantalla en columnas de caracteres, y dwYCountChars indica la altura del buffer de pantalla en líneas de caracteres. Estos valores se ignoran en procesos GUI.

dwFillAttribute: se ignora a no ser que dwFlags especifique STARTF_USEFILLATTRIBUTE. Para procesos de consola, indica los colores iniciales de texto y fondo si se crea una nueva ventana de consola. Estos valores se ignoran en procesos GUI.

dwFlags: se trata de un campo de bits que determina si determinados miembros de la estructura STARTUPINFO serán usados cuando el proceso cree una ventana. Puede usarse cualquier combinación de los siguientes valores:

Valor	Significado
STARTF_USESHOWWINDOW	Si no es especifica este valor, el miembro wShowWindow member será ignorado.
STARTF_USEPOSITION	Si no se especifica este valor, los miembros dwX y dwY serán ignorados.
STARTF_USESIZE	Si no se especifica este valor, los miembros dwXSize y

	dwYSize serán ignorados.
STARTF_USECOUNTCHARS	Si no se especifica este valor, los miembros dwXCountChars y dwYCountChars serán ignorados.
STARTF_USEFILLATTRIBUTE	Si no se especifica este valor, el miembro dwFillAttribute será ignorado.
STARTF_FORCEONFEEDBACK	<p>Si se especifica este valor, el cursor permanecerá dos segundos en modo "feedback" después de que se llame a CreateProcess. Si durante esos dos segundos el proceso hace la primera llamada GUI, el sistema proporciona cinco segundos más al proceso. Si durante esos cinco segundos el proceso muestra una ventana, el sistema proporciona cinco segundos más al proceso para que termine de dibujar la ventana.</p> <p>El sistema apaga el cursor "feedback" después de la primera llamada a GetMessage, sin preocuparse de si el proceso está dibujando.</p> <p>Para más información sobre "feedback", ver la sección de observaciones del final.</p> <p>*feedback: retroalimentación.</p>
STARTF_FORCEOFFFEEDBACK	Si se especifica, se eliminará el cursor "feedback" durante el arranque del proceso. Se mostrará el cursor normal. Para más información sobre "feedback", ver la sección de observaciones del final.
STARTF_SCREENSAVER	<p>Si se especifica, el sistema tratará la aplicación como un salva pantallas:</p> <ul style="list-style-type: none"> • El sistema permite a la aplicación iniciarse en la

	<p>prioridad de primer plano de la clase de prioridad pasada a la función CreateProcess. Para más información sobre prioridades de procesos, ver Scheduling Priorities.</p> <ul style="list-style-type: none"> • Cuando la aplicación comprueba la entrada usando las funciones GetMessage o PeekMessage por primera vez, el sistema la cambia a la clase de prioridad desocupado (idle). Esto deja a las aplicaciones de fondo tener prioridad sobre ella. • Cuando el usuario mueve el ratón o presiona una tecla, el sistema da a la aplicación la prioridad de primer plano o la clase de prioridad pasada originalmente en la función CreateProcess. <p>Crear un salva pantallas con <code>NORMAL_PRIORITY_CLASS</code> le permite iniciarse y terminar por encima de aplicaciones con prioridad normal.</p>
<p>STARTF_USESTDHANDLES</p>	<p>Si se especifica este valor, se usan los manipuladores de entrada estándar, de salida y de error del proceso a los especificados en los miembros <code>hStdInput</code>, <code>hStdOutput</code> y <code>hStdError</code> de la estructura <code>STARTUPINFO</code>. El parámetro <code>flInheritHandles</code> de la función CreateProcess debe ser puesto a <code>TRUE</code> para que funcione</p>

	<p>correctamente.</p> <p>Si no se especifica, se ignoraran los miembros hStdInput, hStdOutput y hStdError de la estructura STARTUPINFO.</p>
--	---

wShowWindow: se ignora a no ser que dwFlags especifique STARTF_USESHOWWINDOW. Para procesos GUI, wShowWindow indica el valor por defecto la primera vez que se llama a [ShowWindow](#), si el parámetro nCmdShow de la función ShowWindow es SW_SHOWDEFAULT. El miembro wshowWindow puede tener como valor cualquier constante SW_ definida en WINUSER.H. En versiones anteriores de Windows, la función [WinMain](#) tenía un parámetro nCmdShow que se recomendaba que la aplicación pasara a las funciones [CreateWindow](#) o [ShowWindow](#). El parámetro nCmdShow ya no está disponible a través de WinMain. Ahora, las aplicaciones deben usar SW_SHOWDEFAULT como el comando de por defecto en ShowWindow, en éste caso, ShowWindow usará el miembro wShowWindow de la estructura STARTUPINFO del proceso.

cbReserved2: reservado; debe ser cero.

lpReserved2: reservado; debe ser NULL.

hStdInput: se ignora a no ser que dwFlags especifique STARTF_USESTDHANDLES. Indica un manipulador que se usará como manipulador de entrada estándar del proceso si se especifica STARTF_USESTDHANDLES.

hStdOutput: se ignora a no ser que dwFlags especifique STARTF_USESTDHANDLES. Indica un manipulador que se usará como manipulador de salida estándar del proceso si se especifica STARTF_USESTDHANDLES.

hStdError: se ignora a no ser que dwFlags especifique STARTF_USESTDHANDLES. Indica un manipulador que se usará como manipulador de error estándar del proceso si se especifica STARTF_USESTDHANDLES.

Observaciones:

Si un proceso GUI está comenzado y ni STARTF_FORCEONFEEDBACK ni STARTF_FORCEOFFFEEDBACK fueron especificados, se usará el proceso de cursor "feedback". Un proceso GUI es uno cuyo subsistema es especificado como "windows."

Estructura WNDCLASS

Definición:

```
typedef struct _WNDCLASS { // wc
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
} WNDCLASS;
```

Descripción:

style: Es un entero de 16 bits que codifica el estilo de la clase de ventana. Estos bits se pueden combinar con la función OR (`|`), para obtener estilos con las características deseadas.

Estos valores pueden ser:

Valor	Significado
CS_BYTEALIGNCLIENT	Hace que el área de cliente coincida con el límite de un byte en la dirección de las x. Esto mejora las prestaciones a la hora de pintar en la pantalla. Este estilo afecta tanto al ancho de la ventana como a su posición en la pantalla.
CS_BYTEALIGNWINDOW	Lo mismo que el anterior, pero con el borde de la ventana, en lugar del área de cliente.
CS_CLASSDC	Crea un DC (Device Context) que será compartido por todas las ventanas de la misma clase. Cuando varias ventanas intenten acceder simultáneamente al DC, el sistema operativo permite sólo a una el acceso hasta que termina.
CS_DBLCLKS	Envía los mensaje de doble-clic al procedimiento de la ventana.

	cuando el usuario hace doble-clic sobre una ventana de esta clase.
CS_GLOBALCLASS	Permite a una aplicación crear una ventana de esta clase independientemente del valor de hInstance que se proporcione a CreateWindow. Si no se especifica, el valor de hInstance debe ser el mismo que se uso para registrar la clase.
CS_HREDRAW	Redibuja toda la ventana cada vez que un movimiento o cambio de tamaño cambia la anchura del área de cliente.
CS_NOCLOSE	Deshabilita del comando de cerrar del menú del sistema.
CS_OWNDC	Crea un DC único para cada ventana de esta clase.
CS_PARENTDC	Hace que cada ventana hija herede el DC de su padre.
CS_SAVEBITS	Guarda como mapas de bits los trozos de pantalla tapados por la ventana. Windows usará estos mapas de bits para reconstruir la pantalla cuando la ventana se cierre. Windows muestra los mapas en su lugar original y no envía el mensaje de actualizar la ventana a las ventanas afectadas. Este estilo se usa normalmente con ventanas pequeñas como menús o diálogos que se muestran y son cerrados antes de que haya alguna actividad en la pantalla. Este estilo aumenta el tiempo necesario para mostrar la ventana, ya que Windows debe reservar memoria para guardar el mapa de bits.
CS_VREDRAW	Redibuja toda la ventana cada vez que un movimiento o cambio de tamaño cambia la altura del área de cliente.

lpfnWndProc: Apunta al procedimiento de ventana. Esto quedará más claro cuando veamos los procedimientos de ventana.

cbClsExtra: Especifica cuantos bytes extra se reservarán a continuación de la estructura de la clase. El sistema operativo inicializará estos bytes a cero.

cbWndExtra: Especifica cuantos bytes extra se reservarán a continuación de la instancia de la ventana. El sistema operativo inicializará estos bytes a cero.

hInstance: Identifica la instancia de la ventana a la que esta clase pertenece.

hIcon: Identificador del icono de la clase. Debe ser un manipulador de un recurso de tipo icono. Si es NULL, la aplicación debe mostrar un icono cuando el usuario minimice la ventana de la aplicación.

hCursor: Identificador del cursor de la clase. Debe ser un manipulador de recurso de tipo cursor. Si es NULL, la aplicación debe mostrar el cursor explícitamente cada vez que el usuario mueve el ratón sobre la ventana de la aplicación.

hbrBackground: Identificador del pincel para la clase. Puede ser un manipulador para un pincel físico que se usará para pintar el fondo de la ventana, o puede ser un valor de color.

El sistema operativo elimina el pincel de fondo de la clase cuando ésta se libera. Una aplicación nunca debe borrar estos pinceles, ya que la misma clase de ventana puede ser usada por varias instancias de la misma aplicación. Si este miembro es NULL, la aplicación es la encargada de pintar el fondo del área de cliente cada vez que el sistema los requiera. Para determinar si el fondo debe ser actualizado, la aplicación puede procesar el mensaje WM_ERASEBKGD o comprobar el miembro fErase de la estructura PAINTSTRUCT que se actualiza con la función BeginPaint.

lpszMenuName: Puntero a una cadena terminada con cero que especifica un nombre de recurso de la clase menú, es el nombre con el que aparece en el fichero de recursos. Si se usa un entero para identificar el menú, hay que usar la macro MAKEINTRESOURCE. Si este miembro es NULL, las ventanas de esta clase no tendrán un menú por defecto.

lpszClassName: Apunta a una cadena o es un átomo. Si es un átomo, este debe estar previamente creado como global con una llamada a la función GlobalAddAtom. El átomo, un valor de 16 bits, debe ser la palabra de menor peso del parámetro lpszClassName; la palabra de mayor peso debe ser cero. Si lpszClassName es una cadena, especificará el nombre de la clase de ventana.



Funciones

AppendMenu:

La función AppendMenu añade un nuevo ítem de menú al final del menú especificado. Una aplicación puede usar esta función para especificar el contenido, apariencia y comportamiento del ítem de menú.

Sintaxis:

```
BOOL AppendMenu(  
    HMENU hMenu,      // manipulador de menú  
    UINT uFlags,     // flags del ítem de menú  
    UINT uIDNewItem, // identificador de ítem de menú o manipulador de menú  
    pop-up  
    LPCTSTR lpNewItem // contenido del ítem de menú  
);
```

Parámetros:

hMenu: identifica el menú a modificar.

uFlags: especifica los flags para controlar la apariencia y el comportamiento del ítem de menú. Este parámetro puede ser una combinación de los valores listados en la sección de observaciones.

uIDNewItem: especifica o bien el identificador del nuevo ítem de menú o si el parámetro uFlags incluye MF_POPUP, el manipulador de un menú pop-up.

lpNewItem: especifica el contenido del nuevo ítem de menú. La interpretación de lpNewItem depende de si el parámetro uFlags incluye los flags MF_BITMAP, MF_OWNERDRAW o MF_STRING, como se especifica a continuación:

Valor	Descripción
MF_BITMAP	Contiene un manipulador de bitmap.
MF_OWNERDRAW	Contiene un valor de 32 bits suministrado por la aplicación que puede ser usado para mantener datos adicionales relacionados con el ítem de menú. El valor está en el miembro itemData member de la estructura apuntada por el parámetro lparam del mensaje WM_MEASUREITEM o WM_DRAWITEM enviado cuando el menú es creado o su apariencia es actualizada.
MF_STRING	Contiene un puntero a una cadena de

	caracteres terminada con cero.
--	--------------------------------

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

La aplicación debe llamar a la función [DrawMenuBar](#) cada vez que el menú cambie, tanto si el menú pertenece a una ventana que se está mostrando como si no.

Los siguientes flags deben ser usados en el parámetro uFlags:

Valor	Descripción
MF_BITMAP	Usa un bitmap como ítem. El parámetro lpNewItem contiene el manipulador del bitmap.
MF_CHECKED	Pone una marca de comprobación junto al ítem. Si la aplicación proporciona los bitmaps de las marcas de comprobación (ver SetMenuItemBitmaps), éste flag muestra la marca junto al ítem de menú.
MF_DISABLED	Deshabilita el ítem de menú de modo que no puede ser seleccionado, pero éste flag no pone el ítem en gris.
MF_ENABLED	Habilita en ítem de menú de modo que pueda ser seleccionado y restaura su apariencia si su estado estaba en gris.
MF_GRAYED	Deshabilita el ítem y lo muestra en gris de modo que no puede ser seleccionado.
MF_MENUBARBREAK	Funciona igual que el flag MF_MENUBREAK excepto para menús pop-up, donde la nueva columna es separada de la anterior por una línea vertical.
MF_MENUBREAK	Coloca el ítem en una línea nueva (para barras de menú) o en una columna nueva (para menús pop-up)

	sin columnas separadas.
MF_OWNERDRAW	<p>Especifica que el ítem es dibujado por la ventana propietaria, (owner-drawn). Antes de que el menú sea mostrado por primera vez, la ventana que posee el menú recibirá un mensaje WM_MEASUREITEM para recuperar la anchura y altura del ítem de menú. El mensaje WM_DRAWITEM es enviado al procedimiento de ventana de la ventana propietaria cada vez que la apariencia del ítem deba ser actualizada.</p>
MF_POPUP	<p>Indica que el ítem es un ítem pop-up; es decir, al seleccionarlo se activa un menú pop-up. El parámetro <code>uIDNewItem</code> especifica el manipulador del menú pop-up. El flag es usado para añadir un ítem pop-up a una barra de menú o a un menú pop-up.</p>
MF_SEPARATOR	<p>Dibuja una línea horizontal de división. Este flag se usa sólo en menús pop-up. La línea no puede ser puesta en gris, deshabilitada o resaltada. Los parámetros <code>lpNewItem</code> y <code>uIDNewItem</code> se ignoran.</p>
MF_STRING	<p>Especifica que el ítem es una cadena de texto; el parámetro <code>lpNewItem</code> apunta a una cadena.</p>
MF_UNCHECKED	<p>No pone la marca de comprobación junto al ítem (valor por defecto). Si la aplicación suministra bitmaps para las marcas de comprobación (ver SetMenuItemBitmaps), este flag muestra el bitmap de no comprobación junto al ítem.</p>

La lista siguiente muestra grupos de flags que no pueden ser usados juntos:

- MF_DISABLED, MF_ENABLED y MF_GRAYED
- MF_BITMAP, MF_STRING y MF_OWNERDRAW
- MF_MENUBARBREAK y MF_MENUBREAK
- MF_CHECKED y MF_UNCHECKED

CreateMenu:

La función CreateMenu crea un menú. El menú estará inicialmente vacío, pero puede ser rellenado con ítems usando las funciones [AppendMenu](#) e [InsertMenu](#).

Sintaxis:

```
HMENU CreateMenu(VOID)
```

Parámetros:

Esta función no tiene parámetros.

Valor de retorno:

Si la función tiene éxito el valor de retorno será el manipulador del nuevo menú creado.

Si la función falla, el valor de retorno será NULL.

Observaciones:

Los recursos asociados con el menú que es asignado a una ventana son liberados automáticamente cuando la ventana es destruida. Si el menú no está asignado a una ventana, la aplicación debe liberar los recursos del sistema asociados con el menú antes de cerrar. Una aplicación libera los recursos de menú llamando a la función [DestroyMenu](#).

Sólo en Windows 95: El sistema puede soportar un máximo de 16,364 manipuladores de menú.

CreateWindow:

Esta función crea ventanas superponibles (ventanas normales: overlapped), ventanas pop-up y ventanas hijas. Especifica la clase, título, estilo y opcionalmente la posición y tamaño de la ventana. También especifica la ventana padre o propietaria, si existe y el menú de la ventana.

Sintaxis:

```
HWND CreateWindow(  
    LPCTSTR lpClassName, // puntero al nombre de la clase registrada  
    LPCTSTR lpWindowName, // puntero al nombre de la ventana  
    DWORD dwStyle, // estilo de ventana  
    int x, // posición horizontal de la ventana  
    int y, // posición vertical de la ventana  
    int nWidth, // anchura de la ventana  
    int nHeight, // altura de la ventana  
    HWND hWndParent, // manipulador de la ventana padre o propietaria  
    HMENU hMenu, // manipulador de menú o identificador de ventana hija  
    HANDLE hInstance, // manipulador de la instancia de la aplicación  
    LPVOID lpParam // puntero a los datos para la creación de la ventana  
);
```

Parámetros:

lpClassName: puntero a una cadena terminada con cero o un átomo entero. Si el parámetro es un átomo, debe ser un átomo global creado previamente con una llamada a la función [GlobalAddAtom](#). El átomo, un valor de 16 bits menor de 0xC000, debe estar en la palabra de menor orden de lpClassName; la palabra de mayor orden debe ser cero.

Si lpClassName es una cadena, debe especificar el nombre de una clase de ventana. Este debe ser un nombre previamente registrado con la función [RegisterClass](#) o uno de los nombres de controles predefinidos. Para ver una lista completa, mira la siguiente sección de observaciones.

lpWindowName: puntero a una cadena terminada con NULL que especifica el nombre de la ventana.

dwStyle: especifica el estilo de la ventana que se creará. Este parámetro debe ser una combinación de los estilos de ventana y de los estilos de controles que se listan en el apartado de observaciones.

x: especifica la posición inicial horizontal de la ventana. Para una ventana "superpuesta" o una pop-up, este parámetro es la coordenada x inicial de la esquina superior izquierda de la ventana en coordenadas de pantalla. Para ventanas hijas, x

es la coordenada x de la esquina superior izquierda de la ventana relativa a la esquina superior izquierda del área de cliente de la ventana padre.

Si se usa el valor `CW_USEDEFAULT` para éste parámetro, Windows seleccionará la posición por defecto para la ventana e ignorará el parámetro y.

`CW_USEDEFAULT` sólo es válido para ventanas "superpuestas"; si se especifica para una ventana pop-up, tanto el parámetro x como el y serán puestos a cero.

y: especifica la posición inicial vertical de la ventana. Para una ventana "superpuesta" o una pop-up, este parámetro es la coordenada y inicial de la esquina superior izquierda de la ventana en coordenadas de pantalla. Para ventanas hija, y es la coordenada y de la esquina superior izquierda de la ventana relativa a la esquina superior izquierda del área de cliente de la ventana padre. Para un list-box, y es la coordenada y inicial de la esquina superior izquierda del área de cliente del list-box relativa al área de cliente de la ventana padre.

Si una ventana "superpuesta" se crea con el bit de estilo `WS_VISIBLE` y para el parámetro x se especifica `CW_USEDEFAULT`, Windows ignora el parámetro y.

nWidth: especifica el ancho de la ventana, en unidades de dispositivo. Para ventanas "superpuestas", nWidth puede ser el ancho de la ventana en coordenadas de pantalla, o `CW_USEDEFAULT`. Si es `CW_USEDEFAULT`, Windows elige un ancho y alto por defecto para la ventana; el valor de anchura por defecto es desde la coordenada x inicial hasta el borde derecho de la pantalla, y la altura por defecto desde la coordenada y hasta la parte superior del área de iconos.

`CW_USEDEFAULT` sólo es válido para ventanas "superpuestas"; si se especifica `CW_USEDEFAULT` para una ventana pop-up o una ventana hija tanto nWidth como nHeight serán cero.

nHeight: especifica la altura de la ventana en unidades de dispositivo. Para ventanas "superpuestas", nHeight es la altura de la ventana en coordenadas de pantalla. Si se especifica `CW_USEDEFAULT` para nWidth, Windows ignorará nHeight.

hWndParent: identifica la ventana padre o propietaria de la ventana creada. Debe indicarse un manipulador de ventana válido cuando se crea una ventana hija o con dueño. Una ventana hija estará confinada al área de cliente de su ventana padre. Una ventana con dueño es una ventana "superpuesta" que será destruida cuando su ventana padre sea destruida u ocultada cuando su ventana padre sea minimizada; además siempre se mostrará encima de su ventana padre. A pesar de que este parámetro debe ser un manipulador válido si el parámetro `dwStyle` incluye el estilo `WS_CHILD`, es opcional si `dwStyle` incluye el estilo `WS_POPUP`.

hMenu: identifica un menú o el identificador de una ventana hija, dependiendo del estilo de ventana. Para una ventana "superpuesta" o una ventana pop-up, hMenu identifica el menú que se usará con la ventana; puede ser `NULL` si se usa el menú de la clase. Para una ventana hija, hMenu especifica el identificador de la ventana hija, un valor entero que se usa por un control de cuadro de diálogo para informar a su ventana padre sobre eventos. Es la aplicación la que determina el identificador

de la ventana hija; debe ser único para todas las ventanas hija con la misma ventana padre.

hInstance: identifica la instancia del módulo asociado con la ventana.

lpParam: puntero a un valor pasado a la ventana a través de la estructura [CREATESTRUCT](#) referenciada por el parámetro lpParam del mensaje WM_CREATE. Si una aplicación llama a CreateWindow para crear una ventana de cliente de una aplicación MDI (multiple document interface), lpParam debe apuntar a una estructura [CLIENTCREATESTRUCT](#).

Valor de retorno:

Si al función tiene éxito, el valor de retorno es el manipulador de la nueva ventana.

Si la función falla, el valor de retorno es NULL. Para conseguir información adicional hay que llamar a la función [GetLastError](#).

Observaciones:

Antes de volver, CreateWindow envía un mensaje [WM_CREATE](#) al procedimiento de ventana.

Para ventanas "superpuestas", pop-up y ventanas hija, CreateWindow envía los mensajes WM_CREATE, [WM_GETMINMAXINFO](#), y [WM_NCCREATE](#) a la ventana. El parámetro lpParam del mensaje WM_CREATE contiene un puntero a la estructura [CREATESTRUCT](#). Si se especifica el estilo WS_VISIBLE, CreateWindow envía a la ventana todos los mensjaes necesarios para activar y mostrar la ventana.

Si el estilo de ventana especifica una barra de título, la cadena con el título de la ventana, apuntada por lpWindowName se muestra en la barra de título. Cuando se usa CreateWindow para crear controles, como buttons, checkboxes, y controles static, lpWindowName se usa para especificar el texto del control.

Las siguientes clases de controles predefinidos pueden ser usados en el parámetro lpClassName:

Clase	Significado
BUTTON	Designa una pequeña ventana rectangular que representa un botón que el usuario puede pulsar para encenderlo o apagarlo. Estos controles pueden ser usados solos o en grupos, también pueden tener etiquetas o aparecer sin texto. Estos controles normalmente cambian de aspecto cuando el usuario los pulsa.
COMBOBOX	Designa un control que consiste en un list box y

	<p>un campo de selección similar a un edit. Cuando usa este estilo, una aplicación debe o bien mostrar el list box todo el tiempo o bien habilitar un listbox desplegable.</p> <p>Dependiendo del estilo del combo box, el usuario podrá o no editar el contenido del campo de selección. Si el list box es visible, tecleando caracteres en el campo de selección, se seleccionará la primera entrada del list box que coincida con los caracteres escritos. Y a la inversa, seleccionando un ítem de la list box se muestra el texto seleccionado en el campo de selección.</p>
EDIT	<p>Se trata de una pequeña ventana hija dentro de la cual el usuario puede escribir texto desde el teclado. El usuario puede seleccionar el control que recibe el foco del teclado haciendo click sobre él, o moviéndose a él usando la tecla TAB. El usuario puede escribir texto cuando el control edit muestra un cursor (caret) intermitente; usar el ratón para mover el cursor, seleccionar caracteres para reemplazarlos, o situar el cursor para insertar caracteres o borrarlos usando la tecla de retroceso.</p> <p>Los controles Edit usan una fuente de sistema de anchura variable y muestran caracteres del conjunto ANSI. Se puede usar el mensaje WM_SETFONT, enviándolo al control edit para cambiar la fuente por defecto.</p> <p>Los controles edit expanden los caracteres tab en tantos espacios como se precisen para mover el cursor (caret) a la siguiente columna de tabulación (tab stop). Se asume que hay tab stops cada ocho caracteres.</p>
LISTBOX	<p>Se trata de listas de cadenas de caracteres. Se usa cada vez que la aplicación debe presentar una lista de nombres, como nombres de fichero, entre los cuales el usuario puede elegir. El usuario puede elegir una cadena haciendo clic sobre ella. Una cadena seleccionada se realzará y se pasa un mensaje de notificación a la ventana padre. Se usa una barra de scroll vertical u horizontal con list boxes para desplazar aquellas listas demasiado grandes para la ventana de control. El list box oculta o muestra automáticamente los scroll bars cuando sea necesario.</p>

MDICLIENT	Se trata de ventanas de cliente MDI. Esta ventana recibirá mensajes para el control de las ventanas hija de una aplicación MDI. Los bits de estilo recomendados son WS_CLIPCHILDREN y WS_CHILD. Se pueden especificar los estilos WS_HSCROLL y WS_VSCROLL para crear una ventana de cliente MDI que permita al usuario desplazar las ventanas hija MDI dentro de ella.
SCROLLBAR	<p>Se trata de un rectángulo que contiene un cuadro de scroll y dos flechas de dirección en sus extremos. Las scroll bars envían un mensaje de notificación a su ventana padre cada vez que el usuario hace clic en el control. La ventana padre es la responsable de actualizar la posición del cuadro de scroll, si es necesario. Los controles scroll bar tienen la misma apariencia y función que los usados en las ventanas corrientes. La diferencia es que los controles scroll bar pueden situarse en cualquier parte de la ventana para ser usados para introducir cualquier tipo de información que requiera la aplicación.</p> <p>La clase scroll bar también incluye los controles size box. Un size box es un pequeño rectángulo que el usuario puede agrandar para cambiar el tamaño de una ventana.</p>
STATIC	Se trata de un simple campo de texto, una caja o un rectángulo usado para etiquetar, enmarcar o separar otros controles. Los controles static no toman entradas ni proporcionan salidas.

Ver estilos de ventanas para el parámetro dwStyle ([window styles](#)).

Ver estilos de botones ([button styles](#)).

Ver estilos de combo boxes ([combobox styles](#)).

Ver estilos de controles edit ([control edit styles](#)).

Ver estilos de controles list boxes ([list box styles](#)).

Ver estilos de scroll bars ([scroll bar styles](#)).

Ver estilos de controles static ([static styles](#)).

Ver estilos de dialog boxes ([dialog box styles](#)).

Sólo en Windows 95: el sistema puede soportar un máximo de 16,364 manipuladores de ventana.

DefWindowProc:

Esta función llama al procedimiento de ventana por defecto que proporciona un mecanismo para procesar aquellos mensajes de ventana que a la aplicación no le interesa procesar. Esta función asegura que todos los mensajes serán procesados. Debe llamarse a DefWindowProc con los mismos parámetros suministrados al procedimiento de ventana.

Sintaxis:

```
LRESULT DefWindowProc(  
    HWND hWnd, // manipulador de la ventana  
    UINT Msg, // identificador de mensaje  
    WPARAM wParam, // primer parámetro del mensaje  
    LPARAM lParam // segundo parámetro del mensaje  
);
```

Parámetros:

hWnd: es el manipulador de la ventana a la que está destinado el mensaje.

msg: es el código del mensaje.

wParam: es el parámetro de tipo palabra asociado al mensaje, su valor depende del tipo de mensaje.

lParam: es el parámetro de tipo doble palabra asociado al mensaje, su valor depende del tipo de mensaje.

Valor de retorno:

El valor de retorno es el resultado del proceso del mensaje, y depende del tipo de mensaje. The return value is the result of the message processing and depends on the message.

DestroyMenu:

La función DestroyMenu destruye el menú indicado y libera la memoria que ocupaba.

Sintaxis:

```
BOOL DestroyMenu(  
    HMENU hMenu    // manipulador del menú a destruir  
);
```

Parámetros:

hMenu: identifica el menú a destruir.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

Antes de cerrar, una aplicación debe usar ésta función para destruir un menú que no esté asignado a una ventana. Un menú que esté asignado a una ventana es automáticamente destruido cuando la aplicación se cierra.

DialogBox:

La función DialogBox crea un cuadro de diálogo modal a partir de un recurso de plantilla de cuadro de diálogo. DialogBox no devuelve el control hasta que la función callback especificada termine el cuadro de diálogo modal con la llamada a la función EndDialog.

Sintaxis:

```
int DialogBox(  
    HANDLE hInstance,           // manipulador a la instancia de la aplicación  
    LPCTSTR lpTemplate, // identifica el recurso de cuadro de diálogo  
    HWND hWndParent,          // manipulador a la ventana propietaria  
    DLGPROC lpDialogFunc      // dirección del procedimiento de diálogo  
);
```

Parámetros:

hInstance: identifica una instancia del módulo cuyo fichero ejecutable contiene la plantilla del cuadro de diálogo.

lpTemplate: identifica la plantilla del cuadro de diálogo. Este parámetro puede ser la dirección de una cadena de caracteres terminada en cero que especifique el nombre de la plantilla o un valor entero que especifique el identificador de recurso de la plantilla. Si el parámetro se refiere a un identificador de recurso, su palabra de mayor peso debe ser cero y la de menor peso contendrá el identificador. Se puede usar la macro [MAKEINTRESOURCE](#) para crear este valor.

hWndParent: identifica la ventana a la que pertenece el cuadro de diálogo.

lpDialogFunc: apunta al procedimiento de diálogo. Para más información ver la función callback [DialogProc](#).

Valor de retorno:

Si la función tiene éxito, el valor de retorno será el parámetro nResult en la llamada a la función [EndDialog](#) usada para terminar el diálogo.

Si la función fracasa, el valor de retorno es -1.

Observaciones:

La función [DialogBoxIndirect](#) usa la función [CreateWindowEx](#) para crear el cuadro de diálogo. Entonces DialogBox envía un mensaje [WM_INITDIALOG](#) (y un mensaje [WM_SETFONT](#) si la plantilla especifica el estilo DS_SETFONT) al

procedimiento de diálogo. La función muestra el cuadro de diálogo (independientemente de si la plantilla especifica el estilo WS_VISIBLE), desactiva la ventana propietaria y comienza su propio bucle de mensajes para recuperar y despachar los mensajes del cuadro de diálogo.

Cuando el procedimiento de diálogo llama a la función EndDialog, DialogBox destruye el cuadro de diálogo, termina el bucle de mensajes, activa la ventana padre (si antes estaba activa) y devuelve el parámetro nResult suministrado por el procedimiento de diálogo cuando llamó a EndDialog.

Sólo en Windows 95: el sistema puede soportar un máximo de 16364 manipuladores de ventanas.

DialogProc:

DialogProc es una función callback definida en una aplicación que procesa los mensajes enviados a un cuadro de diálogo modal o no modal.

Sintaxis:

```
BOOL CALLBACK DialogProc(  
    HWND hwndDlg,    // Manipulador de cuadro de diálogo  
    UINT uMsg,       // Mensaje  
    WPARAM wParam,  // Parámetro palabra, varía  
    LPARAM lParam   // Parámetro doble palabra, varía  
);
```

Parámetros:

hwndDlg: es el manipulador de la ventana que identifica el cuadro de diálogo.

uMsg: es el código del mensaje.

wParam: es el parámetro de tipo palabra asociado al mensaje.

lParam: es el parámetro de tipo doble palabra asociado al mensaje.

Valor de retorno:

Excepto en la respuesta al mensaje [WM_INITDIALOG](#), el procedimiento de diálogo debe retornar con un valor no nulo si procesa el mensaje y cero si no lo hace. Cuando responde a un mensaje [WM_INITDIALOG](#), el procedimiento debe retornar cero si llama a la función [SetFocus](#) para poner el foco a uno de los controles del cuadro de diálogo. En otro caso, debe retornar un valor distinto de cero, en ese caso el sistema pondrá el foco en el primer control del diálogo que pueda recibirlo.

Observaciones:

Se debe usar el procedimiento de diálogo sólo si se usa una clase de diálogo para el cuadro de diálogo. Esta es la clase por defecto y se usa cuando no se toma una clase explícitamente en la plantilla del cuadro de diálogo. A pesar de que el procedimiento de diálogo es similar al procedimiento de ventana, nunca debe llamar a la función [DefWindowProc](#) para procesar mensajes no deseados. Estos mensajes son procesados internamente por el procedimiento de ventana del cuadro de diálogo.

DialogProc es una especie de comodín, un término que se debe sustituir por un nombre de función en la definición de una aplicación.

DispatchMessage:

Esta función envía un mensaje al procedimiento de ventana. Normalmente se usa para enviar los mensajes recogidos mediante la función [GetMessage](#).

Sintaxis:

```
LONG DispatchMessage(  
    CONST MSG *lpmsg // puntero a una estructura con el mensaje  
);
```

Parámetros:

lpmsg: puntero a una estructura [MSG](#) que contiene el mensaje.

Valor de retorno:

Especifica el valor retornado por el procedimiento de ventana. Su significado depende del tipo de mensaje, aunque este valor suele ser ignorado.

Observaciones:

La estructura MSG debe contener valores de mensaje válidos. Si el parámetro lpmsg apunta a un mensaje de tipo [WM_TIMER](#) y el parámetro lParam del mensaje WM_TIMER no es NULL, entonces lParam apunta a la función que será llamada en lugar del procedimiento de ventana.

DrawMenuBar:

La función DrawMenuBar redibuja la barra de menú de la ventana especificada. Si la barra de menú cambia después de que Windows ha creado la ventana, ésta función debe ser llamada para dibujar la barra de menú modificada.

Sintaxis:

```
BOOL DrawMenuBar(  
    HWND hWnd // manipulador de ventana con la barra de menú a actualizar  
);
```

Parámetros:

hWnd: identifica a la ventana cuya barra de menú necesita repintarse.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

EndDialog:

La función EndDialog destruye un cuadro de diálogo modal, hace que el sistema finalice cualquier procesamiento para el cuadro de diálogo.

Sintaxis:

```
BOOL EndDialog(  
    HWND hDlg,           // manipulador del cuadro de diálogo  
    int nResult          // valor de retorno  
);
```

Parámetros:

hDlg: identifica el cuadro de diálogo a destruir.

nResult: especifica el valor que se devolverá a la aplicación desde la función que creó el cuadro de diálogo.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE.

Observaciones:

Los cuadros de diálogo creados por las funciones [DialogBox](#), [DialogBoxParam](#), [DialogBoxIndirect](#) y [DialogBoxIndirectParam](#) deben ser destruidos con la función EndDialog. Una aplicación debe llamar a EndDialog desde el interior del procedimiento de diálogo; la función no debe ser usada con ningún otro propósito.

Un procedimiento de diálogo puede llamar a EndDialog en cualquier momento, incluso durante el procesamiento del mensaje [WM_INITDIALOG](#). Si tu aplicación llama a la función mientras WM_INITDIALOG está siendo procesado, el cuadro de diálogo es destruido antes de ser mostrado y antes de que el foco de entrada sea asignado.

EndDialog no destruye el diálogo inmediatamente. En vez de eso, activa un flag y permite al procedimiento de diálogo devolver el control al sistema. El sistema verifica ese flag antes de intentar recuperar el siguiente mensaje de la cola de la aplicación. Si el flag está activo, el sistema termina el bucle de mensajes, destruye el cuadro de diálogo y usa el valor en nResult como valor de retorno de la función que creó el cuadro de diálogo.

GetDlgItem:

la función GetDlgItem devuelve el manipulador de un control en el cuadro de diálogo especificado.

Sintaxis:

```
HWND GetDlgItem(  
    HWND hDlg,           // manipulador del cuadro de diálogo  
    int nIDDlgItem // identificador del control  
);
```

Parámetros:

hDlg: identifica el cuadro de diálogo que contiene el control.

nIDDlgItem: especifica el identificador del control del que se quiere recuperar el manipulador.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es el manipulador de la ventana del control.

Si la función falla el valor de retorno en NULL, que indica que el manipulador de diálogo no es válido o que el control no existe.

Observaciones:

Se puede usar GetDlgItem con cualquiera pareja de ventanas padre-hija, no sólo con cuadro de diálogo. Siempre que el parámetro hDlg especifique una ventana padre y la ventana hija tenga un identificador único (tal como se especifica en el parámetro hMenu de la función [CreateWindow](#) o [CreateWindowEx](#) de la ventana hija creada), GetDlgItem devolverá un manipulador válido a la ventana hija.

GetDlgItemInt:

La función GetDlgItemInt recupera el texto asociado con un control en un cuadro de diálogo y lo convierte a un valor entero.

Sintaxis:

```
UINT GetDlgItemInt(  
    HWND hDlg,           // manipulador al cuadro de diálogo  
    int nIDDlgItem, // identificador del control  
    BOOL *lpTranslated, // apunta a una variable que recibirá el indicador de  
    éxito/fracaso  
    int bSigned         // especifica si el valor es con o sin signo  
);
```

Parámetros:

hDlg: identifica al cuadro de diálogo que contienen el control.

nIDDlgItem: especifica el identificador del control del que se recuperará el número.

lpTranslated: apunta a una variable booleana que recibirá un valor dependiendo del éxito o fallo de la función. TRUE indica éxito, FALSE indica fallo.

Este parámetro es opcional: puede ser NULL. En éste caso, la función no devuelve información sobre el éxito.

bSigned: especifica si la función debe examinar el texto buscando un signo menos al principio y devolver un valor entero con signo si lo encuentra. TRUE indica que esto debe hacerse y FALSE que no.

Valor de retorno:

Si la función tiene éxito, se asignará TRUE a la variable apuntada por lpTranslated y el valor de retorno será el valor del texto en el interior del control edit.

Si la función falla, se asignará FALSE a la variable apuntada por lpTranslated y el valor de retorno es cero. Observar que, como cero es un posible valor para el texto en el control edit, un valor de retorno nulo no tiene por qué indicar un error.

Si lpTranslated es NULL, la función no devuelve ninguna información sobre el éxito.

Si el parámetro bSigned es TRUE, especifica que el valor a recuperar es un entero con signo, debe hacerse un casting del valor de retorno a un tipo int.

Observaciones:

La función `GetDlgItemInt` envía un mensaje [WM_GETTEXT](#) al control especificado. La función convierte el texto recuperado eliminando cualquier espacio al principio del texto y a continuación convirtiendo los dígitos decimales. La función interrumpe la conversión cuando encuentra el final del texto o un carácter ni numérico.

Si el parámetro `bSigned` es `TRUE`, la función `GetDlgItemInt` comprueba la existencia al principio de un signo menos (-) y convierte el texto a un valor entero con signo. En otro caso, la función crea un valor entero sin signo.

La función `GetDlgItemInt` devuelve cero si el valor convertido es mayor que `INT_MAX` (para números con signo) o `UINT_MAX` (para números sin signo).

GetDlgItemText:

La función GetDlgItemText recupera el título o el texto asociado con un control en un cuadro de diálogo.

Sintaxis:

```
UINT GetDlgItemText(  
    HWND hDlg,           // manipulador al cuadro de diálogo  
    int nIDDlgItem, // identificador del control  
    LPCTSTR lpString,   // dirección del buffer de texto  
    int nMaxCount      // máximo tamaño de la cadena  
);
```

Parámetros:

hDlg: identifica al cuadro de diálogo que contienen el control.

nIDDlgItem: especifica el identificador del control del que se recuperará el título o el texto.

lpString: puntero al buffer que recibirá el título o el texto.

nMaxCount: especifica la longitud máxima, en caracteres, de la cadena a copiar en le buffer apuntado por lpString. Si la longitud de la cadena excede el límite, la cadena se truncará.

Valor de retorno:

Si la función tiene éxito, el valor de retorno indicará el número de caracteres copiados a buffer, sin incluir el carácter nulo de terminación.

Si la función falla, el valor de retorno es cero.

Observaciones:

La función GetDlgItemText envía un mensaje [WM_GETTEXT](#) al control especificado.

GetMessage:

Recupera un mensaje de la lista de mensajes de la aplicación y lo coloca en la estructura especificada. Esta función puede recuperar tanto mensajes asociados con una ventana concreta, como mensajes enviados con la función [PostThreadMessage](#). La función recuperará los mensajes en el rango especificado. Pero no recuperará mensajes de ventanas que pertenezcan a otros procesos o aplicaciones.

Sintaxis:

```
BOOL GetMessage(  
    LPMSG lpMsg, // puntero a la estructura que contendrá el mensaje  
    HWND hWnd, // manipulador de ventana  
    UINT wMsgFilterMin, // primer mensaje  
    UINT wMsgFilterMax // último mensaje  
);
```

Parámetros:

lpMsg: puntero a una estructura de tipo [MSG](#) que recibirá la información sobre el mensaje procedente de la lista de mensajes del proceso.

hWnd: identificador de la ventana de la que se recibirán los mensajes. Si el valor es NULL GetMessage recibirá mensajes para cualquier ventana que pertenezca al proceso desde el que se llama, o mensajes enviados al proceso usando [PostThreadMessage](#).

wMsgFilterMin: especifica el valor entero para el límite inferior del rango de mensajes a recibir.

wMsgFilterMax: especifica el valor entero para el límite superior del rango de mensajes a recibir.

Valor de retorno:

Si la función recibe un mensaje distinto de WM_QUIT, el valor de retorno es TRUE. Si recibe WM_QUIT, retornará con FALSE. Si hubo un error, retornará con -1. Por ejemplo la función falla si se llama con un manipulador de ventana no válido.

Observaciones:

Normalmente, el valor de retorno se usa para detectar el final del bucle de mensajes y salir del programa.

Esta función sólo recibe mensajes asociados a la ventana identificada por el parámetro hWnd o cualquiera de sus ventanas hija y dentro del rango de mensajes especificado por los parámetros wParamFilterMin y wParamFilterMax. Si hWnd es NULL la función tomará mensajes procedentes de cualquier ventana que pertenezca al proceso desde donde se usa y mensajes enviados a través de [PostThreadMessage](#). Esta función no recibe mensajes de ventanas que pertenezcan a otros procesos ni tampoco de otros procesos. Los mensajes de procesos, enviados por la función PostThreadMessage, tienen como valor de hWnd NULL. Si wParamFilterMin y wParamFilterMax son cero, la función tomará todos los mensajes disponibles, es decir, no se filtran los mensajes.

Las constantes WM_KEYFIRST y WM_KEYLAST pueden usarse para filtrar los mensajes procedentes desde el teclado; las constantes WM_MOUSEFIRST y WM_MOUSELAST para filtrar los mensajes procedentes del ratón.

GetMessage no retira los mensajes [WM_PAINT](#) de la lista de mensajes. Estos mensajes permanecen en la lista hasta que son procesados.

Observa que la función puede retornar con TRUE, FALSE, o -1. Así que se debe evitar usar expresiones como:

```
while (GetMessage( lpMsg, hWnd, 0, 0) ...
```

La posibilidad de un valor de retorno de -1 significa que la aplicación puede obtener errores fatales.

GetScrollInfo:

[Nuevo - Windows NT] (No puede usarse en win32s, sí en Win95 y Win NT)

Recupera los parámetros de un scrollbar, incluyendo las posiciones mínima y máxima de desplazamiento, el tamaño de la página y la posición de la caja de desplazamiento (thumb).

Sintaxis:

```
BOOL GetScrollInfo(  
    HWND hwnd,           // manipulador de la ventana con el scrollbar  
    int fnBar,           // flag del scrollbar  
    LPSCROLLINFO lpsi, // puntero a estructura con los parámetros de  
    desplazamiento  
);
```

Parámetros:

hwnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro fnBar.

fnBar: especifica el tipo de scrollbar cuyos parámetros se recuperarán. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Recupera los parámetros de un control scrollbar. El parámetro hwnd debe ser un manipulador del control scrollbar.
SB_HORZ	Recupera los parámetros del scrollbar estándar horizontal de una ventana.
SB_VERT	Recupera los parámetros del scrollbar estándar vertical de una ventana.

lpsi: puntero a una estructura [SCROLLINFO](#) cuyo miembro fMask, especifica los parámetros del scrollbar a recuperar. Antes de retornar, la función copia los parámetros especificados a los miembros apropiados de la estructura.

El miembro fMask puede ser una combinación de los siguientes valores:

Valor	Significado
SIF_ALL	Combinación de SIF_PAGE, SIF_POS y SIF_RANGE.
SIF_PAGE	Copia el valor de la página de desplazamiento al miembro nPage de la estructura SCROLLINFO apuntada por lpsi.
SIF_POS	Copia el valor de la posición de desplazamiento al miembro nPos de la estructura SCROLLINFO apuntada por lpsi.

SIF_RANGE	Copia el rango de desplazamiento a los miembros nMin y nMax de la estructura SCROLLINFO apuntada por lpsi.
-----------	--

fRedraw: especifica si el scrollbar debe ser redibujado para reflejar el cambio. Si es TRUE, el scrollbar es repintado. Si es FALSE, no.

Valor de retorno:

Si la función recupera algún valor, retorna con TRUE. Si no recupera ninguno el valor de retorno es FALSE.

Observaciones:

La función GetScrollInfo permite usar posiciones de desplazamiento de 32-bits. Como los mensajes que indican posiciones de scrollbar, [WM_HSCROLL](#) y [WM_VSCROLL](#), están limitados a 16 bits para el dato de la posición, las funciones [SetScrollInfo](#) y GetScrollInfo suministran valores de posiciones de 32 bit. Así que, una aplicación puede llamar a GetScrollInfo mientras procesa mensajes WM_HSCROLL o WM_VSCROLL para obtener posiciones de 32-bits.

La limitación de esta técnica aparece cuando se procesan desplazamientos en tiempo real. Ese proceso se hace respondiendo a los mensajes WM_HSCROLL o WM_VSCROLL que poseen el mensaje de notificación SB_THUMBTRACK, y actualizando la posición de la caja de scroll (thumb) al mismo tiempo que el usuario la mueve. Desgraciadamente, no existe ninguna función para recuperar la posición de 32-bits del thumb mientras el usuario lo mueve. GetScrollInfo proporciona sólo un dato de posición estática; las aplicaciones sólo pueden obtener posiciones de 32-bits antes o después de que el desplazamiento haya tenido lugar.

GetScrollPos:

Recupera la posición actual de la caja de desplazamiento (thumb) de un scrollbar específico. La posición actual es un valor relativo que depende del rango del scrollbar. Por ejemplo, si el rango de desplazamiento es entre 0 y 100 y la caja de desplazamiento está en el centro de la barra, la posición actual es 50.

Sintaxis:

```
int GetScrollPos(  
    HWND hWnd,           // manipulador de la ventana con el scrollbar  
    int nBar             // flag de scrollbar
```

Parámetros:

hWnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro nBar.

nBar: especifica el scrollbar a modificar. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Recupera la posición de la caja de desplazamiento de un control scrollbar. El parámetro hWnd debe ser un manipulador del control scrollbar.
SB_HORZ	Recupera la posición de la caja de desplazamiento del scrollbar estándar horizontal de una ventana.
SB_VERT	Recupera la posición de la caja de desplazamiento del scrollbar estándar vertical de una ventana.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es el valor de la posición actual de la caja de desplazamiento.

Si falla, el valor de retorno es cero. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

La función GetScrollPos permite a las aplicaciones usar posiciones de scroll de 32-bits. A pesar de que los mensajes que indican posiciones de scrollbar, [WM_HSCROLL](#) y [WM_VSCROLL](#), están limitadas a posiciones de 16 bits, las funciones [SetScrollPos](#), [SetScrollRange](#), GetScrollPos y [GetScrollRange](#) soportan posiciones de scrollbar de 32-bit. De modo que una aplicación puede llamar a

GetScrollPos mientras procesa mensajes WM_HSCROLL o WM_VSCROLL para obtener datos de posiciones de 32-bits.

La limitación de esta técnica aparece cuando se procesan desplazamientos en tiempo real. Ese proceso se hace respondiendo a los mensajes WM_HSCROLL o WM_VSCROLL que poseen el mensaje de notificación SB_THUMBTRACK, y actualizando la posición de la caja de scroll (thumb) al mismo tiempo que el usuario la mueve. Desgraciadamente, no existe ninguna función para recuperar la posición de 32-bits del thumb mientras el usuario lo mueve. GetScrollPos proporciona sólo un dato de posición estática; las aplicaciones sólo pueden obtener posiciones de 32-bits antes o después de que el desplazamiento haya tenido lugar.

GetScrollRange:

Recupera los valores de las posiciones mínimas y máximas de la caja de desplazamiento (thumb) de un scrollbar especificado.

Para Windows 95, esta función existe por compatibilidad con versiones de sistemas operativos anteriores a la 4.0. Para la versión 4.0 o superiores, debe usarse la función [GetScrollInfo](#).

Sintaxis:

```
BOOL GetScrollRange(  
    HWND hWnd,           // manipulador de la ventana con el scrollbar  
    int nBar,           // flag del scrollbar  
    LPINT lpMinPos,     // dirección de la variable que recibe la posición mínima  
    LPINT lpMaxPos     // dirección de la variable que recibe la posición máxima  
);
```

Parámetros:

hWnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro nBar.

nBar: especifica el scrollbar cuyos valores de rango queremos recuperar. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Recupera el rango de un control scrollbar. El parámetro hWnd debe ser un manipulador del control scrollbar.
SB_HORZ	Recupera el rango del scrollbar estándar horizontal de una ventana.
SB_VERT	Recupera el rango del scrollbar estándar vertical de una ventana.

lpMinPos: apunta a una variable entera que recibirá la posición mínima de desplazamiento.

lpMaxPos: apunta a una variable entera que recibirá la posición máxima de desplazamiento.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

Si la ventana especificada no tiene scrollbars estándar o no se trata de un control scrollbar, la función `GetScrollRange` pone a cero los parámetros `lpMinPos` y `lpMaxPos`.

El rango por defecto de un scrollbar estándar es de 0 a 100. Para controles scrollbar es un rango vacío (ambos valores son cero).

Los mensajes que indican posiciones de scrollbar, [WM_HSCROLL](#) y [WM_VSCROLL](#), están limitados a 16 bits para el dato de la posición. Sin embargo, como las funciones [SetScrollPos](#), [SetScrollRange](#), [GetScrollPos](#) y [GetScrollRange](#) soportan valores de 32-bit para la posición del scrollbar, existe un modo de sortear la barrera de 16-bit de los mensajes `WM_HSCROLL` y `WM_VSCROLL`. Ver [GetScrollPos](#) para ver una descripción de esa técnica y sus límites.

InsertMenu:

La función InsertMenu inserta un nuevo ítem de menú dentro de un menú, moviendo los otros ítems hacia abajo en el menú.

Sintaxis:

```
BOOL InsertMenu(  
    HMENU hMenu,      // manipulador de menú  
    UINT uPosition, // ítem de menú al que el nuevo ítem precederá  
    UINT uFlags,     // flags del ítem de menú  
    UINT uIDNewItem, // identificador de ítem de menú o manipulador de menú  
    pop-up  
    LPCTSTR lpNewItem // contenido del ítem de menú  
);
```

Parámetros:

hMenu: identifica el menú a modificar.

uPosition: indica el ítem de menú delante del cual se insertará el nuevo, tal como se determine por el parámetro uFlags.

uFlags: especifica los flags para controlar la interpretación del parámetro uPosition y el contenido, apariencia y el comportamiento del ítem de menú. Este parámetro puede ser una combinación de uno de los valores requeridos siguientes y por lo menos uno de los valores listados en la sección de observaciones.

Valor	Descripción
MF_BYCOMMAND	Indica que el parámetro uPosition toma el identificador del ítem de menú. El flag MF_BYCOMMAND es el valor por defecto si no se indica el flag MF_BYCOMMAND ni MF_BYPOSITION.
MF_BYPOSITION	Indica que el parámetro uPosition toma la posición relativa, basada en cero del nuevo ítem de menú. Si uPosition es 0xFFFFFFFF, el nuevo ítem se añadirá al final del menú.

uIDNewItem: especifica o bien el identificador del nuevo ítem de menú o si el parámetro uFlags incluye MF_POPUP, el manipulador de un menú pop-up.

lpNewItem: especifica el contenido del nuevo ítem de menú. La interpretación de lpNewItem depende de si el parámetro uFlags incluye los flags MF_BITMAP, MF_OWNERDRAW o MF_STRING, como se especifica a continuación:

Valor	Descripción
MF_BITMAP	Contiene un manipulador de bitmap.
MF_OWNERDRAW	Contiene un valor de 32 bits suministrado por la aplicación que puede ser usado para mantener datos adicionales relacionados con el ítem de menú. El valor esta en el miembro itemData member de la estructura apuntada por el parámetro lparam del mensaje WM_MEASUREITEM o WM_DRAWITEM enviado cuando el menú es creado o su apariencia es actualizada.
MF_STRING	Contiene un puntero a una cadena de caracteres terminada con cero.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

La aplicación debe llamar a la función [DrawMenuBar](#) cada vez que el menú cambie, tanto si el menú pertenece a una ventana que se está mostrando como si no.

Los siguientes flags deben ser usados en el parámetro uFlags:

Valor	Descripción
MF_BITMAP	Usa un bitmap como ítem. El parámetro lpNewItem contiene el manipulador del bitmap.
MF_CHECKED	Pone una marca de comprobación junto al ítem. Si la aplicación proporciona los bitmaps de las marcas de comprobación (ver SetMenuItemBitmaps), éste flag muestra la marca junto al ítem de

	menú.
MF_DISABLED	Deshabilita el ítem de menú de modo que no puede ser seleccionado, pero éste flag no pone el ítem en gris.
MF_ENABLED	Habilita en ítem de menú de modo que pueda ser seleccionado y restaura su apariencia si su estado estaba en gris.
MF_GRAYED	Deshabilita el ítem y lo muestra en gris de modo que no puede ser seleccionado.
MF_MENUBARBREAK	Funciona igual que el flag MF_MENUBREAK excepto para menús pop-up, donde la nueva columna es separada de la anterior por una línea vertical.
MF_MENUBREAK	Coloca el ítem en una línea nueva (para barras de menú) o en una columna nueva (para menús pop-up) sin columnas separadas.
MF_OWNERDRAW	Especifica que el ítem es dibujado por la ventana propietaria, (owner-drawn). Antes de que el menú sea mostrado por primera vez, la ventana que posee el menú recibirá un mensaje WM_MEASUREITEM para recuperar la anchura y altura del ítem de menú. El mensaje WM_DRAWITEM es enviado al procedimiento de ventana de la ventana propietaria cada vez que la apariencia del ítem deba ser actualizada.
MF_POPUP	Indica que el ítem es un ítem pop-up; es decir, al seleccionarlo se activa un menú pop-up. El parámetro uIDNewItem especifica el manipulador del menú pop-up. El flag es usado para añadir un ítem pop-up a una barra de menú o a un menú pop-up.
MF_SEPARATOR	Dibuja una línea horizontal de división. Este flag se usa sólo en menús pop-up. La línea no puede ser puesta en gris, deshabilitada o resaltada. Los parámetros lpNewItem y uIDNewItem se ignoran.
MF_STRING	Especifica que el ítem es una cadena

	de texto; el parámetro lpNewItem apunta a una cadena.
MF_UNCHECKED	No pone la marca de comprobación junto al ítem (valor por defecto). Si la aplicación suministra bitmaps para las marcas de comprobación (ver SetMenuItemBitmaps), este flag muestra el bitmap de no comprobación junto al ítem.

La lista siguiente muestra grupos de flags que no pueden ser usados juntos:

- MF_BYCOMMAND y MF_BYPOSITION
- MF_DISABLED, MF_ENABLED y MF_GRAYED
- MF_BITMAP, MF_STRING, MF_OWNERDRAW y MF_SEPARATOR
- MF_MENUBARBREAK y MF_MENUBREAK
- MF_CHECKED y MF_UNCHECKED

LoadMenu:

La función LoadMenu carga el recurso de menú especificado desde el fichero ejecutable (.EXE) asociado con la instancia de la aplicación.

Sintaxis:

```
HMENU LoadMenu(  
    HINSTANCE hInstance,      // manipulador de instancia de la aplicación  
    LPCTSTR lpMenuName       // cadena con el nombre del menú o  
    identificador de recurso de menú  
);
```

Parámetros:

hInstance: identifica la instancia del módulo que contiene el recurso de menú a cargar.

lpMenuName: apunta a una cadena terminada en cero que contiene el nombre del recurso de menú. Alternativamente, éste parámetro puede ser un identificador de recurso en la palabra de menor peso y cero en la de mayor peso. Para crear éste valor usar la macro [MAKEINTRESOURCE](#).

Valor de retorno:

Si la función tiene éxito el valor de retorno será el manipulador del recurso de menú cargado.

Si la función falla, el valor de retorno será NULL. Para obtener información adicional usar la función [GetLastError](#).

Observaciones:

Se usa la función DestroyMenu, antes de cerrar la aplicación, para destruir el menú y liberar la memoria ocupada por el menú cargado.

MessageBox:

La función MessageBox crea, muestra y ejecuta un cuadro de mensaje. El cuadro de mensaje contiene un mensaje definido por la aplicación y un título, y además cualquier combinación de iconos predefinidos y botones.

Sintaxis:

```
int MessageBox(  
    HWND hWnd, // manipulador de la ventana propietaria  
    LPCTSTR lpText, // dirección del texto del mensaje  
    LPCTSTR lpCaption, // dirección del texto del título  
    UINT uType // estilo del cuadro de mensaje  
);
```

Parámetros:

hWnd: identifica a la ventana propietaria del cuadro de mensaje a crear. Si éste parámetro es NULL, el cuadro de mensaje no tendrá ventana propietaria.

lpText: apunta a una cadena terminada en cero que contiene el mensaje a mostrar.

lpCaption: apunta a una cadena terminada en cero usada como título para el cuadro de diálogo. Si este parámetro es NULL, se usará el título por defecto de Error.

uType: especifica el contenido y comportamiento del cuadro de diálogo. Este parámetro puede ser una combinación de los siguientes valores:

Valor	Significado
MB_ABORTRETRYIGNORE	El cuadro de mensaje contiene tres botones: Anular, Reintentar e Omitir.
MB_APPLMODAL	El usuario debe responder al cuadro de diálogo antes de continuar trabajando en la ventana identificada por el parámetro hWnd. Sin embargo, el usuario puede trasladarse a ventanas de otras aplicaciones y trabajar en sus ventanas. Dependiendo de la gerarquía de ventanas de la aplicación, el usuario puede estar

	<p>capacitado para moverse a otras ventanas dentro de la misma aplicación. Todas las ventanas hija de la ventana padre del cuadro de mensaje serán automáticamente deshabilitadas, pero las ventanas popup no.</p> <p>MB_APPLMODAL es el valor por defecto si no se especifica MB_SYSTEMMODAL ni MB_TASKMODAL.</p>
MB_DEFAULT_DESKTOP_ONLY	<p>El escritorio que actualmente recibe la entrada debe ser el escritorio por defecto; en otro caso, la función falla. Un escritorio por defecto es aquel en que una aplicación se ejecuta después de que el usuario ha hecho logon.</p>
MB_DEFBUTTON1	<p>El primer botón es el botón por defecto. Observa que el primer botón es siempre el botón por defecto salvo que se especifique MB_DEFBUTTON2, MB_DEFBUTTON3 o MB_DEFBUTTON4.</p>
MB_DEFBUTTON2	<p>El segundo botón es el botón por defecto.</p>
MB_DEFBUTTON3	<p>El tercer botón es el botón por defecto.</p>
MB_DEFBUTTON4	<p>El cuarto botón es el botón por defecto.</p>
MB_HELP	<p>Windows 95: añade un botón de ayuda al cuadro de mensaje. Pulsando el botón de ayuda o la tecla F1 se genera un evento de ayuda.</p>
MB_ICONASTERISK	<p>Igual que MB_ICONINFORMATION.</p>
MB_ICONERROR	<p>Windows 95: igual que MB_ICONHAND.</p>
MB_ICONEXCLAMATION	<p>Se mostrará un icono de</p>

	exclamación.
MB_ICONHAND	Lo mismo que MB_ICONSTOP.
MB_ICONINFORMATION	Se mostrará un icono que consiste en una 'i' minúscula en in bocadillo
MB_ICONQUESTION	Se mostrará un icono con una interrogación.
MB_ICONSTOP	Se mostrará un icono con un signo de stop.
MB_ICONWARNING	Windows 95: igual que MB_ICONEXCLAMATION.
MB_OK	El cuadro de mensaje contiene un único botón de Aceptar.
MB_OKCANCEL	El cuadro de mensaje contiene dos botones: Aceptar y Cancelar.
MB_RETRYCANCEL	El cuadro de mensaje contiene dos botones: Reintentar y Cancelar.
MB_RIGHT	Windows 95: El texto estará justificado a la derecha.
MBRTLREADING	Windows 95: Muestra el mensaje y el título usando el sentido de lectura de derecha a izquierda en sistemas Hebreos y Árabes.
MB_SERVICE_NOTIFICATION	Windows NT: el proceso que llama al cuadro de mensaje es un servicio que informa al usuario de un evento. La función muestra un cuadro de mensaje en el escritorio activo aunque no exista ningún usuario activo en el ordenador. Si éste flag está activo, el parámetro hWnd debe ser NULL. Este el cuadro de mensaje también puede aparecer en un escritorio diferente del correspondiente al de hWnd.

MB_SETFOREGROUND	El cuadro de mensaje se convierte en la ventana de primer plano. Internamente, Windows llama a la función SetForegroundWindow para el cuadro de mensaje.
MB_SYSTEMMODAL	Todas las aplicaciones son suspendidas hasta que el usuario responda al cuadro de mensaje. A no ser que la aplicación especifique MB_ICONHAND, el cuadro de mensaje no se hace modal hasta después de creado; consecuentemente, la ventana propietaria y otras ventanas continúan recibiendo mensajes resultantes de su activación. Usar cuadros de mensaje system-modal para notificar al usuario de serios y errores que potencialmente puedan dañar y requieran atención inmediata (por ejemplo, ejecución sin memoria).
MB_TASKMODAL	Lo mismo que MB_APPLMODAL excepto que que todas las ventanas de mayor nivel pertenecientes al proceso (task) actual serán deshabilitadas si el parámetro hWnd es NULL. Usar éste flag cuando la aplicación o librería que la llama no tiene un manipulador de ventana disponible pero but necesita impedir una entrada a otras ventanas de la aplicación actual sin suspender otras aplicaciones.
MB_YESNO	El cuadro de mensaje contiene dos botones: Sí y No.
MB_YESNOCANCEL	El cuadro de mensaje contiene tres botones: Sí, No y Cancelar.

Valor de retorno:

El valor de retorno es cero si no ha suficiente memoria para crear el cuadro de mensaje.

Si la función tiene éxito, el valor de retorno es uno de los siguientes valores de ítem de menú devueltos por el cuadro de diálogo:

Valor	Significado
IDABORT	Se seleccionó el botón de Anular.
IDCANCEL	Se seleccionó el botón de Cancelar.
IDIGNORE	Se seleccionó el botón de Omitir.
IDNO	Se seleccionó el botón de No.
IDOK	Se seleccionó el botón de Aceptar.
IDRETRY	Se seleccionó el botón de Reintentar.
IDYES	Se seleccionó el botón de Sí.

Si un cuadro de mensaje contiene un botón de Cancelar, la función devuelve el valor IDCANCEL tanto si se pulsa la tecla ESC como si se pulsa el botón de Cancelar. Si el cuadro de mensaje no tiene el botón de Cancelar, pulsar ESC no tiene efecto.

Observaciones:

Cuando uses un cuadro de mensaje de tipo para indicar que el sistema está bajo de memoria, las cadenas apuntadas por los parámetros lpText y lpCaption no deben ser tomados desde un fichero de recursos, porque un intento de cargar un recurso podría fallar.

Cuando una aplicación llama a MessageBox y especifica los flags MB_ICONHAND y MB_SYSTEMMODAL para el parámetro uType, Windows muestra el cuadro de mensaje resultante independientemente de la memoria disponible. Cuando esos flags son especificados, Windows limita la longitud del texto del mensaje a tres líneas. Windows no rompe automáticamente las líneas para que quepan en el cuadro de mensaje, así que la cadena del mensaje debe contener retornos de línea para separar las líneas en los lugares apropiados.

Si creas un cuadro de mensaje mientras un cuadro de diálogo está abierto, usa el manipulador del cuadro de diálogo como parámetro para hWnd. El parámetro hWnd no debe identificar a una ventana hija como por ejemplo un control de un cuadro de diálogo.

Sólo en Windows 95: el sistema puede soportar un máximo de 16,364 manipuladores de ventana.

PostMessage:

La función PostMessage coloca (postea) un mensaje en la cola de mensajes asociada con el proceso que creó la ventana especificada y después retorna sin esperar a que el mensaje sea procesado. Los mensajes en la cola de mensajes son recuperados por llamadas a las funciones [GetMessage](#) o [PeekMessage](#).

Sintaxis:

```
BOOL PostMessage(  
    HWND hwnd,           // manipulador de la ventana de destino  
    UINT uMsg,           // mensaje a enviar  
    WPARAM wParam,      // primer parámetro del mensaje  
    LPARAM lParam        // segundo parámetro del mensaje  
);
```

Parámetros:

hwnd: identifica la ventana cuyo procedimiento de ventana recibirá el mensaje. Hay dos valores con significados especiales:

HWND_BROADCAST: el mensaje se envía a todas las ventanas de nivel superior en el sistema, incluyendo las incluyendo las deshabilitadas o invisibles sin dueño, ventanas superpuestas u ventanas pop-up; pero el mensaje no se envía a las ventanas hijas.

NULL: la función se comporta como una llamada a PostThreadMessage con el valor del identificador del proceso actual como parámetro dwThreadId.

uMsg: especifica el mensaje a enviar.

wParam: especifica información adicional específica para el mensaje.

lParam: especifica información adicional específica para el mensaje.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si la función falla, el valor de retorno es FALSE. Para obtener información adicional sobre el motivo del error, hay que llamar a la función [GetLastError](#).

Observaciones:

Una aplicación de 32 bits no debe postear mensajes menores que WM_USER que contengan punteros.

PostQuitMessage:

Indica a Windows que el "hilo" o proceso ha hecho una petición para terminar. Esta función se usa normalmente como respuesta a un mensaje [WM_DESTROY](#).

Sintaxis:

```
VOID PostQuitMessage(  
    int nExitCode // código se salida  
);
```

Parámetros:

nExitCode: indica un código de salida de la aplicación. Este valor será usado como parámetro wParam del mensaje [WM_QUIT](#).

Valor de retorno:

No tiene valor de retorno.

Observaciones:

La función PostQuitMessage envía un mensaje [WM_QUIT](#) a las cola de mensajes del "hilo" y vuelve inmediatamente; la función indica simplemente al sistema que el "hilo" está pidiendo salir en algún momento del futuro.

Cuando el "hilo" recupera el mensaje WM_QUIT desde la cola de mensajes, debe salir del bucle de mensajes y devolver el control a Windows. El valor de salida que es devuelto a Windows debe ser el parámetro wParam del mensaje WM_QUIT.

RegisterClass:

Esta función registra una clase de ventana para que pueda ser usada en llamadas con [CreateWindow](#) o [CreateWindowEx](#).

Sintaxis:

```
ATOM RegisterClass(  
    CONST WNDCLASS *lpwc // dirección de la estructura con los datos de la  
    clase  
);
```

Parámetros:

lpwc: puntero a una estructura del tipo [WNDCLASS](#). Es necesario rellenar adecuadamente la estructura antes de llamar a la función.

Valor de retorno:

Si la función tiene éxito, retorna con un átomo que es un identificador único para la clase registrada. Si la función falla, retorna con valor nulo. Para conseguir más información sobre el motivo del error se puede llamar a [GetLastError](#).

Observaciones:

Todas las clases que registre una aplicación dejarán de estarlo cuando la aplicación termine.

SendDlgItemMessage:

La función SendDlgItemMessage envía un mensaje al control especificado de un cuadro de diálogo.

Sintaxis:

```
LONG SendDlgItemMessage(  
    HWND hwndDlg,    // manipulador del cuadro de diálogo  
    int idControl,   // identificador del control  
    UINT uMsg,       // mensaje a enviar  
    WPARAM wParam,  // primer parámetro del mensaje  
    LPARAM lParam    // segundo parámetro del mensaje  
);
```

Parámetros:

hwndDlg: identifica al cuadro de diálogo que contienen el control.

idControl: especifica el identificador del control que recibe el mensaje.

uMsg: especifica el mensaje a enviar.

wParam: especifica información adicional específica para el mensaje.

lParam: especifica información adicional específica para el mensaje.

Valor de retorno:

El valor de retorno especifica el resultado del procesamiento del mensaje y depende del mensaje enviado.

Observaciones:

La función SendDlgItemMessage no retorna hasta que el mensaje haya sido procesado.

Usar SendDlgItemMessage es lo mismo que tomar el manipulador del control especificado y llamar a la función [SendMessage](#).

SendMessage:

La función SendMessage envía el mensaje especificado a una o varias ventanas. La función llama al procedimiento de ventana de la ventana especificada y no retorna hasta que el procedimiento de ventana ha procesado el mensaje. La función [PostMessage](#), por otra parte, postea un mensaje a la cola de mensajes del proceso y retorna inmediatamente.

Sintaxis:

```
LRESULT SendMessage(  
    HWND hwnd,           // manipulador de la ventana de destino  
    UINT uMsg,           // mensaje a enviar  
    WPARAM wParam,      // primer parámetro del mensaje  
    LPARAM lParam        // segundo parámetro del mensaje  
);
```

Parámetros:

hwnd: identifica la ventana cuyo procedimiento de ventana recibirá el mensaje. Si este parámetro es HWND_BROADCAST, el mensaje se envía a todas las ventanas de nivel superior en el sistema, incluyendo las deshabilitadas o invisibles sin dueño, ventanas superpuestas u ventanas pop-up; pero el mensaje no se envía a las ventanas hijas.

uMsg: especifica el mensaje a enviar.

wParam: especifica información adicional específica para el mensaje.

lParam: especifica información adicional específica para el mensaje.

Valor de retorno:

El valor de retorno especifica el resultado del procesamiento del mensaje y depende del mensaje enviado.

Observaciones:

Si la ventana especificada fue creada por el proceso que llama, el procedimiento de ventana es llamado inmediatamente como si fuera una subrutina. En caso contrario, Windows cambia a ese proceso y llama al procedimiento de ventana adecuado.

SetDlgItemInt:

La función SetDlgItemInt cambia el texto de un control en un cuadro de diálogo a la representación en forma de cadena de un valor entero especificado.

Sintaxis:

```
BOOL SetDlgItemInt(  
    HWND hwndDlg, // manipulador al cuadro de diálogo  
    int idControl, // identificador del control  
    UINT uValue, // valor a cambiar  
    BOOL fSigned // indicador de con o sin signo  
);
```

Parámetros:

hwndDlg: identifica al cuadro de diálogo que contienen el control.

idControl: especifica el identificador del control que recibe el mensaje.

uValue: especifica el valor entero usado para generar el texto.

fSigned: especifica si el parámetro uValue es con o sin signo. Si este parámetro es TRUE, uValue es un valor con signo. Si este parámetro es TRUE y uValue es menor que cero, se insertará un signo menos antes del primer dígito de la cadena. Si el parámetro es FALSE, uValue es un valor sin signo.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si la función falla, el valor de retorno es FALSE. Para obtener más información, hay que usar la función [GetLastError](#).

Observaciones:

Para cambiar el nuevo texto, la función SetDlgItemInt envía un mensaje [WM_SETTEXT](#) al control especificado.

SetDlgItemText:

La función SetDlgItemText cambia el título o el texto de un control de un cuadro de diálogo.

Sintaxis:

```
BOOL SetDlgItemText(  
    HWND hwndDlg, // manipulador al cuadro de diálogo  
    int idControl, // identificador del control  
    LPCTSTR lpsz // texto a poner  
);
```

Parámetros:

hwndDlg: identifica al cuadro de diálogo que contienen el control.

idControl: especifica el identificador del control que recibe el mensaje.

lpsz: puntero a cadena terminada en cero que contiene el texto a ser copiado al control.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si la función falla, el valor de retorno es FALSE. Para obtener más información, hay que usar la función [GetLastError](#).

Observaciones:

La función SetDlgItemText envía un mensaje [WM_SETTEXT](#) al control especificado.

SetFocus:

La función SetFocus asigna el foco del teclado a la ventana especificada. Todas las entradas desde el teclado a partir de ese momento se dirigen a esa ventana. La ventana que tenía el foco del teclado previamente, si es que había alguna, lo pierde.

Sintaxis:

```
HWND SetFocus(  
    HWND hwnd           // manipulador a la ventana que recibirá el  
    foco  
);
```

Parámetros:

hwnd: identifica la ventana que recibirá el foco del teclado. Si este parámetro es NULL, las pulsaciones del teclado se ignoran.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es el manipulador de la ventana que tenía previamente el foco del teclado. Si no existe tal ventana, o el parámetro hwnd es inválido, el valor de retorno es NULL.

Observaciones:

Si la ventana identificada por el parámetro hwnd fue creada por el proceso llamado, el estado del foco del teclado del proceso llamado se pone a hwnd.

La función SetFocus envía un mensaje [WM_KILLFOCUS](#) a la ventana que pierde el foco del teclado un mensaje [WM_SETFOCUS](#) a la ventana que lo recibe. También activa la ventana que recibe el foco o la ventana padre de la ventana que recibe el foco.

Si una ventana está activa pero no tiene el foco del teclado (es decir, ninguna ventana tiene el foco), cualquier tecla pulsada producirá un mensaje [WM_SYSCHAR](#), [WM_SYSKEYDOWN](#) o [WM_SYSKEYUP](#). Si la tecla VK_MENU está pulsada también, el parámetro lParam del mensaje tendrá el bit 30 activado. En otro caso, los mensajes producidos no tendrán este bit activo.

Si una aplicación no está en primer plano, y se quiere que lo esté, debe llamarse a la función [SetForegroundWindow](#).

SetMenu:

La función SetMenu asigna un nuevo menú a la ventana especificada.

Sintaxis:

```
BOOL SetMenu(  
    HWND hwnd,           // manipulador de ventana  
    HMENU hmenu         // manipulador de menú  
);
```

Parámetros:

hwnd: identifica la ventana a la que se asignará el menú.

hmenu: identifica el nuevo menú. Si el parámetro es NULL, el menú actual de la ventana será eliminado.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

La ventana es redibujada para reflejar el cambio de menú.

La función SetMenu reemplaza el menú anterior, si existe, pero no lo destruye. La aplicación debe llamar a la función [DestroyMenu](#) para cumplimentar esa tarea.

SetScrollInfo:

[Nuevo - Windows NT] (No puede usarse en win32s, sí en Win95 y Win NT)

Cambia los parámetros de un scrollbar, incluyendo las posiciones mínima y máxima de desplazamiento, el tamaño de la página y la posición de la caja de desplazamiento (thumb). También redibuja el scrollbar, si se requiere.

Sintaxis:

```
int SetScrollInfo(  
    HWND hwnd,                // manipulador de la ventana  
    con el scrollbar  
    int fnBar,                // flag del scrollbar  
    LPSCROLLINFO lpsi,       // puntero a estructura con los  
    parámetros de desplazamiento  
    BOOL fRedraw              // flag de repintado  
);
```

Parámetros:

hwnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro fnBar.

fnBar: especifica el tipo de scrollbar cuyos parámetros se modificarán. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Cambia los parámetros de un control scrollbar. El parámetro hwnd debe ser un manipulador del control scrollbar.
SB_HORZ	Cambia los parámetros del scrollbar estándar horizontal de una ventana.
SB_VERT	Cambia los parámetros del scrollbar estándar vertical de una ventana.

lpsi: puntero a una estructura [SCROLLINFO](#) cuyo miembro fMask, especifica los parámetros del scrollbar a modificar.

El miembro fMask puede ser una combinación de los siguientes valores:

Valor	Significado
SIF_DISABLENOSCROLL	Deshabilita el scrollbar en lugar de eliminarlo, si los nuevos parámetros hacen que el scrollbar sea innecesario.
SIF_PAGE	Cambia el valor de la página de desplazamiento al valor especificado por el miembro nPage de la estructura SCROLLINFO apuntada por lpsi.
SIF_POS	Cambia el valor de la posición de desplazamiento al valor

	especificado por el miembro nPos de la estructura SCROLLINFO apuntada por lpsi.
SIF_RANGE	Cambia el rango de desplazamiento al valor especificado por los miembros nMin y nMax de la estructura SCROLLINFO apuntada por lpsi.

fRedraw: especifica si el scrollbar debe ser redibujado para reflejar el cambio. Si es TRUE, el scrollbar es repintado. Si es FALSE, no.

Valor de retorno:

El valor de retorno es la posición actual de la caja de desplazamiento.

Observaciones:

La función SetScrollInfo realiza una comprobación de los valores especificados por los miembros nPage y nPos de la estructura SCROLLINFO. El miembro nPage debe especificar un valor entre 0 y nMax - nMin + 1. El miembro nPos debe especificar un valor entre nMin y nMax - max(nPage - 1, 0). Si alguno de esos valores está fuera de rango, la función cambia su valor para que tenga un valor permitido.

SetScrollPos:

Esta función cambia la posición de la caja de desplazamiento (thumb) del scrollbar especificado y, si se requiere, dibuja de nuevo el scrollbar para mostrar la nueva posición. Esta función se incluye para mantener la compatibilidad con Windows 3.x. Las aplicaciones basadas en Win32 deben usar la función [SetScrollInfo](#).

Sintaxis:

```
int SetScrollPos(  
    HWND hwnd,           // manipulador de la ventana con el  
    scrollbar           // scrollbar  
    int fnBar,          // flag de scrollbar  
    int nPos,           // nueva posición de la caja de  
    desplazamiento  
    BOOL fRedraw       // flag de redibujado  
);
```

Parámetros:

hwnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro fnBar.

fnBar: especifica el scrollbar a modificar. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Cambia la posición de un control scrollbar. El parámetro hwnd debe ser un manipulador del control scrollbar.
SB_HORZ	Cambia la posición del scrollbar estándar horizontal de una ventana.
SB_VERT	Cambia la posición del scrollbar estándar vertical de una ventana.

nPos: especifica la nueva posición de la caja de desplazamiento. La posición debe estar dentro del rango del scrollbar. Para más información sobre el rango del scrollbar ver la función [SetScrollRange](#).

fRedraw: especifica si el scrollbar debe ser redibujado para reflejar el cambio. Si es TRUE, el scrollbar es repintado. Si es FALSE, no.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es el valor anterior de la caja de desplazamiento.

Si falla, el valor de retorno es cero. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

Si el scrollbar es redibujado por la llamada a otra función a continuación de esta, es corriente usar FALSE en el parámetro fRedraw.

Como los mensajes que indican posiciones de scrollbar, [WM_HSCROLL](#) y [WM_VSCROLL](#), están limitados a 16 bits para el dato de la posición, las aplicaciones que confían sólo en esos mensajes para actualizar las posiciones tienen un valor máximo de 65,535 para el parámetro nMaxPos de la función SetScrollRange.

Sin embargo, como las funciones SetScrollPos, SetScrollRange, [GetScrollPos](#) y [GetScrollRange](#) soportan valores de 32-bit para la posición del scrollbar, existe un modo de sortear la barrera de 16-bit de los mensajes WM_HSCROLL y WM_VSCROLL. Ver [GetScrollPos](#) para ver una descripción de esa técnica y sus límites.

SetScrollRange:

Asigna los valores mínimo y máximo de los valores de posición de una scrollbar. También puede ser usada para ocultar o mostrar un scrollbar estándar. Esta función se incluye para mantener la compatibilidad con Windows 3.x. Las aplicaciones basadas en Win32 deben usar la función [SetScrollInfo](#).

Sintaxis:

```
BOOL SetScrollRange(  
    HWND hwnd,           // manipulador de la ventana con el  
    scrollbar           // scrollbar  
    int fnBar,          // flag del scrollbar  
    int nMinPos,       // posición mínima de desplazamiento  
    int nMaxPos,       // posición máxima de desplazamiento  
    BOOL fRedraw       // flag de repintado  
);
```

Parámetros:

hwnd: identifica el control scrollbar o la ventana con un scrollbar estándar, dependiendo del valor del parámetro fnBar.

fnBar: especifica el scrollbar a modificar. Puede tener uno de los siguientes valores:

Valor	Significado
SB_CTL	Modifica el rango de un control scrollbar. El parámetro hwnd debe ser un manipulador del control scrollbar.
SB_HORZ	Modifica el rango del scrollbar estándar horizontal de una ventana.
SB_VERT	Modifica el rango del scrollbar estándar vertical de una ventana.

nMinPos: especifica la posición mínima de desplazamiento.

nMaxPos: especifica la posición mínima de desplazamiento.

fRedraw: especifica si el scrollbar debe ser redibujado para reflejar el cambio. Si es TRUE, el scrollbar es repintado. Si es FALSE, no.

Valor de retorno:

Si la función tiene éxito, el valor de retorno es TRUE.

Si falla, el valor de retorno es FALSE. Para conseguir más información, se puede llamar a [GetLastError](#).

Observaciones:

Una aplicación no puede llamar a la función `SetScrollRange` para ocultar un scrollbar mientras se procesa un mensaje de un scrollbar.

Si la llamada a `SetScrollRange` está justo a continuación de una llamada a la función [SetScrollPos](#), el parámetro `fRedraw` en `SetScrollPos` debe ser cero (`FALSE`) para evitar que el scrollbar se dibuje dos veces.

El rango por defecto de un scrollbar estándar es de 0 a 100. Para un control scrollbar es nulo, ambos parámetros son cero: `nMinPos` y `nMaxPos`. La diferencia de valores especificados por `nMinPos` y `nMaxPos` no puede ser mayor del valor de `MAXLONG`.

Como los mensajes que indican posiciones de scrollbar, [WM_HSCROLL](#) y [WM_VSCROLL](#), están limitados a 16 bits para el dato de la posición, las aplicaciones que confían sólo en esos mensajes para actualizar las posiciones tienen un valor máximo de 65,535 para el parámetro `nMaxPos` de la función `SetScrollRange`.

Sin embargo, como las funciones `SetScrollPos`, `SetScrollRange`, [GetScrollPos](#) y [GetScrollRange](#) soportan valores de 32-bit para la posición del scrollbar, existe un modo de sortear la barrera de 16-bit de los mensajes `WM_HSCROLL` y `WM_VSCROLL`. Ver [GetScrollPos](#) para ver una descripción de esa técnica y sus límites.

ShowWindow:

Esta función especifica cómo se mostrará la ventana.

Sintaxis:

```
BOOL ShowWindow(  
    HWND hwnd, // manipulador de ventana  
    int nCmdShow // modo de visualización de ventana  
);
```

Parámetros:

hwnd: manipulador de ventana. Identifica la ventana.

nCmdShow: modo en que se mostrará la ventana. Puede tomar uno de los siguientes [valores](#).

Valor de retorno:

Si la ventana era visible previamente, el valor de retorno es TRUE. Si la ventana estaba oculta, el valor de retorno es FALSE.

Observaciones:

Esta función debe ser llamada sólo una vez por programa con el parámetro nCmdShow de WinMain. Las siguientes llamadas deben usar uno de los valores de la tabla anterior.

Una aplicación debe llamar a ShowWindow con el parámetro nCmdShow para usar la información de arranque de aplicaciones que afecta a cómo se muestran las ventanas. Por ejemplo, si el Administrador de Programas especifica que las aplicaciones deben arrancar con su ventana principal minimizada. Las aplicaciones basadas en Win32 también usan esta información cuando se llama a ShowWindow por primera vez con el valor SW_SHOW en nCmdShow. Este comportamiento está diseñado para las siguientes situaciones:

Aplicaciones que crean su ventana principal llamando a CreateWindow con el flag WS_VISIBLE.

Aplicaciones que crean su ventana principal llamando a CreateWindow sin el flag WS_VISIBLE, y después llaman a ShowWindow con este flag activado para hacerla visible.

TranslateMessage:

Esta función traduce los mensajes de teclas virtuales a su equivalente en mensajes de carácter. Estos a su vez son reenviados a la lista de mensajes del proceso al que pertenecen, para que sean leídos la próxima vez que el proceso llame a la función [GetMessage](#) o [PeekMessage](#).

Sintaxis:

```
BOOL TranslateMessage(  
    CONST MSG *lpmsg // puntero a la estructura con el mensaje  
);
```

Parámetros:

lpmsg: puntero a una estructura [MSG](#) que contiene la información sobre el mensaje tomado de la lista de mensajes del proceso mediante la función [GetMessage](#) o [PeekMessage](#).

Valor de retorno:

Si el mensaje fue traducido, es decir, si un mensaje de carácter se envió a la lista de mensajes del proceso, retorna con TRUE; en caso contrario, retorna con FALSE.

Windows NT: la función retorna con TRUE tanto para teclas de función, como teclas de flechas, teclas de caracteres y o de dígitos.

Observaciones:

Esta función no modifica el mensaje apuntado por el parámetro lpmsg.

Las combinaciones con los mensajes [WM_KEYDOWN](#) y [WM_KEYUP](#) producen un mensaje [WM_CHAR](#) o [WM_DEADCHAR](#). Las combinaciones con [WM_SYSKEYDOWN](#) y [WM_SYSKEYUP](#) producen un mensaje [WM_SYSCHAR](#) o [WM_SYSDEADCHAR](#).

Unicamente se producirán mensajes WM_CHAR para teclas que estén mapeadas a caracteres ASCII por el driver del teclado.

Las aplicaciones que procesen los mensajes de teclas virtuales para otros propósitos, no deben llamar a la función TranslateMessage. Por ejemplo, una aplicación no debería llamar a TranslateMessage si la función [TranslateAccelerator](#) retorna con TRUE.

WindowProc:

WindowProc es una función callback definida en una aplicación que procesa los mensajes enviados a una ventana.

Sintaxis:

```
LRESULT CALLBACK WindowProc(  
    HWND hwnd,      // Manipulador de ventana  
    UINT msg,       // Mensaje  
    WPARAM wParam, // Parámetro palabra, varía  
    LPARAM lParam   // Parámetro doble palabra, varía  
);
```

Parámetros:

hwnd: es el manipulador de la ventana a la que está destinado el mensaje.

msg: es el código del mensaje.

wParam: es el parámetro de tipo palabra asociado al mensaje.

lParam: es el parámetro de tipo doble palabra asociado al mensaje.

Valor de retorno:

El valor de retorno es el resultado de procesar el mensaje y depende del tipo de mensaje enviado.

Observaciones:

WindowProc es una especie de comodín, un término que se debe sustituir por un nombre de función en la definición de una aplicación.

WinMain:

Esta función es llamada por el sistema como el punto de entrada de una aplicación Windows.

Sintaxis:

```
int WINAPI WinMain(  
    HINSTANCE hInstance,      // manipulador de la instancia  
    actual  
    HINSTANCE hPrevInstance,  // manipulador de la instancia  
    previa  
    LPSTR lpszCmdLine,        // puntero a la línea de comando  
    int nCmdShow               // estado de visualización de la  
    ventana  
    );
```

Parámetros:

hInstance: identifica la instancia actual de la aplicación.

hPrevInstance: identifica la instancia previa de la aplicación. Para aplicaciones basadas en Win32-based, este parámetro es siempre NULL.

lpszCmdLine: puntero a una cadena (terminada en cero) que especifica la línea de comando de la aplicación.

nCmdShow: especifica cómo se mostrará la ventana. Este parámetro puede tomar uno de los siguientes [valores](#).

Valor de retorno:

Si la función tiene éxito, terminando cuando recibe el mensaje [WM_QUIT](#), debe retornar con el valor de salida contenido en el parámetro wParam del mensaje. Si la función termina antes de entrar en el bucle de mensajes, debe retornar con 0.

Observaciones:

WinMain inicia un aplicación, y después realiza un bucle de recuperación y envío de mensajes que es la estructura de control de mayor nivel de toda la ejecución de la aplicación. El bucle termina cuando se recibe un mensaje WM_QUIT. En ese punto, WinMain sale de la aplicación, devolviendo el valor pasado por el mensaje en el parámetro wParam. Si el mensaje WM_QUIT fue recibido como resultado de una llamada a [PostQuitMessage](#), el valor del parámetro wParam es el valor del parámetro nExitCode de la función PostQuitMessage.



Macros

LOWORD

La macro LOWORD extrae la palabra de menor peso de un valor de 32 bits.

Definición:

```
WORD LOWORD(  
    DWORD dwValue    // valor desde el que se extrae la palabra  
);
```

Descripción:

dwValue: especifica el valor a convertir.

Valor de retorno:

El valor de retorno es la palabra de menor peso del valor especificado.

Observaciones:

La macro LOWORD está definida como sigue:

```
#define LOWORD(l) ((WORD) (l))
```

MAKEINTRESOURCE

La macro MAKEINTRESOURCE convierte un valor entero a un valor compatible con las funciones de manejo de recursos de Win32. Esta macro se usa en lugar de cadenas que contengan el nombre de un recurso.

Definición:

```
LPTSTR MAKEINTRESOURCE(  
    WORD wInteger // entero a convertir  
  
);
```

Descripción:

wInteger: especifica el valor entero a convertir.

Valor de retorno:

El valor de retorno es el valor especificado en la palabra de menor orden y cero en la de mayor orden.

Observaciones:

El valor de retorno sólo debe ser pasado a funciones de manejo de recursos de Win32, como el parámetro lpType. La macro MAKEINTRESOURCE está definida así:

```
#define MAKEINTRESOURCE(i) (LPTSTR) ((DWORD) ((WORD) (i)))
```



Mensajes

Mensaje CB_ADDSTRING

Definición:

```
CB_ADDSTRING
wParam = 0; // no usado, debe ser cero
lParam = (LPARAM) (LPCTSTR) lpsz; // puntero a la cadena de
caracteres a añadir
```

El mensaje CB_ADDSTRING se envía a un combo box para añadir una cadena a su lista. Si el combo box no posee el estilo CBS_SORT, la cadena se añade al final de la lista. En caso contrario, la cadena se insertará en el lugar correspondiente, y la lista permanecerá ordenada.

Descripción:

lpsz: valor de lParam. Apunta a la cadena terminada en cero a añadir a la lista. Si creas un combo box with con el estilo owner-drawn pero sin el estilo CBS_HASSTRINGS, el valor del parámetro lpsz se almacenará como dato del ítem (item data) en lugar de la cadena a la que en otro caso apuntaría. El item data puede ser recuperado o modificado mediante el envío de de mensajes [CB_GETITEMDATA](#) o [CB_SETITEMDATA](#).

Valor de retorno:

El valor de retorno es el índice, empezando en cero, de la cadena en la lista del combo box. Si ocurre un error, el valor de retorno es CB_ERR. Si no hay suficiente espacio para almacenar la nueva cadena, devolverá CB_ERRSPACE.

Observaciones:

Si creas un combo box owner-drawn con el estilo CBS_SORT pero sin el estilo CBS_HASSTRINGS, el mensaje [WM_COMPAREITEM](#) será enviado una o más veces a la ventana propietaria del combo box de modo que el nuevo ítem pueda ser debidamente colocado en la lista.

Para insertar una cadena en una posición concreta de la lista, usar el mensaje [CB_INSERTSTRING](#).

Mensaje CB_FINDSTRING

Definición:

```
CB_FINDSTRING
wParam = (WPARAM) indexStart;           // ítem anterior al comienzo
de la búsqueda
lParam = (LPARAM) (LPCSTR) lpszFind    // prefijo de la cadena a
buscar
```

El mensaje CB_FINDSTRING se envía para buscar en la lista de un combo box un ítem que empiece con los caracteres especificados en la cadena lpszFind.

Descripción:

indexStart: valor de wParam. Especifica el índice, comenzando en cero, del ítem anterior al primer ítem en el que se empieza a buscar. Cuando la búsqueda llega al final del listbox, continúa desde el principio hasta que llegue al ítem cuyo índice es el especificado por el parámetro indexStart. Si indexStart es -1, se busca en todo el listbox desde el principio.

lpszFind: valor de lParam. Apunta a una cadena terminada en cero que contiene el prefijo a buscar. La búsqueda es independiente del tipo, es decir que esta cadena puede contener cualquier combinación de letras mayúsculas o minúsculas.

Valor de retorno:

Si la búsqueda tuvo éxito, el valor de retorno es el índice del ítem seleccionado. Si no lo tuvo, el valor de retorno es CB_ERR y la selección actual no cambia.

Observaciones:

Si se crea un combobox con el estilo owner-drawn pero sin el estilo CBS_HASSTRINGS, entonces el mensaje CB_FINDSTRING depende de si se usó el estilo CBS_SORT. Si se usó, el sistema envía el mensaje [WM_COMPAREITEM](#) al propietario del combobox para determinar qué ítem coincide con la cadena especificada. Si no se usó, CB_FINDSTRING buscará un ítem que coincida con el valor del parámetro lpszFind.

Mensaje

CB_FINDSTRINGEXACT

Definición:

```
CB_FINDSTRING
wParam = (WPARAM) indexStart;           // ítem anterior al comienzo
de la búsqueda
lParam = (LPARAM) (LPCSTR) lpszFind    // prefijo de la cadena a
buscar
```

El mensaje CB_FINDSTRINGEXACT se envía para buscar en la lista de un combo box un ítem que coincida con la cadena lpszFind.

Descripción:

indexStart: valor de wParam. Especifica el índice, comenzando en cero, del ítem anterior al primer ítem en el que se empieza a buscar. Cuando la búsqueda llega al final del listbox, continúa desde el principio hasta que llegue al ítem cuyo índice es el especificado por el parámetro indexStart. Si indexStart es -1, se busca en todo el listbox desde el principio.

lpszFind: valor de lParam. Apunta a una cadena terminada en cero que contiene la cadena a buscar. Esta cadena puede contener un nombre de fichero completo, incluyendo la extensión. La búsqueda es independiente del tipo, es decir que esta cadena puede contener cualquier combinación de letras mayúsculas o minúsculas.

Valor de retorno:

Si la búsqueda tuvo éxito, el valor de retorno es el índice del ítem seleccionado. Si no lo tuvo, el valor de retorno es CB_ERR y la selección actual no cambia.

Observaciones:

Si se crea un combobox con el estilo owner-drawn pero sin el estilo CBS_HASSTRINGS, entonces el mensaje CB_FINDSTRINGEXACT depende de si se usó el estilo CBS_SORT. Si se usó, el sistema envía el mensaje [WM_COMPAREITEM](#) al propietario del combobox para determinar qué ítem coincide con la cadena especificada. Si no se usó, CB_FINDSTRINGEXACT buscará un ítem que coincida con el valor del parámetro lpszFind.

Mensaje CB_GETCURSEL

Definición:

```
CB_GETCURSEL  
wParam = 0;      // no se usa; debe ser cero  
lParam = 0;      // no se usa; debe ser cero
```

Se envía un mensaje CB_GETCURSEL para recuperar el índice del ítem actualmente seleccionado, si hay alguno, en un listbox asociado a un control combobox

Descripción:

Este mensaje no tiene parámetros.

Valor de retorno:

El valor de retorno es el índice (empezando en cero) del ítem actualmente seleccionado. Si no hay selección, el valor de retorno es CB_ERR.

Mensaje CB_GETLBTEXT

Definición:

CB_GETLBTEXT

```
wParam = (WPARAM) index;           // item index  
lParam = (LPARAM) (LPCSTR) lpzBuffer; // address of buffer
```

El mensaje CB_GETLBTEXT se envía a un control combobox para recuperar una cadena desde la lista asociada a un combo box.

Descripción:

index: valor de wParam. Indica el índice, comenzando en cero, de la cadena a recuperar.

lpzBuffer: valor de lParam. Apunta al buffer que recibirá la cadena. El buffer debe disponer de espacio suficiente para la cadena y el carácter nulo terminador. Se puede usar el mensaje [CB_GETLBTEXTLEN](#) antes para obtener la longitud en bytes de la cadena.

Valor de retorno:

El valor de retorno es la longitud de la cadena en bytes, sin contar el carácter nulo de terminación. Si el parámetro index no especifica un valor de índice válido, el valor de retorno será CB_ERR.

Observaciones:

Si se crea un combobox con el estilo owner-drawn pero sin el estilo CBS_HASSTRINGS, el buffer apuntado por el parámetro lpzBuffer del mensaje recibirá el valor de 32 bits asociado con el ítem (el item data).

Mensaje CB_GETLBTEXTLEN

Definición:

```
CB_GETLBTEXTLEN
wParam = (WPARAM) index;    // índice del ítem
lParam = 0;                  // no se usa; debe ser cero
```

Una aplicación envía un mensaje CB_GETLBTEXTLEN para recuperar la longitud en caracteres de una cadena en una lista de un combobox.

Descripción:

index: valor de wParam. Especifica el índice, comenzando en cero, de la cadena a recuperar.

Valor de retorno:

El valor de retorno es la longitud de la cadena, en caracteres, excluyendo el terminador nulo. Bajo algunas condiciones, éste valor puede ser más grande que la longitud del texto. Para mayor información, ver la sección de observaciones.

Si el parámetro index no especifica un índice válido, el valor de retorno es CB_ERR.

Observaciones:

Bajo algunas condiciones, éste valor puede ser más grande que la longitud del texto. Esto ocurre con ciertas mezclas de ANSI y Unicode, y es debido a que el sistema operativo permite la posible existencia de juegos de caracteres de doble byte (DBCS) en el texto. El valor de retorno, sin embargo, será siempre por lo menos tan largo como el tamaño actual del texto; y puede ser usado siempre como guía para obtener memoria para el buffer. Este comportamiento puede ocurrir cuando una aplicación use funciones ANSI y diálogos comunes, junto con Unicode.

Para obtener la longitud exacta del texto, usar los mensajes [WM_GETTEXT](#), [LB_GETTEXT](#) o [CB_GETLBTEXT](#), o la función [GetWindowText](#).

Mensaje CB_SELECTSTRING

Definición:

```
CB_SELECTSTRING
wParam = (WPARAM) indexStart;           // ítem a partir de que se
hará la primera selección
lParam = (LPARAM) (LPCSTR) lpszSelect; // dirección de la cadena o
prefijo de cadena
```

Al enviar el mensaje CB_SELECTSTRING se buscará en la lista de un combo box un ítem que empiece con los mismos caracteres que la cadena especificada. Si se encuentra una cadena que coincida, será seleccionada y copiada al control edit.

Descripción:

indexStart: valor de wParam. Especifica el índice, comenzando en cero, del ítem anterior al primer ítem en el que se empieza a buscar. Cuando la búsqueda llega al final del listbox, continúa desde el principio hasta que llegue al ítem cuyo índice es el especificado por el parámetro indexStart. Si indexStart es -1, se busca en todo el listbox desde el principio.

lpszSelect: valor de lParam. Apunta a una cadena terminada en cero que contiene el prefijo a buscar. La búsqueda es independiente del tipo, es decir que esta cadena puede contener cualquier combinación de letras mayúsculas o minúsculas.

Valor de retorno:

Si la búsqueda tuvo éxito, el valor de retorno es el índice del ítem seleccionado. Si no lo tuvo, el valor de retorno es CB_ERR y la selección actual no cambia.

Observaciones:

Un ítem está seleccionado sólo si sus caracteres iniciales coinciden con la cadena especificada en el parámetro lpszSelect.

Si se crea un combobox con el estilo owner-drawn pero sin el estilo CBS_HASSTRINGS, entonces el mensaje CB_SELECTSTRING depende de si se usó el estilo CBS_SORT. Si se usó, el sistema envía el mensaje [WM_COMPAREITEM](#) al propietario del combobox para determinar qué ítem coincide con la cadena especificada. Si no se usó, CB_SELECTSTRING intentará comparar el valor DWORD con el valor del parámetro lpszSelect.

Mensaje EM_LIMITTEXT

Definición:

```
EM_LIMITTEXT  
wParam = (WPARAM) cchMax;    // longitud del texto, en caracteres  
lParam = 0;                  // no se usa; debe ser cero
```

Una aplicación puede enviar un mensaje EM_LIMITTEXT para limitar la cantidad de texto que el usuario puede introducir en un control edit.

Descripción:

cchMax: valor de wParam. Especifica el número máximo de caracteres que el usuario puede introducir. Si este parámetro es cero, la longitud del texto se limita a 0x7FFFFFFE caracteres para controles edit de una sola línea o a 0xFFFFFFFF para controles edit multilínea.

Valor de retorno:

Este mensaje no devuelve valores.

Observaciones:

El mensaje EM_LIMITTEXT limita sólo el texto que el usuario puede introducir. Eso no afecta a ningún texto que ya esté en el control edit cuando el mensaje es enviado, tampoco afecta a la longitud del texto copiado al control edit mediante el mensaje [WM_SETTEXT](#). Si una aplicación usa el mensaje WM_SETTEXT para colocar más texto en un control edit del que se limitó mediante el mensaje EM_LIMITTEXT, el usuario podrá editar el contenido completo del control edit..

El límite de texto por defecto que un usuario puede introducir en un control edit es de 30000 caracteres.

Mensaje LB_ADDSTRING

Definición:

```
LB_ADDSTRING  
wParam = 0; // no se usa; debe ser  
cero  
lParam = (LPARAM) (LPCTSTR); // dirección de la cadena a añadir
```

Una aplicación envía un mensaje LB_ADDSTRING para añadir una cadena a un listbox. Si el listbox no tiene el estilo LBS_SORT, la cadena se añade al final de la lista. En otro caso, la cadena se inserta en la lista de modo que permanezca ordenada.

Descripción:

lpsz: valor de lParam. Apunta a la cadena terminada con cero que será añadida.

Si se crea un listbox con el estilo owner-drawn pero sin el estilo LBS_HASSTRINGS, el valor del parámetro lpsz es almacenado como un "item data" en lugar de la cadena a la que de otro modo apuntaría. Se pueden usar los mensajes [LB_GETITEMDATA](#) y [LB_SETITEMDATA](#) para recuperar o modificar el "item data".

Valor de retorno:

El valor de retorno es el índice de la cadena en el listbox, los valores del índice empiezan en cero. Si ocurre un error, el valor de retorno es LB_ERR. Si no hay suficiente espacio para almacenar la nueva cadena, el valor de retorno será LB_ERRSPACE.

Observaciones:

Si se crea un listbox owner-drawn con el estilo LBS_SORT pero sin el estilo LBS_HASSTRINGS, el sistema envía el mensaje [WM_COMPAREITEM](#) una o más veces al propietario del listbox para colocar el nuevo elemento en el lugar adecuado del listbox.

Mensaje LB_GETCURSEL

Definición:

```
LB_GETCURSEL  
wParam = 0;      // no se usa; debe ser cero  
lParam = 0;      // no se usa; debe ser cero
```

Se envía un mensaje LB_GETCURSEL para recuperar el índice del ítem actualmente seleccionado, si hay alguno, en un listbox de selección sencilla. Para listbox de selección múltiple, este mensaje recupera el índice del primer ítem seleccionado, el ítem base, si hay alguno.

Descripción:

Este mensaje no tiene parámetros.

Valor de retorno:

El valor de retorno es el índice (empezando en cero) del ítem actualmente seleccionado o el ítem base de una selección múltiple.

Si no hay selección, el valor de retorno es LB_ERR.

Observaciones:

Usar el mensaje [LB_GETCARETINDEX](#) para recuperar el índice del ítem que tiene el rectángulo de foco en un listbox de selección múltiple.

Mensaje LB_GETTEXT

Definición:

```
LB_GETTEXT  
wParam = (WPARAM) index;           // índice del ítem  
lParam = (LPARAM) (LPCTSTR) lpszBuffer; // dirección del buffer
```

Una aplicación envía un mensaje LB_GETTEXT para recuperar una cadena desde un listbox.

Descripción:

index: valor de wParam. Especifica el índice, comenzando en cero, de la cadena a recuperar.

Sólo en Windows 95: el parámetro wParam está limitado a valores de 16 bits. Esto significa que los listbox no pueden contener más de 32767 ítems. Aunque el número de ítems está restringido, el tamaño total en bytes de los ítems en un listbox está también limitado por la memoria disponible.

lpszBuffer: valor de lParam. Apunta al buffer que recibirá la cadena. El buffer deberá tener el suficiente espacio para la cadena u el carácter nulo de terminación. Se puede enviar un mensaje [LB_GETTEXTLEN](#) antes del mensaje LB_GETTEXT para averiguar la longitud, en caracteres, de la cadena.

Valor de retorno:

El valor de retorno es la longitud de la cadena, en caracteres, excluyendo el terminador nulo. Si el parámetro index no especifica un índice válido, el valor de retorno es LB_ERR.

Observaciones:

Si se crea un listbox con el estilo owner-drawn pero sin el estilo LBS_HASSTRINGS, el buffer apuntado por el parámetro lpszBuffer recibirá el valor de 32 bits asociado con el ítem, (el item data).

Mensaje LB_GETTEXTLEN

Definición:

```
LB_GETTEXTLEN  
wParam = (WPARAM) index;    // índice del ítem  
lParam = 0;                  // no se usa; debe ser cero
```

Una aplicación envía un mensaje LB_GETTEXTLEN para recuperar la longitud de una cadena de un listbox.

Descripción:

index: valor de wParam. Especifica el índice, comenzando en cero, de la cadena a recuperar.

Sólo en Windows 95: el parámetro wParam está limitado a valores de 16 bits. Esto significa que los listbox no pueden contener más de 32767 ítems. Aunque el número de ítems está restringido, el tamaño total en bytes de los ítems en un listbox está también limitado por la memoria disponible.

Valor de retorno:

El valor de retorno es la longitud de la cadena, en caracteres, excluyendo el terminador nulo. Bajo algunas condiciones, éste valor puede ser más grande que la longitud del texto. Para mayor información, ver la sección de observaciones.

Si el parámetro index no especifica un índice válido, el valor de retorno es LB_ERR.

Observaciones:

Bajo algunas condiciones, éste valor puede ser más grande que la longitud del texto. Esto ocurre con ciertas mezclas de ANSI y Unicode, y es debido a que el sistema operativo permite la posible existencia de juegos de caracteres de doble byte (DBCS) en el texto. El valor de retorno, sin embargo, será siempre por lo menos tan largo como el tamaño actual del texto; y puede ser usado siempre como guía para obtener memoria para el buffer. Este comportamiento puede ocurrir cuando una aplicación use funciones ANSI y diálogos comunes, junto con Unicode.

Para obtener la longitud exacta del texto, usar los mensajes [WM_GETTEXT](#), [LB_GETTEXT](#) o [CB_GETLBTEXT](#), o la función [GetWindowText](#).

Mensaje LB_SELECTSTRING

Definición:

```
LB_SELECTSTRING
wParam = (WPARAM) indexStart;           // ítem anterior al de
comenzar a buscar
lParam = (LPARAM) (LPCTSTR) lpszFind;   // dirección de la cadena a
buscar
```

Una aplicación envía un mensaje LB_SELECTSTRING para buscar en un listbox un ítem que empiece con los caracteres de una cadena especificada. Si se encuentra un ítem, éste es seleccionado.

Descripción:

indexStart: valor de wParam. Especifica el índice, comenzando en cero, del ítem anterior al primer ítem en el que se empieza a buscar. Cuando la búsqueda llega al final del listbox, continúa desde el principio hasta que llegue al ítem cuyo índice es el especificado por el parámetro indexStart. Si indexStart es -1, se busca en todo el listbox desde el principio.

Sólo en Windows 95: el parámetro wParam está limitado a valores de 16 bits. Esto significa que los listbox no pueden contener más de 32767 ítems. Aunque el número de ítems está restringido, el tamaño total en bytes de los ítems en un listbox está también limitado por la memoria disponible.

lpszFind: valor de lParam. Apunta a una cadena terminada en cero que contiene el prefijo a buscar. La búsqueda es independiente del tipo, es decir que esta cadena puede contener cualquier combinación de letras mayúsculas o minúsculas.

Valor de retorno:

Si la búsqueda tuvo éxito, el valor de retorno es el índice del ítem seleccionado. Si no lo tuvo, el valor de retorno es LB_ERR y la selección actual no cambia.

Observaciones:

El contenido del listbox es desplazado, si es necesario, para mostrar el ítem seleccionado.

No usar este mensaje con un listbox que tenga el estilo LBS_MULTIPLESEL.

Un ítem está seleccionado sólo si sus caracteres iniciales coinciden con la cadena especificada en el parámetro lpszFind.

Si se crea un listbox con el estilo owner-drawn pero sin el estilo LBS_HASSTRINGS, este mensaje devuelve el índice del ítem cuyo valor largo (suministrado por el parámetro lParam del mensaje [LB_ADDSTRING](#) o [LB_INSERTSTRING](#)) coincida con el valor suministrado en el parámetro lParam de LB_SELECTSTRING.

Mensaje SBM_GETPOS

Definición:

```
SBM_GETPOS  
wParam = 0; // no usado, debe ser cero  
lParam = 0; // no usado, debe ser cero
```

Una aplicación envía el mensaje SBM_GETPOS para recuperar la posición actual de la caja de desplazamiento de un control scrollbar. La posición actual es un valor relativo que depende del valor actual del rango de desplazamiento. Por ejemplo, si el rango es de 0 a 100 y la caja de desplazamiento esta en la mitad de la barra, la posición actual es 50.

Descripción:

Este mensaje no tiene parámetros.

Valor de retorno:

Si el mensaje tiene éxito, el valor de retorno es la posición actual de la caja de desplazamiento en el scrollbar.

Mensaje SBM_GETRANGE

Definición:

```
SBM_GETRANGE  
wParam = (WPARAM) (LPINT) lpnMinPos; // posición mínima  
lParam = (LPARAM) (LPINT) lpnMaxPos; // posición máxima
```

Una aplicación envía el mensaje SBM_GETRANGE a un control scrollbar para recuperar los valores de las posiciones mínima y máxima del control.

Descripción:

lpnMinPos: puntero al valor que recibirá la posición de desplazamiento mínima.

lpnMaxPos: puntero al valor que recibirá la posición de desplazamiento máxima.

Valor de retorno:

Este mensaje no devuelve ningún valor.

Mensaje SBM_GETSCROLLINFO

Definición:

[Nuevo - Windows NT] (No puede usarse en win32s, sí en Win95 y Win NT)

```
SBM_GETSCROLLINFO
wParam = 0; // no usado, debe ser cero
lParam = (LPARAM) (LPSCROLLINFO) lpsi; // parámetros de scrollbar
```

Una aplicación envía este mensaje para recuperar los parámetros de un scrollbar, incluyendo las posiciones mínima y máxima de desplazamiento, el tamaño de la página y la posición de la caja de desplazamiento (thumb).

Descripción:

lpsi: valor de lParam. Apunta a una estructura [SCROLLINFO](#) cuyo miembro fMask, especifica los parámetros del scrollbar a recuperar. Antes de retornar, la función copia los parámetros especificados a los miembros apropiados de la estructura.

El miembro fMask puede ser una combinación de los siguientes valores:

Valor	Significado
SIF_ALL	Combinación de SIF_PAGE, SIF_POS y SIF_RANGE.
SIF_PAGE	Copia el valor de la página de desplazamiento al miembro nPage de la estructura SCROLLINFO apuntada por lpsi.
SIF_POS	Copia el valor de la posición de desplazamiento al miembro nPos de la estructura SCROLLINFO apuntada por lpsi.
SIF_RANGE	Copia el rango de desplazamiento a los miembros nMin y nMax de la estructura SCROLLINFO apuntada por lpsi.

Valor de retorno:

Si el mensaje recupera algún valor, el valor de retorno es TRUE; en caso contrario será FALSE.

Mensaje SBM_SETPOS

Definición:

```
SBM_SETPOS  
wParam = (WPARAM) nPos;           // nueva posición de la caja de  
desplazamiento  
lParam = (LPARAM) (BOOL) fRedraw; // flag de redibujado
```

Una aplicación envía el mensaje SBM_SETPOS a un control scrollbar para cambiar la posición de la caja de desplazamiento (thumb) y, se requiere, repintar el scrollbar para mostrar la nueva posición de la caja de desplazamiento.

Descripción:

nPos: especifica la nueva posición de la caja de desplazamiento. Debe estar dentro de los límites del rango de desplazamiento.

fRedraw: especifica si el scrollbar debe ser redibujado para mostrar la nueva posición de la caja de desplazamiento. Si este parámetro es TRUE, el scrollbar es repintado. Si es FALSE, no.

Valor de retorno:

Si la posición de la caja ha cambiado, el valor de retorno es el valor anterior de la posición, en otro caso es cero.

Observaciones:

Si el control scrollbar es redibujado por llamadas posteriores a otras funciones o mensajes, es corriente usar FALSE como valor del parámetro fRedraw .

Mensaje SBM_SETRANGE

Definición:

```
SBM_SETRANGE  
wParam = (WPARAM) nMinPos; // posición de desplazamiento mínima  
lParam = (LPARAM) nMaxPos; // posición de desplazamiento máxima
```

Una aplicación envía el mensaje SBM_SETRANGE a un control scrollbar para modificar los valores de las posiciones mínima y máxima del control.

Descripción:

lpnMinPos: especifica la posición de desplazamiento mínima.

lpnMaxPos: especifica la posición de desplazamiento máxima.

Valor de retorno:

Si la posición de la caja de desplazamiento cambia, el valor de retorno es la posición previa de la caja, en otro caso será cero.

Observaciones:

Los valores de las posiciones por defecto mínimo y máximo son cero. La diferencia entre los valores especificados por los parámetros nMinPos y nMaxPos no debe ser mayor que el valor de MAXLONG.

Si los valores mínimo y máximo son iguales, el control scrollbar se oculta y se desactiva.

Mensaje

SBM_SETSCROLLINFO

Definición:

[Nuevo - Windows NT] (No puede usarse en win32s, sí en Win95 y Win NT)

```
SBM_SETSCROLLINFO
wParam = (WPARAM) fRedraw;           // flag de redibujado
lParam = (LPARAM) (LPSCROLLINFO) lpsi; // parámetros de scrollbar
```

Una aplicación envía este mensaje para modificar los parámetros de un scrollbar, incluyendo las posiciones mínima y máxima de desplazamiento, el tamaño de la página y la posición de la caja de desplazamiento (thumb).

Descripción:

fRedraw: valor de wParam. Especifica si el scrollbar será redibujado para reflejar la nueva posición de la caja de desplazamiento. Si este parámetro es TRUE, el scrollbar es redibujado. Si el FALSE, no.

lpsi: valor de lParam. Apunta a una estructura [SCROLLINFO](#) cuyo miembro fMask, especifica los parámetros del scrollbar a modificar.

El miembro fMask puede ser una combinación de los siguientes valores:

Valor	Significado
SIF_ALL	Combinación de SIF_PAGE, SIF_POS y SIF_RANGE.
SIF_PAGE	Modifica el valor de la página de desplazamiento al miembro nPage de la estructura SCROLLINFO apuntada por lpsi.
SIF_POS	Modifica el valor de la posición de desplazamiento al miembro nPos de la estructura SCROLLINFO apuntada por lpsi.
SIF_RANGE	Modifica el rango de desplazamiento a los miembros nMin y nMax de la estructura SCROLLINFO apuntada por lpsi.

Valor de retorno:

El valor de retorno es la posición actual de la caja de desplazamiento.

Mensaje WM_CHAR

Definición:

```
WM_CHAR  
chCharCode = (TCHAR) wParam; // código de carácter  
lKeyData = lParam; // dato de la tecla
```

El mensaje WM_CHAR se envía a la ventana con el foco del teclado cuando un mensaje [WM_KEYDOWN](#) es traducido por la función [TranslateMessage](#). WM_CHAR contiene el código del carácter de la tecla pulsada.

Descripción:

chCharCode: valor de wParam. Especifica el código de carácter de la tecla.

lKeyData: valor de lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Observaciones:

Puesto que no es necesario que exista una correspondencia uno a uno entre las teclas pulsadas y los mensaje de carácter generados, la información en la palabra alta del parámetro lKeyData normalmente no es útil para las aplicaciones. Esta información se aplica sólo al mensaje [WM_KEYDOWN](#) que precede al mensaje WM_CHAR más recientemente enviado.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_COMMAND

Definición:

```
WM_COMMAND
wNotifyCode = HIWORD(wParam); // código de notificación
wID = LOWORD(wParam);         // identificador de ítem, control, o
acelerador
hwndCtl = (HWND) lParam;      // manipulador de control
```

El mensaje WM_COMMAND es enviado cuando el usuario selecciona un comando de un ítem de un menú, cuando un control envía un mensaje de notificación a su ventana padre, o cuando una pulsación de un acelerador es traducida.

Descripción:

wNotifyCode: valor de la palabra de mayor peso de wParam. Especifica el código de notificación si el mensaje proviene de un control. Si el mensaje proviene de un acelerador, éste parámetro es 1. Si el mensaje proviene de un menú, éste parámetro es 0.

wID: valor de la palabra de menor peso de wParam. Especifica el identificador del ítem de menú, control, o acelerador.

hwndCtl: valor de lParam. Identifica el control que envía el mensaje, si el mensaje proviene de un control. En otro caso, éste parámetro es NULL.

Valor de retorno:

Si una aplicación procesa éste mensaje, debe retornar con cero.

Observaciones:

Los aceleradores de teclado que seleccionan ítems del menú de sistema son traducidos a mensajes [WM_SYSCOMMAND](#).

Si ocurre un acelerador de teclado que corresponde a un ítem de menú cuando la ventana a la que pertenece el menú está minimizada, no se enviará el mensaje WM_COMMAND. Sin embargo, si ocurre un acelerador de teclado que no corresponde con ningún ítem de menú o de menú de sistema, el mensaje WM_COMMAND se enviará, aunque la ventana esté minimizada.

Si una aplicación habilita un separador de menú, el sistema envía un mensaje WM_COMMAND con la palabra de menor peso de wParam puesta a cero cuando el usuario selecciona el separador.

Mensaje WM_CREATE

Definición:

```
WM_CREATE  
lpcs = (LPCREATESTRUCT) lParam; // estructura con los datos de  
creación
```

El mensaje WM_CREATE se envía cuando una aplicación pide que sea creada una ventana mediante una llamada a la función [CreateWindowEx](#) o [CreateWindow](#). El procedimiento de ventana de la nueva ventana recibe éste mensaje después de que la ventana ha sido creada, pero antes de que sea visible. El mensaje es enviado antes de que la función CreateWindowEx o CreateWindow retorne.

Descripción:

lParam: valor de lParam. Apunta a una estructura [CREATESTRUCT](#) que contiene información sobre la ventana que se está siendo creada. Los miembros de CREATESTRUCT son idénticos a los parámetros de la función CreateWindowEx.

Valor de retorno:

Si una aplicación procesa este mensaje, debe retornar con 0 para que continúe la creación de la ventana. Si la aplicación retorna con -1, la ventana será destruida y la función CreateWindowEx o CreateWindow devolverá un manipulador NULL.

Mensaje WM_DESTROY

Definición:

WM_DESTROY

El mensaje WM_DESTROY se envía a una ventana que va a ser destruída. Es enviado por el procedimiento de ventana de la ventana que se destruirá antes de que sea retirada de la pantalla.

Este mensaje es enviado primero a la ventana a destruir y después a las ventanas hijas (si existen) que también serán destruidas. Durante el procesamiento del mensaje, se da por hecho que estas ventanas aún existen.

Descripción:

Este mensaje no tiene parámetros.

Valor de retorno:

Si una aplicación procesa este mensaje debe retronar cero.

Observaciones:

Si la ventana a destruir es parte de la cadena del visor del portapapeles (hecho mediante la llamada a la función [SetClipboardViewer](#)), al ventana debe eliminarse a sí misma de la cadena procesando la función [ChangeClipboardChain](#) antes de volver del mensaje WM_DESTROY.

Mensaje

WM_GETMINMAXINFO

Definición:

```
WM_GETMINMAXINFO  
lpmmi = (LPMINMAXINFO) lParam; // dirección de la estructura
```

El mensaje WM_GETMINMAXINFO es enviado a una ventana cuando su tamaño o posición va a cambiar. Una aplicación puede usar este mensaje para ignorar el tamaño o posición por defecto de la ventana maximizada, o su mínimo o máximo tamaño de "tracking".

Descripción:

lpmmi: valor del parámetro lParam. Apunta a una estructura MINMAXINFO que contiene la posición y las dimensiones máximas por defecto para la ventana maximizada, y la posición y las dimensiones máximas y mínimas por defecto para el tamaño de "tracking". Una aplicación puede ignorar los valores por defecto sobrescribiendo los valores de esta estructura.

Valor de retorno:

Si una aplicación procesa este mensaje, debe retornar cero.

Observaciones:

El máximo tamaño de "tracking" es el tamaño de la ventana más grande que se puede producir usando el borde para cambiar el tamaño de la ventana. El tamaño mínimo de "tracking" es la ventana más pequeña que se puede producir usando los bordes para cambiar el tamaño de la ventana.

Mensaje WM_GETTEXT

Definición:

```
WM_GETTEXT
wParam = (WPARAM) cchTextMax; // número de caracteres a copiar
lParam = (LPARAM) lpszText;   // dirección del buffer de texto
```

Una aplicación envía un mensaje WM_GETTEXT para copiar el texto que corresponde a una ventana o control en un buffer suministrado por el usuario.

Descripción:

cchTextMax: valor de wParam. Especifica el número máximo de caracteres a copiar, incluyendo el carácter nulo terminador de cadena.

lpszText: valor de lParam. Puntero al buffer que recibirá el texto.

Valor de retorno:

El valor de retorno es el número de caracteres copiados.

Acción por defecto:

La función [DefWindowProc](#) copia el texto asociado con la ventana en el buffer especificado y devuelve el número de caracteres leídos.

Observaciones:

Para un control edit, el texto a copiar es el contenido del control edit. Para un combo box, el texto es el contenido de la porción control edit del combo box. Para un button, el texto es el nombre del botón. Para otras ventanas, el texto es el título de la ventana. Para copiar el texto de un ítem de un list box, debe usarse el mensaje [LB_GETTEXT](#).

Cuando el mensaje WM_GETTEXT se envía a un control static con el estilo SS_ICON, el manipulador del icono se devolverá en los primeros cuatro bytes del buffer apuntado por by lpszText. Esto es verdad sólo si el mensaje [WM_SETTEXT](#) fue usado para asignar el icono.

Mensaje

WM_GETTEXTLENGTH

Definición:

```
WM_GETTEXTLENGTH
wParam = 0; // no se usa; debe ser cero
lParam = 0; // no se usa; debe ser cero
```

El mensaje WM_GETTEXTLENGTH se envía para determinar la longitud en caracteres del texto asociado a una ventana. La longitud no incluye el carácter nulo de terminación.

Descripción:

Este mensaje no tiene parámetros

Valor de retorno:

El valor de retorno es la longitud de la cadena, en caracteres, excluyendo el terminador nulo.

Acción por defecto:

La función [DefWindowProc](#) devuelve la longitud en caracteres del texto. Bajo algunas condiciones, ese valor puede ser mayor que la longitud real del texto. Para mayor detalle, ver la sección de observaciones.

Observaciones:

Para un control edit, el texto a copiar es el contenido del control edit. Para un combo box, el texto es el contenido del control edit (o del control static) asociado al combo box. Para un button, el texto es el nombre del button. Para otras ventanas, es el título de la ventana. Para determinar la longitud de un ítem en un list box, se puede usar el mensaje LB_GETTEXTLEN.

Bajo algunas condiciones, la función DefWindowProc puede retornar un valor más grande que la longitud del texto. Esto ocurre con ciertas mezclas de ANSI y Unicode, y es debido a que el sistema operativo permite la posible existencia de juegos de caracteres de doble byte (DBCS) en el texto. El valor de retorno, sin embargo, será siempre por lo menos tan largo como el tamaño actual del texto; y puede ser usado siempre como guía para obtener memoria para el buffer. Este comportamiento puede ocurrir cuando una aplicación use funciones ANSI y diálogos comunes, junto con Unicode.

Para obtener la longitud exacta del texto, usar los mensajes [WM_GETTEXT](#), [LB_GETTEXT](#) o [CB_GETLBTEXT](#), o la función [GetWindowText](#).

Mensaje WM_HSCROLL

Definición:

```
WM_HSCROLL
nScrollCode = (int) LOWORD(wParam); // valor de scrollbar
nPos = (short int) HIWORD(wParam); // posición de la caja de
desplazamiento
hwndScrollBar = (HWND) lParam; // manipulador del scrollbar
```

El mensaje WM_HSCROLL es enviado a una ventana cuando ocurre un evento de scroll en su scrollbar horizontal estándar. Este mensaje también se envía a la ventana propietaria de un control scrollbar horizontal cuando un evento ocurre en ese control.

Descripción:

nScrollCode: valor de la palabra de menor peso de wParam. Especifica un valor de scrollbar que indica una petición de desplazamiento del usuario. Este parámetro puede tomar uno de los siguientes valores:

Valor	Significado
SB_BOTTOM	Desplazamiento total a la derecha.
SB_ENDSCROLL	Termina el desplazamiento.
SB_LINELEFT	Desplaza a la izquierda una unidad.
SB_LINERIGHT	Desplaza a la derecha una unidad.
SB_PAGELEFT	Desplaza a la izquierda en el ancho de la ventana.
SB_PAGERIGHT	Desplaza a la derecha en el ancho de la ventana.
SB_THUMBPOSITION	Desplaza a una posición absoluta. La posición actual se especifica mediante el parámetro nPos.
SB_THUMBTRACK	Arrastra la caja de desplazamiento a la posición especificada. La posición actual se especifica mediante el parámetro nPos.
SB_TOP	Desplazamiento total a la izquierda.

nPos: valor de la palabra de mayor peso de wParam. Especifica la posición actual de la caja de desplazamiento si el parámetro nScrollCode es SB_THUMBPOSITION o SB_THUMBTRACK; en cualquier otro caso, nPos no se usa.

hwndScrollBar: valor de lParam. Identifica el control si el mensaje WM_HSCROLL fue enviado por un control scrollbar. Si el mensaje WM_HSCROLL fue enviado por un scrollbar estándar, hwndScrollBar no se usa.

Valor de retorno:

Si una aplicación procesa éste mensaje, debe retornar con cero.

Observaciones:

El mensaje de notificación SB_THUMBTRACK se usa normalmente por aplicaciones que proporcionan una respuesta mientras el usuario arrastra la caja de desplazamiento.

Su una aplicación desplaza el contenido de una ventana, debe también actualizar la posición de la caja de desplazamiento usando la función [SetScrollPos](#).

Como el mensaje WM_HSCROLL proporciona sólo 16 bits para el dato de la posición, las aplicaciones que confían sólo en WM_HSCROLL (y WM_VSCROLL) para actualizar las posiciones tienen un valor máximo de 65,535.

Sin embargo, como las funciones [SetScrollPos](#), [SetScrollRange](#), [GetScrollPos](#) y [GetScrollRange](#) soportan valores de 32-bit para la posición del scrollbar, existe un modo de sortear la barrera de 16-bit de los mensajes WM_HSCROLL y WM_VSCROLL. Ver [GetScrollPos](#) para ver una descripción de esa técnica y sus límites.

Mensaje WM_INITDIALOG

Definición:

```
WM_INITDIALOG  
hwndFocus = (HWND) wParam; // manipulador del control que recibe  
el foco  
lInitParam = lParam;      // parámetro de inicialización
```

El mensaje WM_INITDIALOG se envía al procedimiento de diálogo inmediatamente antes de que el diálogo sea mostrado. El procedimiento de diálogo normalmente usa este mensaje para inicializar los controles y cumplimentar cualquier trabajo de inicialización que afecte a la apariencia del cuadro de diálogo.

Descripción:

hwndFocus: valor de wParam. Identifica el control que recibe el foco del teclado por defecto. Windows asigna el foco de teclado por defecto sólo si el procedimiento de ventana retorna con TRUE.

lInitParam: valor de lParam. Especifica datos de inicialización adicionales. Estos datos son pasados a Windows como el parámetro lParamInit en una llamada a las funciones [CreateDialogIndirectParam](#), [CreateDialogParam](#), [DialogBoxIndirectParam](#) o [DialogBoxParam](#) usadas para crear el cuadro de diálogo. Este parámetro es cero si se usa cualquier otra función para crear el cuadro de diálogo.

Valor de retorno:

El procedimiento de diálogo retorna con TRUE para indicar a Windows que debe asignar el foco del teclado al control dado por hwndFocus. En otro caso debe retornar FALSE para evitar que Windows asigne el foco de teclado por defecto.

Comentarios:

El control que recibe el foco de teclado por defecto es siempre el primer control del diálogo que es visible, no está desactivado y tiene el estilo WS_TABSTOP. Entonces el procedimiento de diálogo retorna TRUE, y Windows comprueba el control para asegurarse de que el procedimiento no lo ha desactivado. Si ha sido desactivado, Windows asigna el foco del teclado al siguiente control que es visible, no está desactivado y tiene el estilo WS_TABSTOP.

Una aplicación puede volver con FALSE sólo si ha asignado el foco del teclado a uno de los controles del cuadro de diálogo.

Mensaje WM_KEYDOWN

Definición:

```
WM_KEYDOWN
nVirtKey = (int) wParam;    // código de tecla virtual
lKeyData = lParam;        // dato de tecla
```

El mensaje WM_KEYDOWN es enviado a la ventana que tiene el foco del teclado cuando se pulsa una tecla que no es de sistema. Una tecla no es de sistema cuando cuando la tecla ALT no está presionada.

Descripción:

nVirtKey: valor del parámetro wParam. Indica el código de la tecla virtual no de sistema.

lKeyData: valor del parámetro lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Acción por defecto:

Si se pulsó la tecla F10, la función DefWindowProc activa un flag interno. Cuando DefWindowProc recibe el mensaje [WM_KEYUP](#), la función comprueba si ese flag interno está activo y, si lo está, envía un mensaje [WM_SYSCOMMAND](#) a la ventana de mayor nivel. Como parámetro wParam del mensaje se usa SC_KEYMENU.

Observaciones:

Puesto que no es necesario que exista una correspondencia uno a uno entre las teclas pulsadas y los mensajes de carácter generados, la información en la palabra alta del parámetro lKeyData normalmente no es útil para las aplicaciones. Esta información se aplica sólo al mensaje WM_KEYDOWN que precede al mensaje WM_CHAR más recientemente enviado.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_KEYUP

Definición:

```
WM_KEYUP  
nVirtKey = (int) wParam;    // código de tecla virtual  
lKeyData = lParam;        // dato de tecla
```

El mensaje WM_KEYUP es enviado a la ventana que tiene el foco del teclado cuando se libera una tecla que no es de sistema. Una tecla no es de sistema cuando fue pulsada y la tecla ALT no lo estaba o la tecla fue presionada cuando la ventana no tenía el foco del teclado.

Descripción:

nVirtKey: valor del parámetro wParam. Indica el código de la tecla virtual no de sistema.

lKeyData: valor del parámetro lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT y CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Acción por defecto:

La función DefWindowProc envía un mensaje [WM_SYSCOMMAND](#) a la ventana de mayor nivel si la tecla F10 o la tecla ALT fueron liberadas. Como parámetro wParam parameter del mensaje se usa SC_KEYMENU.

Observaciones:

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_NCCREATE

Definición:

```
WM_NCCREATE  
lpcs = (LPCREATESTRUCT) lParam; // datos de inicialización
```

El mensaje WM_NCCREATE se envía previamente al mensaje [WM_CREATE](#) cuando una ventana es creada por primera vez.

Descripción:

lpcs: valor del parámetro wParam. Apunta a una estructura [CREATESTRUCT](#) para la ventana.

Valor de retorno:

Si una aplicación procesa este mensaje debe retornar con TRUE para que continúe la creación de la ventana. Si la aplicación retorna con FALSE, las funciones [CreateWindow](#) o [CreateWindowEx](#) retornarán con un manipulador NULL.

Acción por defecto:

La función [DefWindowProc](#) devuelve TRUE para este mensaje.

Mensaje WM_NCPAINT

Definición:

```
WM_NCPAINT  
hrgn = (HRGN) wParam; // manipulador a la region a actualizar
```

Una aplicación envía un mensaje WM_NCPAINT a una ventana cuando su marco debe ser pintado.

Descripción:

hrgn: valor de wParam. Identifica la región de la ventana a actualizar. La región es enmascarada al marco de la ventana.

Valor de retorno:

Una aplicación debe retornar con cero si procesa este mensaje.

Acción por defecto:

La función [DefWindowProc](#) pinta el marco de la ventana.

Observaciones:

Una aplicación puede interceptar este mensaje y pintar su propio a medida marco. La región enmascarada para una ventana es siempre rectangular, aunque la forma del marco sea modificada.

Mensaje WM_PAINT

Definición:

WM_PAINT

Una aplicación envía un mensaje WM_PAINT cuando Windows u otra aplicación hace una petición para pintar una porción de la ventana de la aplicación. El mensaje es enviado cuando las funciones [UpdateWindow](#) o [RedrawWindow](#) son llamadas o por la función DispatchMessage cuando la aplicación obtiene un mensaje WM_PAINT tras el uso de las funciones [GetMessage](#) o [PeekMessage](#).

Descripción:

Este mensaje no tiene parámetros.

Valor de retorno:

Una aplicación debe retornar con cero si procesa este mensaje.

Acción por defecto:

La función [DefWindowProc](#) valida la región a actualizar. La función también enviará un mensaje [WM_NCPAINT](#) al procedimiento de ventana si el marco de la ventana ha de ser pintado y envía un mensaje [WM_ERASEBKGD](#) si el fondo de la ventana debe ser borrado.

Observaciones:

El sistema envía este mensaje cuando no hay ningún otro mensaje en la cola de la aplicación. [DispatchMessage](#) determina a dónde enviar el mensaje; [GetMessage](#) determina el mensaje a procesar. GetMessage vuelve con el mensaje WM_PAINT cuando no hay otros mensajes en la cola de la aplicación, y DispatchMessage envía el mensaje al procedimiento de ventana adecuado.

Una ventana recibirá mensajes de pintar internos como resultado de llamadas a [RedrawWindow](#) con el flag RDW_INTERNALPAINT activo. En este caso, la ventana no tendrá una región de actualización. La aplicación deberá llamar a la función [GetUpdateRect](#) para determinar si la ventana tiene una región a actualizar. Si GetUpdateRect retorna con cero, la aplicación no debería llamar a las funciones [BeginPaint](#) y [EndPaint](#).

Una aplicación debe comprobar internamente si es necesario que se pinte consultando sus estructuras de datos internas para cada mensaje WM_PAINT, ya que un mensaje WM_PAINT puede ser causado tanto con una región de

actualización no nula como por una llamada a la función `RedrawWindow` con el flag `RDW_INTERNALPAINT` activo.

Windows envía internamente un mensaje `WM_PAINT` sólo una vez. Después, se devuelve un mensaje interno `WM_PAINT` desde `GetMessage` o `PeekMessage` o se envía a la ventana mediante `UpdateWindow`, Windows no envía ni cursa más mensajes `WM_PAINT` hasta que la ventana no sea invalidada o hasta que `RedrawWindow` sea llamada de nuevo con el flag `RDW_INTERNALPAINT` activo.

Mensaje WM_QUIT

Definición:

```
WM_QUIT  
nExitCode = (int) wParam; // código de salida
```

El mensaje WM_QUIT indica una petición para terminar una aplicación y es generado cuando una aplicación llama a la función [PostQuitMessage](#). También provoca que la función [GetMessage](#) retorne con cero.

Descripción:

nExitCode: valor de wParam. Indica el código de salida suministrado en la función PostQuitMessage.

Valor de retorno:

Este mensaje no tiene valor de retorno, ya que hace que el bucle de mensajes termine antes de que el mensaje sea enviado al procedimiento de ventana de la aplicación.

Mensaje WM_SETTEXT

Definición:

```
WM_SETTEXT
wParam = 0; // no se usa; debe ser cero
lParam = (LPARAM) (LPCTSTR) lpsz; // dirección de la cadena del
texto de la ventana
```

Una aplicación envía un mensaje WM_SETTEXT para cambiar el texto de una ventana.

Descripción:

lpsz: valor de lParam. Puntero a una cadena terminada en cero que contiene el texto de la ventana.

Valor de retorno:

El valor de retorno es TRUE si el texto se cambia. Es FALSE (para un control edit), LB_ERRSPACE (para un list box) o CB_ERRSPACE (oara un combo box) si no hay suficiente espacio disponible para colocar el texto en el control. Es CB_ERR si el mensaje se envía a un combo box sin control edit.

Acción por defecto:

La función [DefWindowProc](#) cambia el texto de la ventana y lo muestra.

Observaciones:

Para un control edit, el texto es el contenido el control edit. Para un combo box, el texto es el contenido de la porción control edit del combo box. Para un button, el texto es el nombre del botón. Para otras ventanas, el texto es el título de la ventana.

Este mensaje no cambia la selección actual en un list box o en un combo box. Una aplicación debe usar el mensaje [CB_SELECTSTRING](#) para seleccionar el item en un list box que coincida con el texto en el control edit.

Mensaje WM_SYSCHAR

Definición:

```
WM_SYSCHAR  
chCharCode = (TCHAR) wParam; // código de carácter  
lKeyData = lParam; // dato de la tecla
```

El mensaje WM_SYSCHAR se envía a la ventana con el foco del teclado cuando un mensaje [WM_SYSKEYDOWN](#) es traducido por la función [TranslateMessage](#). Especifica el código de carácter de una tecla de carácter de sistema, es decir, una tecla de carácter que se pulso mientras la tecla ALT estaba pulsada.

Descripción:

chCharCode: valor de wParam. Especifica el código tecla de menú del sistema.

lKeyData: valor de lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Observaciones:

Cuando el código de contexto es cero, el mensaje pudo ser pasado por la función [TranslateAccelerator](#), la cual lo manejará como si se tratara de un mensaje de tecla estándar en lugar de un mensaje de tecla de sistema. Esto permite usar las teclas de acelerador por la ventana activa aunque no tenga el foco del teclado.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_SYSCOMMAND

Definición:

```
WM_SYSCOMMAND
uCmdType = wParam;          // tipo de comando de sistema pedido
xPos = LOWORD(lParam);      // posición horizontal, en coordenadas
de pantalla
yPos = HIWORD(lParam);     // posición vertical, en coordenadas de
pantalla
```

Una ventana recibe este mensaje cuando el usuario elige un comando desde el menú de sistema (también conocido como menú de control) o cuando el usuario pulsa el botón de maximizar o minimizar.

Descripción:

uCmdType: indica el tipo de comando de sistema pedido. Puese ser uno de los siguientes valores:

Valor	Significado
SC_CLOSE	Cierra la ventana.
SC_CONTEXTHELP	Cambia el cursor al signo de interrogación con un puntero. Si el usuario hace click en un control en el cuadro de diálogo, el control recibirá un mensaje WM_HELP.
SC_DEFAULT	Selecciona el elemento por defecto; el usuarui hizo doble click en el menú de sistema.
SC_HOTKEY	Activa la ventana asociada con la acción especificada en el "hot key". La palabra de menor peso del parámetro lParam identifica la ventaa a activar.
SC_HSCROLL	Desplaza el contenido de la ventana horizontalmente.
SC_KEYMENU	Recupera el menú del sistema como resultado de la pulsación de una tecla.
SC_MAXIMIZE (o SC_ZOOM)	Maximiza la ventana.
SC_MINIMIZE (o SC_ICON)	Minimiza la ventana.
SC_MONITORPOWER	Sólo para Windows 95: ajusta el estado del monitor. Este comando soporta

	dispositivos que tengan propiedades de ahorro de energía, como las baterías de un ordenador portátil.
SC_MOUSEMENU	Recupera el menú del sistema como resultado de un click del ratón.
SC_MOVE	Mueve la ventana.
SC_NEXTWINDOW	Cambia a la siguiente ventana.
SC_PREVWINDOW	Cambia a la ventana anterior.
SC_RESTORE	Recupera el tamaño y posición normales de la ventana.
SC_SCREENSAVE	Ejecuta la aplicación de salva pantallas especificada en la sección [boot] del fichero SYSTEM.INI.
SC_SIZE	Cambia el tamaño de la ventana.
SC_TASKLIST	Ejecuta o activa el gestor de tareas de Windows.
SC_VSCROLL	Desplaza el contenido de la ventana verticalmente.

xPos: indica la posición horizontal del cursor en coordenadas de pantalla, si se eligió un comando del menú del sistema con el ratón. De otro modo, este parámetro no se usa.

yPos: indica la posición vertical del cursor en coordenadas de pantalla, si se eligió un comando del menú del sistema con el ratón. Este parámetro es -1 si el comando fue elegido mediante un acelerador o cero si fue usando un mnenónico.

Valor de retorno:

Una aplicación que procese este mensaje debe retornar con cero.

Observaciones:

La función [DefWindowProc](#) ejecuta las acciones predefinidas de las peticiones del menú del sistema de la tabla anterior.

En los mensajes WM_SYSCOMMAND, los cuatro bits de menor peso del parámetro uCmdType se usan internamente por Windows. Para obtener el resultado correcto cuando se comprueba el valor de uCmdType, la aplicación debe cambiar el valor 0xFFFF0 con el valor de uCmdType usando el operadore de bits AND.

Los elementos del menú de sistema pueden ser modificados usando las funciones [GetSystemMenu](#), [AppendMenu](#), [InsertMenu](#), [ModifyMenu](#), [InsertMenuItem](#) y

[SetMenuItem](#). Las aplicaciones que modifiquen el menú del sistema deben procesar los mensajes WM_SYSCOMMAND.

Una aplicación puede llevar a cabo cualquier comando del sistema en cualquier momento pasando un mensaje WM_SYSCOMMAND a la función DefWindowProc. Cualquier mensaje WM_SYSCOMMAND no manipulado por la aplicación debe ser pasado a la función DefWindowProc. Cualquier comando nuevo añadido por la aplicación debe ser procesado por ella y nunca debe ser pasado a la función DefWindowProc.

Las teclas de aceleración que están definidas para elegir elementos del menú del sistema son traducidas a mensajes WM_SYSCOMMAND; el resto de las teclas de aceleración se traducen a mensajes WM_COMMAND.

Mensaje WM_SYSDEADCHAR

Definición:

```
WM_SYSDEADCHAR  
chCharCode = (TCHAR) wParam;    // código de carácter  
lKeyData = lParam;             // dato de la tecla
```

El mensaje WM_SYSDEADCHAR es enviado a la ventana con el foco del teclado cuando un mensaje [WM_SYSKEYDOWN](#) es traducido por la función [TranslateMessage](#). WM_SYSDEADCHAR indica el código de carácter generado por una tecla muerta, es decir, una tecla muerta pulsada mientras la tecla ALT estaba pulsada. Una tecla muerta es la que genera un carácter como la a acentuada (á), que se combina con otro carácter para formar un carácter compuesto. Por ejemplo, el carácter (á) se genera pulsando la tecla muerta del acento, y después la tecla 'a'.

Descripción:

chCharCode: valor de wParam. Especifica el código de carácter generado por la tecla muerta.

lKeyData: valor de lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.

31

Indica el estado de transición. El valor es 1 si la tecla fue liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si proceso este mensaje.

Observaciones:

El mensaje WM_SYSDEADCHAR se usa normalmente por aplicaciones que informan al usuario sobre cada tecla pulsada. Por ejemplo, una aplicación que pueda mostrar el acento en la posición actual del cursor sun over el caret.

Puesto que no es necesario que exista una correspondencia uno a uno entre las teclas pulsadas y los mensaje de carácter generados, la información en la palabra alta del parámetro lKeyData normalmente no es útil para las aplicaciones. Esta información se aplica sólo al mensaje [WM_KEYDOWN](#) que precede al mensaje WM_CHAR más recientemente enviado.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_SYSKEYDOWN

Definición:

```
WM_SYSKEYDOWN
nVirtKey = (int) wParam;    // código de tecla virtual
lKeyData = lParam;         // dato de tecla
```

El mensaje WM_SYSKEYDOWN es enviado a la ventana que tiene el foco del teclado cuando es usuario mantiene pulsada la tecla ALT y después pulsa otra tecla. También ocurre cuando ninguna ventana tiene el foco del teclado; en ese caso se envía el mensaje WM_SYSKEYDOWN a la ventana activa. La ventana que recibe el mensaje puede distinguir entre estas dos situaciones comprobando el código de contexto en el parámetro lKeyData.

Descripción:

nVirtKey: valor del parámetro wParam. Indica el código de la tecla virtual pulsada.

lKeyData: valor del parámetro lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue

	liberada o 0 si permacece pulsada.
--	------------------------------------

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Acción por defecto:

La función [DefWindowProc](#) examina la tecla leída y genera un mensaje [WM_SYSCOMMAND](#) si la tecla es TAB o ENTER.

Observaciones:

Cuando un código de contexto es cero, el mensaje puede ser pasado a la función [TranslateAccelerator](#), que lo manejará como si fuera un mensaje normal de tecla en lugar de un mensaje de carácter de sistema. Esto permite usar aceleradores de teclado con la ventana activa aunque no tenga el foco del teclado.

A causa de la repetición automática, más de un mensaje WM_SYSKEYDOWN puede ocurrir antes de que se envíe un mensaje [WM_SYSKEYUP](#). El estado previo de la tecla (bit 30) puede usarse para determinar si el mensaje [WM_SYSKEYDOWN](#) indica una primera pulsación o una repetición automática.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Mensaje WM_SYSKEYUP

Definición:

```
WM_SYSKEYUP
nVirtKey = (int) wParam;    // código de tecla virtual
lKeyData = lParam;        // dato de tecla
```

El mensaje WM_SYSKEYUP es enviado a la ventana con el foco del teclado cuando el usuario suelta una tecla que fue presionada mientras estaba pulsada la tecla ALT. También cuando ninguna ventana tiene el foco del teclado, en ese caso, el mensaje WM_SYSKEYUP se envía a la ventana activa. La ventana que recibe el mensaje puede distinguir entre estas dos situaciones verificando el código de contexto del parámetro lKeyData.

Descripción:

nVirtKey: valor del parámetro wParam. Indica el código de la tecla virtual liberada.

lKeyData: valor del parámetro lParam. Especifica el contador de repetición, código de barrido, el flag de teclado extendido, código de entorno, flag de estado previo de la tecla y flag de estado de transición, se trata de un campo de bits como se muestra en la siguiente tabla:

Valor	Descripción
0— 15	Indica el contador de repetición. El valor es el número de veces que se repite una tecla como resultado de haber dejado pulsada una tecla.
16— 23	Indica el código de barrido. Depende del fabricante y modelo del equipo.
24	Indica si la tecla es una tecla extendida, como la tecla derecha de ALT and CTRL que aparece en el teclado mejorado de 101 o 102 teclas. El valor es uno si es una tecla extendida y cero si no lo es.
25— 28	Reservado, no usar.
29	Indica el código de contexto. El valor es 1 si la tecla ALT estaba pulsada mientras la tecla fue pulsada; en caso contrario es 0.
30	Indica el estado previo de la tecla. El valor es 1 si la tecla estaba pulsada antes de que fuera enviado el mensaje y 0 si no lo estaba.
31	Indica el estado de transición. El valor es 1 si la tecla fue

liberada o 0 si permanece pulsada.

Valor de retorno:

La aplicación debe retornar cero si procesó este mensaje.

Acción por defecto:

La función DefWindowProc envía un mensaje [WM_SYSCOMMAND](#) a la ventana de mayor nivel si la tecla F10 o la tecla ALT fueron liberadas. Como parámetro wParam parameter del mensaje se usa SC_KEYMENU.

Observaciones:

Cuando un código de contexto es cero, el mensaje puede ser pasado a la función [TranslateAccelerator](#), que lo manejará como si fuera un mensaje normal de tecla en lugar de un mensaje de carácter de sistema. Esto permite usar aceleradores de teclado con la ventana activa aunque no tenga el foco del teclado.

Para teclados mejorados de 101 y 102 teclas, las teclas extendidas son el ALT y el CTRL derechos en la sección principal del teclado y las teclas INS, DEL, HOME, END, PAGE UP, PAGE DOWN y las teclas de flechas en el grupo de teclas a la izquierda del teclado numérico; también la tecla de dividir (/) y ENTER del teclado numérico. Algunos otros teclados pueden soportar el bit de tecla extendida en el parámetro lKeyData.

Para teclados mejorados de 102 teclas no U.S., la tecla ALT derecha es manejada como un CTRL+ALT. La siguiente tabla muestra la secuencia de mensajes que resultan cuando el usuario presiona y libera esta tecla:

Mensaje	Código de tecla virtual
WM_KEYDOWN	VK_CONTROL
WM_SYSKEYDOWN	VK_MENU
WM_KEYUP	VK_CONTROL
WM_SYSKEYUP	VK_MENU

Mensaje WM_TIMER

Definición:

```
WM_TIMER
wTimerID = wParam;           // identificador de cronómetro
(timer)
tmprc = (TIMERPROC *) lParam; // dirección de la función de
respuesta (callback)
```

El mensaje WM_TIMER es enviado a la cola de mensajes de procesos (installing thread) o a la función de respuesta [TimerProc](#) apropiada después del intervalo especificado en la función [SetTimer](#) usada para instalar el timer.

Descripción:

wTimerID: valor del parámetro wParam. Especifica el identificador del timer o cronómetro.

tmprc: valor del parámetro lParam. Apunta a la función de respuesta o callback definida por la aplicación que fue pasada a la función SetTimer cuando el timer fue instalado. Si el parámetro tmprc no es NULL, Windows pasas el mensaje WM_TIMER a la función callback especificada con preferencia a enviar el mensaje a la cola de mensajes del proceso.

Valor de retorno:

Una aplicación que procese este mensaje debe retornar con cero.

Observaciones:

La función [DispatchMessage](#) vuelve a enviar este mensaje si no hay otros mensajes en la cola de mensajes del procesos.

Mensaje WM_VSCROLL

Definición:

```
WM_VSCROLL
nScrollCode = (int) LOWORD(wParam); // valor de scrollbar
nPos = (short int) HIWORD(wParam); // posición de la caja de
desplazamiento
hwndScrollBar = (HWND) lParam; // manipulador del scrollbar
```

El mensaje WM_VSCROLL es enviado a una ventana cuando ocurre un evento de scroll en su scrollbar vertical estándar. Este mensaje también se envía a la ventana propietaria de un control scrollbar vertical cuando un evento ocurre en ese control.

Descripción:

nScrollCode: valor de la palabra de menor peso de wParam. Especifica un valor de scrollbar que indica una petición de desplazamiento del usuario. Este parámetro puede tomar uno de los siguientes valores:

Valor	Significado
SB_BOTTOM	Desplazamiento total hacia abajo.
SB_ENDSCROLL	Termina el desplazamiento.
SB_LINEUP	Desplaza hacia arriba una unidad.
SB_LINEDOWN	Desplaza hacia abajo una unidad.
SB_PAGEUP	Desplaza hacia arriba en la altura de la ventana.
SB_PAGEDOWN	Desplaza hacia abajo en la altura de la ventana.
SB_THUMBPOSITION	Desplaza a una posición absoluta. La posición actual se especifica mediante el parámetro nPos.
SB_THUMBTRACK	Arrastra la caja de desplazamiento a la posición especificada. La posición actual se especifica mediante el parámetro nPos.
SB_TOP	Desplazamiento total hacia arriba.

nPos: valor de la palabra de mayor peso de wParam. Especifica la posición actual de la caja de desplazamiento si el parámetro nScrollCode es SB_THUMBPOSITION o SB_THUMBTRACK; en cualquier otro caso, nPos no se usa.

hwndScrollBar: valor de lParam. Identifica el control si el mensaje WM_VSCROLL fue enviado por un control scrollbar. Si el mensaje WM_VSCROLL fue enviado por un scrollbar estándar, hwndScrollBar no se usa.

Valor de retorno:

Si una aplicación procesa éste mensaje, debe retornar con cero.

Observaciones:

El mensaje de notificación SB_THUMBTRACK se usa normalmente por aplicaciones que proporcionan una respuesta mientras el usuario arrastra la caja de desplazamiento.

Su una aplicación desplaza el contenido de una ventana, debe también actualizar la posición de la caja de desplazamiento usando la función [SetScrollPos](#).

Como el mensaje WM_VSCROLL proporciona sólo 16 bits para el dato de la posición, las aplicaciones que confían sólo en WM_VSCROLL (y WM_HSCROLL) para actualizar las posiciones tienen un valor máximo de 65,535.

Sin embargo, como las funciones [SetScrollPos](#), [SetScrollRange](#), [GetScrollPos](#) y [GetScrollRange](#) soportan valores de 32-bit para la posición del scrollbar, existe un modo de sortear la barrera de 16-bit de los mensajes WM_HSCROLL y WM_VSCROLL. Ver [GetScrollPos](#) para ver una descripción de esa técnica y sus límites.



Recursos

Atributos comunes de recursos

Todas las sentencias de definición de recursos incluyen una opción "load-mem" que especifica la carga y las características de memoria del recurso. Estos atributos están divididos en dos grupos: atributos de carga y atributos de memoria. El único atributo que se usa por Win32 es DISCARDABLE. El resto de los atributos especificados se incluyen por compatibilidad con los scripts existentes, pero son ignorados.

Atributos de carga

Los atributos de carga indican cuando debe ser cargado el recurso. Este parámetro debe ser uno de los siguientes:

PRELOAD: ignorado. En Windows 16-bit, el recurso es cargado junto con el fichero ejecutable.

LOADONCALL: ignorado. En Windows 16-bit, el recurso se carga cuando es llamado.

Atributos de memoria

Los atributos de memoria especifican si el recurso es fijo o puede cambiar de posición en memoria, también si es descartable o puro. El parámetro de memoria puede ser uno o más de los siguientes:

FIXED: ignorado. En Windows 16-bit, el recurso permanece en una posición de memoria fija.

MOVEABLE: ignorado. En Windows 16-bit, el recurso puede ser movido si es necesario durante un proceso de compactación de memoria.

DISCARDABLE: el recurso puede ser descartado y no va a ser usado más.

PURE: ignorado. Se acepta por compatibilidad.

IMPURE: ignorado. Se acepta por compatibilidad.

Para recursos de cursores, iconos y fuente, el valor por defecto es DISCARDABLE.



Recursos

Sentencia CAPTION

`CAPTION textocaption`

La sentencia CAPTION define el título de un cuadro de diálogo. El título aparecerá en la barra de título del diálogo (si existe).

El título por defecto es un título vacío.

Parametro:

textocaption: especifica una cadena de caracteres entre comillas dobles.

Sentencia CHARACTERISTICS

`CHARACTERISTICS dword`

La sentencia CHARACTERISTICS permite al programador añadir información sobre el recurso que puede ser usada por herramientas que lean o escriban los ficheros de definición de recursos. El valor dword especificado, aparecerá también en el fichero de recursos compilado .RES. Sin embargo, el valor no se almacena en el fichero ejecutable y no tiene significado para Windows.

La sentencia CHARACTERISTICS aparece antes del BEGIN en definiciones de recursos de tipo ACCELERATORS, DIALOG, MENU, RCDATA o STRINGTABLE. El valor especificado se aplica sólo a ese recurso.

Parametro:

dword: un valor de doble palabra definido por el usuario.

Sentencia CLASS

`CLASS class`

La sentencia CLASS define la clase de un cuadro de diálogo. Debe aparecer en la sección opcional antes de la palabra BEGIN. Si no aparece, se usa la clase estándar de Windows.

Parámetros:

class: especifica un entero sin signo de 16 bits o una cadena entre comillas dobles, que identifica la clase del cuadro de diálogo. Si el procedimiento de ventana para la clase no procesa un mensaje enviado a él, debe llamar a la función [DefDlgProc](#) para asegurar que todos los mensajes son manipulados apropiadamente por el diálogo. Una clase privada puede usar DefDlgProc como procedimiento de ventana por defecto. La clase debe ser registrada con el valor DLGWINDOWEXTRA para el miembro cbWndExtra de la estructura [WNDCLASS](#).

Observaciones:

La sentencia CLASS sólo debe ser usada en casos especiales, puesto que pasa por encima del proceso normal del cuadro de diálogo. La sentencia CLASS convierte un cuadro de diálogo en una ventana de la clase especificada; dependiendo de la clase, esto puede dar resultados no deseados. No uses nombres de clases de controles redefinidos con esta sentencia.

CONTROL: Controles Generales

```
CONTROL text, id, class, style, x, y, width, height [, extended-style]
```

Esta sentencia define una ventana de control definida por el usuario.

Parámetros:

text: especifica el texto que se muestra junto con el control. El texto es colocado en el interior de las dimensiones especificadas para el control o junto al control.

Este parámetro contiene cero o más caracteres encerrados entre comillas dobles. Las cadenas se terminan automáticamente con cero y se convierten a Unicode en el fichero de recursos resultante, excepto para cadenas especificadas en sentencias de datos sin formato. (Los datos sin formato se pueden incluir bajo la sentencia RCDATA y recursos definidos por el usuario.) Para especificar una cadena Unicode en datos sin formato, hay que especificar explícitamente que la cadena es "wide-character" usando el prefijo L.

Por defecto, los caracteres entre comillas dobles son caracteres ANSI y las sentencias de escape son interpretadas como secuencias de escape de un byte. Si la cadena está precedida con el prefijo L, la cadena se considera como cadena "wide-character" y las secuencias de escape se interpretan como secuencias de escape de dos bytes que especifican los caracteres Unicode. Si el texto debe incluir comillas dobles, se deben escribir las comillas dobles dos veces o usar la secuencia de escape \".

EL carácter & en el texto indica que el siguiente carácter se usa como un mnemónico para el control. Cuando el control se muestra por pantalla, el carácter & no se muestra, pero el carácter mnemónico se subraya. El usuario puede seleccionar el control presionando la tecla correspondiente al carácter mnemónico subrayado. Para usar el carácter & en una cadena hay que insertarlo dos veces (&&).

id: especifica el identificador del control. Debe ser un valor de entero sin signo de 16 bits dentro del rango entre 0 y 65535 o una expresión aritmética que se evalúe en ese rango.

class: especifica un nombre redefinido, una cadena de caracteres, un valor entero sin signo de 16 bits que define la clase. Puede ser cualquiera de las clases de controles; para una lista de las clases de controles, ver la lista que sigue a esta descripción. Si el valor es un nombre redefinido por la aplicación, debe ser una cadena entre comillas dobles.

style: especifica un nombre redefinido o un valor entero que especifique el estilo del control. El significado exacto del estilo depende del valor de class, las secciones que siguen a esta descripción muestran las clases de controles y sus correspondientes estilos.

x: especifica la coordenada x del lado izquierdo del control, relativo al lado izquierdo del diálogo. Debe ser un valor entero sin signo de 16 bits entre 0 y 65535. La coordenada se expresa en unidades de diálogo y es relativa al origen del cuadro de diálogo, ventana o control que contenga al control especificado.

y: especifica la coordenada y del lado superior del control, relativo al lado superior del diálogo. Debe ser un valor entero sin signo de 16 bits entre 0 y 65535. La coordenada se expresa en unidades de diálogo y es relativa al origen del cuadro de diálogo, ventana o control que contenga al control especificado.

width: especifica la anchura del control. Este valor es un entero sin signo de 16 bits entre 1 y 65535. La anchura se expresa en cuartos de unidades de carácter.

height: especifica la altura del control. Este valor es un entero sin signo de 16 bits entre 1 y 65535. La altura se expresa en octavos de unidades de carácter.

extended-style: especifica los estilos extendidos (WS_EX_XXX). Es obligatorio especificar un estilo para poder especificar también un estilo extendido. Ver también [EXSTYLE](#).

A continuación se describen las seis clases de controles que existen.

Control de clase Button

Un control "button" es una pequeña ventana hija rectangular que representa un botón o pulsador que el usuario puede encender o apagar mediante un clic del ratón. Los controles Button pueden usarse solos o en grupos, y pueden también estar etiquetados o aparecer sin texto. Los controles Button normalmente cambian de aspecto cuando el usuario hace clic sobre ellos.

Un botón puede tener sólo uno de los siguientes estilos, con la excepción de BS_LEFTTEXTBS, el cual puede ser combinado con "check boxes" y "radio buttons".

Para ver los estilos disponibles para button consultar [estilos button](#).

Control de clase Combobox

Los controles combo-box consisten en un campo de selección, similar a un control edit, y un list box. El list box puede ser mostrado todo el tiempo o puede desplegarse cuando el usuario selecciona el "pop box" que hay junto al campo de selección.

Dependiendo del estilo del combo-box, el usuario podrá o no editar el contenido del campo de selección. Si el list box es visible, escribir caracteres en el cuadro de selección hará que la primera entrada del list-box que coincida con el texto introducido sea seleccionada. Y la inversa, seleccionando un ítem de la lista se muestra el texto en el campo de selección. Los estilos combo-box se describen a continuación.

Para ver los estilos disponibles para combobox consultar [estilos combobox](#).

Control de clase Edit

Un control edit es una ventana hija rectangular en la cual el usuario puede introducir texto desde el teclado. El usuario selecciona el control y le da el foco de entrada haciendo clic con el ratón dentro de él o presionando la tecla TAB. El usuario puede introducir texto cuando el control muestra un punto de inserción intermitente. El ratón puede usarse para mover el cursor y para seleccionar caracteres para ser reemplazados o para mover el cursor para insertar caracteres. La tecla de borrar puede usarse para eliminar caracteres.

Los controles edit usan fuentes de anchura constante y muestran caracteres Unicode. También expanden los caracteres tab en los espacios suficientes para mover el cursor a la siguiente parada de tabulación. Las paradas de tabulación se asume que están cada ocho caracteres.

Para ver los estilos disponibles para edit consultar [estilos edit](#).

Control de clase Listbox

Los controles listbox consisten en una lista de cadenas de caracteres. Este control se usa cada vez que una aplicación necesita presentar una lista de nombres, como nombres de ficheros, que el usuario pueda ver y seleccionar. El usuario puede seleccionar una cadena apuntando a la cadena con el ratón y haciendo clic con el botón del ratón. Cuando una cadena se selecciona, se resalta y se envía un mensaje de notificación a la ventana padre. Se puede usar una barra de scroll con los listbox para desplazar listas muy largas o demasiado anchas para la ventana.

Para ver los estilos disponibles para listbox consultar [estilos listbox](#).

Control de clase Scrollbar

Un control scroll-bar es un rectángulo que contiene un rectángulo para hojear (scroll thumb) y flechas de dirección en ambos extremos. El scroll bar envía mensajes de notificación a su ventana padre cada vez que el usuario hace clic con el ratón en el control. La ventana padre es la responsable de actualizar la posición del "thumb", si es necesario. El control scroll-bar tiene la misma apariencia y función que los scroll bars usados en las ventanas ordinarias. Pero al contrario que los scroll bars, los controles scroll-bar pueden ser colocados en cualquier posición

en el interior de una ventana y usados cuando se necesite introducir valores de desplazamiento para una ventana.

Para ver los estilos disponibles para scrollbar consultar [estilos scrollbar](#).

Control de clase Static

Los controles static son simples campos de texto, cajas y rectángulos que pueden usarse para etiquetar, agrupar o separar otros controles. Los controles static no toman entradas y no proporcionan salidas.

Para ver los estilos disponibles para static consultar [estilos static](#).

DIALOG

La sentencia DIALOG define una ventana que una aplicación puede usar para crear cuadros de diálogo. Esta sentencia define la posición y dimensiones del cuadro en la pantalla así como su estilo.

Definición:

```
nameID DIALOG [load-mem] x, y, ancho, alto  
[sentencias-opcionales]  
BEGIN  
    sentencias de controles  
    . . .  
END
```

Descripción:

nameID: identifica el cuadro de diálogo. Puede ser un nombre único o un valor entero de 16 bits único entre 1 y 65535.

load-mem: indica el tipo de carga y atributos de memoria del recurso. Para más información ver "[Atributos Comunes de Recursos](#)".

sentencias-opcionales: indican opciones para el cuadro de diálogo. Pueden ser ninguna o más de las siguientes:

CAPTION "texto": especifica el título del cuadro de diálogo si tiene barra de título. Ver [CAPTION](#) para más información.

CHARACTERISTICS dword: especifica una palabra doble definida por el usuario para usarse con las herramientas de recursos. Este valor no se usa por Windows. Para más información consultar [CHARACTERISTICS](#).

CLASS clase: especifica un entero sin signo de 16 bits o una cadena, entre comillas dobles, que identifica la clase del cuadro de diálogo. Ver [CLASS](#) para mayor información.

LANGUAGE lenguaje,sublenguaje: indica el lenguaje del cuadro de diálogo. Ver [LANGUAGE](#) para más información.

STYLE estilos: especifica los estilos para el cuadro de diálogo. Ver [STYLE](#) para mayor información.

EXSTYLE=estilos-extendidos: especifica los estilos extendidos del cuadro de diálogo. Ver [EXSTYLE](#) para más información.

VERSION dword: especifica un valor de palabra doble definido por el usuario. Esta sentencia está diseñada para usarse con herramientas de recursos y no se usa por Windows. Para más detalles, ver [VERSION](#).

Para más información sobre los parámetros x, y, ancho y alto, ver "[Parámetros Comunes de Sentencias](#)".

Observaciones:

La función [GetDialogBaseUnits](#) devuelve las unidades base del diálogo en pixels. El significado exacto de las coordenadas depende del estilo definido en la sentencia opcional STYLE. Para cuadros de diálogo child-style, las coordenadas son relativas al origen de la ventana padre, a no ser que el diálogo tenga el estilo DS_ABSALIGN; en ese caso, las coordenadas son relativas al origen de la pantalla.

No uses el estilo WS_CHILD con cuadros de diálogo modales. La función [DialogBox](#) siempre desactiva la ventana padre o propietaria del nuevo cuadro de diálogo creado. Cuando la ventana padre se desactiva, sus ventanas hijas se desactivan también implícitamente. En el momento en que la ventana padre de un diálogo con el estilo child-style se desactiva, el diálogo se desactiva también.

Si un cuadro de diálogo tiene el estilo DS_ABSALIGN, las coordenadas de la esquina superior izquierda son relativas al origen de la pantalla, en lugar de a la esquina superior izquierda de la ventana padre. El uso típico de este estilo es cuando se quiere que el cuadro de diálogo empiece en una zona específica de la pantalla, independientemente de la posición de la ventana padre en la pantalla.

El nombre DIALOG puede ser usado también como parámetro de nombre de clase en la función [CreateWindow](#) para crear una ventana con los atributos de un cuadro de diálogo.

Sentencia EXSTYLE

```
EXSTYLE extended-style
```

El comando EXSTYLE permite asignar a un diálogo estilos del tipo WS_EX_style. En una definición de recursos, la sentencia EXSTYLE se sitúa en la zona de sentencias opcionales antes del BEGIN. También puede ser situada junto a los parámetros load-mem en la sentencia DIALOG de este modo:

```
name DIALOG [load-mem] EXSTYLE=extended-style x, y, ...
```

Para los controles, los estilos extendidos se especifican después de la opción de estilo en cualquier definición de un control:

```
control text, id, x, y, width, height, [style[, extended-style]]
```

Parámetros:

extended-style: el estilo [WS_EX_style](#) para el cuadro de diálogo o para el control.

Sentencia LANGUAGE

LANGUAGE language, sublanguage

La sentencia LANGUAGE selecciona el lenguaje para todos los recursos hasta la siguiente sentencia LANGUAGE o hasta el final del fichero. Cuando la sentencia LANGUAGE aparece antes del BEGIN en definiciones de recursos ACCELERATORS, DIALOG, MENU, RCDATA o STRINGTABLE, el lenguaje especificado se aplica sólo a ese recurso.

Parámetros:

language: identificador de lenguaje. Debe ser una de las constantes definidas en WINNLS.H

sublanguage: identificador de sublenguaje. Debe ser una de las constantes definidas en WINNLS.H

MENU

La sentencia MENU define el contenido de un recurso de menú. Un recurso de menú es una colección de informaciones que definen la apariencia a funcionamiento del menú de una aplicación. Un menú es una herramienta de entrada especial que permite al usuario elegir comandos de una lista de nombres de comandos.

Definición:

```
menuID MENU [load-mem]
[sentencias-opcionales]
BEGIN
    definiciones_de_item
    . . .
END
```

Descripción:

menuID: identifica el menú. Puede ser un nombre único o un valor entero sin signo de 16 bits único entre 1 y 65535.

load-mem: indica el tipo de carga y atributos de memoria del recurso. Para más información ver "[Atributos Comunes de Recursos](#)".

sentencias-opcionales: indican opciones para el cuadro de diálogo. Pueden ser ninguna o más de las siguientes:

CHARACTERISTICS dword: especifica una palabra doble definida por el usuario para usarse con las herramientas de recursos. Este valor no se usa por Windows. Para más información consultar [CHARACTERISTICS](#).

LANGUAGE lenguaje,sublenguaje: indica el lenguaje del cuadro de diálogo. Ver [LANGUAGE](#) para más información.

VERSION dword: especifica un valor de palabra doble definido por el usuario. Esta sentencia está diseñada para usarse con herramientas de recursos y no se usa por Windows. Para más detalles, ver [VERSION](#).

Las definiciones de item están compuestas por sentencias [MENUITEM](#) y/o sentencias [POPUP](#).

Sentencia MENUITEM

```
MENUITEM texto, resultado, [lista_de_opciones]
MENUITEM SEPARATOR
```

La sentencia MENUITEM define un ítem de menú.

Parámetros:

texto: especifica el nombre del ítem de menú. La cadena puede contener las secuencias de escape `\t` y `\a`. El carácter `\t` inserta un tabulador en la cadena y se usa para alinear el texto en columnas. Los caracteres `tab` debes ser usados sólo en menús pop-up, nunca en barras de menú. (Para información sobre menús pop-up, ver la sentencia [POPUP](#)). El carácter `\a` alinea todo el texto que le sigue al borde derecho de la barra de menú o del menú pop-up.

resultado: especifica el resultado generado cuando el usuario selecciona el ítem de menú. Este parámetro debe ser un valor entero. Los resultados de los ítem de menú son siempre enteros; cuando el usuario hace clic sobre un ítem de menú, el resultado es enviado a la ventana a la que pertenece el menú.

lista_de_opciones: especifica la apariencia del ítem de menú. Este parámetro opcional puede tomar uno o más de los valores redefinidos como opciones de menú, separados por comas o espacios. La opciones son las siguientes:

CHECKED: el ítem tiene una marca de de verificación junto a él.

GRAYED: el nombre del ítem esta inicialmente inactivo y aparece el el menú en gris o con el texto del menú ligeramente degradado.

HELP identifica un ítem de ayuda.

INACTIVE el nombre del ítem se muestra, pero no puede ser seleccionado.

MENUBARBREAK lo mismo que **MF_MENUBREAK** excepto que para menús pop-up, separa la nueva columna de la anterior con una línea vertical.

MENUBREAK coloca el ítem de menú en una nueva línea en ítems de menús de barra estáticos. En menús pop-up, coloca el ítem de menú en una nueva columna sin división entre las columnas.

Las opciones **INACTIVE** y **GRAYED** no pueden usarse juntas.

SEPARATOR: el formato **MENUITEM SEPARATOR** crea un ítem de menú inactivo que se usae como barra de división entre dos ítems de menú activos en un menú pop-up.

Parámetros Comunes de Sentencias

Esta sección lista los parámetros en común entre los recursos o las sentencias de control. Ocasionalmente, ciertas sentencias podrán usar un parámetro de un modo diferente o incluso ignorar un parámetro. Las variaciones en sentencias específicas se describen junto con la sentencia en la referencia alfabética.

Esta forma sólo sirve para los controles comunes, pero existe una forma más general para definir todo tipo de controles, que es la que usaremos normalmente. Se trata de los [controles generales](#).

Parámetros de los controles comunes

La sintaxis general para una definición de un control, y su significado para cada parámetro es el siguiente:

```
control [text,] id, x, y, width, height [, style [, extended-style]]
```

Las unidades de diálogo horizontales se expresan en 1/4 de la unidad básica de anchura de diálogo. Las unidades de diálogo verticales se expresan en 1/8 de la unidad básica de altura de diálogo. Las unidades básicas de diálogo se calculan a partir de la altura y anchura de la fuente de sistema actual. La función [GetDialogBaseUnits](#) devuelve las unidades base del diálogo en pixels. Las coordenadas son relativas al origen del cuadro de diálogo.

Parámetros:

control: palabra clave que indica el tipo de control que se está definiendo, como PUSHBUTTON o CHECKBOX.

text: especifica el texto que se muestra junto con el control. El texto es colocado en el interior de las dimensiones especificadas para el control o junto al control.

Este parámetro contiene cero o más caracteres encerrados entre comillas dobles. Las cadenas se terminan automáticamente con cero y se convierten a Unicode en el fichero de recursos resultante, excepto para cadenas especificadas en sentencias de datos sin formato. (Los datos sin formato se pueden incluir bajo la sentencia RCDATA y recursos definidos por el usuario.) Para especificar una cadena Unicode en datos sin formato, hay que especificar explícitamente que la cadena es "wide-character" usando el prefijo L.

Por defecto, los caracteres entre comillas dobles son caracteres ANSI y las sentencias de escape son interpretadas como secuencias de escape de un byte. Si la cadena está precedida con el prefijo L, la cadena se considera como cadena "wide-

character" y las secuencias de escape se interpretan como secuencias de escape de dos bytes que especifican los caracteres Unicode. Si el texto debe incluir comillas dobles, se deben escribir las comillas dobles dos veces o usar la secuencia de escape \".

EL carácter & en el texto indica que el siguiente carácter se usa como un mnemónico para el control. Cuando el control se muestra por pantalla, el carácter & no se muestra, pero el carácter mnemónico se subraya. El usuario puede seleccionar el control presionando la tecla correspondiente al carácter mnemónico subrayado. Para usar el carácter & en una cadena hay que insertarlo dos veces (&&).

id: especifica el identificador del control. Debe ser un valor de entero sin signo de 16 bits dentro del rango entre 0 y 65535 o una expresión aritmética que se evalúe en ese rango.

x: especifica la coordenada x del lado izquierdo del control, relativo al lado izquierdo del diálogo. Debe ser un valor entero sin signo de 16 bits entre 0 y 65535. La coordenada se expresa en unidades de diálogo y es relativa al origen del cuadro de diálogo, ventana o control que contenga al control especificado.

y: especifica la coordenada y del lado superior del control, relativo al lado superior del diálogo. Debe ser un valor entero sin signo de 16 bits entre 0 y 65535. La coordenada se expresa en unidades de diálogo y es relativa al origen del cuadro de diálogo, ventana o control que contenga al control especificado.

width: especifica la anchura del control. Este valor es un entero sin signo de 16 bits entre 1 y 65535. La anchura se expresa en cuartos de unidades de carácter.

height: especifica la altura del control. Este valor es un entero sin signo de 16 bits entre 1 y 65535. La altura se expresa en octavos de unidades de carácter.

style: especifica los [estilos](#) del control. Usar el operador de bits OR (|) para combinar estilos.

extended-style: especifica los estilos extendidos (WS_EX_XXX). Es obligatorio especificar un estilo para poder especificar también un estilo extendido. Ver también [EXSTYLE](#).

Sentencia POPUP

```
POPUP texto, [lista_de_options]
BEGIN
    definiciones_de_item
    .
    .
    .
END
```

La sentencia POPUP marca el comienzo de una definición de un menú pop-up. un menú pop-up (también conocido como menú drop-down) es un ítem de menú especial que muestra una sublista de ítems de menú cuando se selecciona.

Parámetros:

texto: especifica el nombre del menú pop-up. Esta cadena debe estar entre comillas dobles.

lista_de_options: especifica uno o más de las siguientes opciones de menú redefinidas que definen la apariencia del ítem de menú. Las opciones son las siguientes:

CHECKED: el ítem tiene una marca de verificación junto a él. Esta opción no es válida para menús pop-up del primer nivel.

GRAYED: el nombre del ítem está inicialmente inactivo y aparece en el menú en gris o con el texto del menú ligeramente degradado.

INACTIVE el nombre del ítem se muestra, pero no puede ser seleccionado.

MENUBARBREAK lo mismo que **MF_MENUBREAK** excepto que para menús pop-up, separa la nueva columna de la anterior con una línea vertical.

MENUBREAK coloca el ítem de menú en una nueva línea en ítems de menús de barra estáticos. En menús pop-up, coloca el ítem de menú en una nueva columna sin división entre las columnas.

Las opciones **INACTIVE** y **GRAYED** no pueden usarse juntas.

Sentencia STYLE

STYLE style

La sentencia STYLE define el estilo de ventana del cuadro de diálogo. El estilo especifica si el cuadro es un pop-up o una ventana hija. El estilo por defecto tiene los siguientes atributos: WS_POPUP, WS_BORDER y WS_SYSMENU.

Parámetro:

style: especifica el estilo de ventana. Este parámetro puede ser un valor entero o un nombre redefinido. A continuación se muestra una lista de los estilos redefinidos:

Estilo	Definición
DS_LOCALEDIT	Indica que los controles edit en el cuadro de diálogo usarán memoria de la sección de datos de la aplicación. Por defecto, todos los controles edit de los cuadros de diálogo usan memoria fuera de la sección de datos de la aplicación. Esta característica puede suprimirse añadiendo el flag DS_LOCALEDIT al comando STYLE del cuadro de diálogo. Si no se usa este flag, los mensajes EM_GETHANDLE y EM_SETHANDLE no deben ser usados ya que el almacenamiento para el control no estará en la sección de datos de la aplicación. Esta característica no afecta a los controles edit creados fuera de los cuadros de diálogo.
DS_MODALFRAME	Crea un cuadro de diálogo con un border de diálogo modal que puede ser combinado con una barra de título y un menú de sistema especificando los estilos WS_CAPTION y WS_SYSMENU.
DS_NOIDLEMSG	Suprime los mensajes WM_ENTERIDLE que Windows enviaría a la ventana propietaria del diálogo mientras éste es mostrado.
DS_SYSMODAL	Crea un cuadro de diálogo system-modal.
WS_BORDER	Crea una que tiene de borde una línea fina.
WS_CAPTION	Crea una ventana con una barra de título, (incluye el estilo WS_BORDER).
WS_CHILD	Crea una ventana hija. Este estilo no puede ser usado junto con el estilo WS_POPUP.
WS_CHILDWINDOW	Lo mismo que WS_CHILD.

WS_CLIPCHILDREN	Excluye el área ocupada por ventanas hija cuando se pinta dentro de la ventana padre. Este estilo se usa cuando se crea la ventana padre.
WS_CLIPSIBLINGS	Descarta las áreas relativas de cada una de las ventanas hijas restantes; esto es, cuando una ventana hija concreta recibe un mensaje WM_PAINT, el estilo WS_CLIPSIBLINGS excluye el área ocupada por todas las otras ventanas hijas superpuestas con la región de la ventana a actualizar. Si WS_CLIPSIBLINGS no se especifica y la ventana hija se superpone, es posible, cuando se dibuja en el área de cliente de la ventana hija, que se pinte dentro del área de cliente de la ventana hija colindante.
WS_DISABLED	Crea una ventana que inicialmente está deshabilitada. Una ventana deshabilitada no puede recibir datos del usuario.
WS_DLGFRAME	Crea una ventana con un estilo de borde típico de los cuadros de diálogo. Una ventana con este estilo no puede tener una barra de título.
WS_GROUP	Indica que es el primer control de un grupo de controles. El usuario puede cambiar el foco del teclado de un control de un grupo al siguiente en el mismo grupo usando las teclas de dirección. Todos los controles definidos sin el estilo WS_GROUP después del primer control de grupo pertenecerán al mismo grupo. El siguiente control con el estilo WS_GROUP termina el grupo y empieza el siguiente.
WS_HSCROLL	Crea una ventana que tiene una scroll bar horizontal.
WS_ICONIC	Crea una ventana inicialmente minimizada. El mismo efecto que el estilo WS_MINIMIZE.
WS_MAXIMIZE	Crea una ventana inicialmente maximizada.
WS_MAXIMIZEBOX	Crea una ventana que tiene un botón de Maximizar.
WS_MINIMIZE	Crea una ventana inicialmente minimizada. Lo mismo que el estilo WS_ICONIC.
WS_MINIMIZEBOX	Crea una ventana que tiene un botón de Minimizar.
WS_OVERLAPPED	Crea una ventana "superpuesta". Una ventana "superpuesta" (overlapped) tiene una barra de

	título y un borde. El mismo efecto que el estilo WS_TILED.
WS_OVERLAPPEDWINDOW	Crea una ventana superpuesta con los estilos: WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX y WS_MAXIMIZEBOX. El mismo efecto que el estilo WS_TILEDWINDOW.
WS_POPUP	Crea una ventana "pop-up". Este estilo no puede usarse junto con el estilo WS_CHILD.
WS_POPUPWINDOW	Crea una ventana "pop-up" con los estilos: WS_BORDER, WS_POPUP y WS_SYSMENU. Los estilos WS_CAPTION y WS_POPUPWINDOW deben combinarse para que el menú de sistema sea visible.
WS_SIZEBOX	Crea una ventana que tiene un borde que permite cambiar su tamaño. El mismo efecto que el estilo WS_THICKFRAME.
WS_SYSMENU	Crea una ventana que contiene un menú de sistema en su barra de título. El estilo WS_CAPTION debe ser especificado también.
WS_TABSTOP	Define un control que puede recibir el foco del teclado cuando el usuario pulsa la tecla TAB. Presionando la tecla TAB, el usuario mueve el foco del teclado al siguiente control con el estilo WS_TABSTOP.
WS_THICKFRAME	Crea una ventana que tiene un borde que permite cambiar su tamaño. El mismo efecto que el estilo WS_SIZEBOX.
WS_VISIBLE	Crea una ventana inicialmente visible.
WS_VSCROLL	Crea una ventana con un scroll bar vertical.

Para combinar varios estilos se usa el operador de bits OR (|).

Comentarios:

Si se usan los nombre redefinidos, se debe incluir el fichero de cabecera WINDOWS.H.

Sentencia VERSION

VERSION dword

La sentencia VERSION permite al programador incluir información sobre la versión de un recurso, esta información puede ser usada por herramientas que lean y escriban ficheros de recursos. El valor dword especificado aparecerá junto con el recurso en el fichero de recursos compilado .RES. Sin embargo, el valor no se almacenará junto con el fichero ejecutable y no tiene significado para Windows.

La sentencia VERSION aparece antes del BEGIN en una definición de recurso de ACCELERATORS, DIALOG, MENU, RCDATA o STRINGTABLE. EL valor especificado se aplica sólo a ese recurso.

Parámetro:

dword: un valor de doble palabra definido por el usuario.



Glosario

Agruparemos en ésta página las palabras y siglas que se usan a menudo cuando se programa en Windows:

API (Application Programming Interface).

El Win32 API son librerías de C, aunque nadie nos impide usar lo con un compilador de C++. Contiene todas las funciones necesarias para programar para Windows.

Incluye: el fichero windows.h, constantes, funciones, mensajes, secuencias de escape de impresora y estructuras de datos.

OWL y MFC

También existen librerías de clases para programar en Windows, las más conocidas son OWL (Object Windows Library) de Borland y MFC (Microsoft Foundation Class) de Microsoft. Aquí no las usaremos, pero para algunos elementos de uso frecuente probablemente diseñaremos nuestras propias clases, y probablemente construyamos una librería con ellas.

GDI (Graphics Device Interface).

Funciones para el acceso a los gráficos. Permiten hacer programas gráficos independientes de hardware. Nuestros programas Windows funcionaran independientemente de la tarjeta gráfica, monitor o impresora que usemos. Y tendrán acceso a fuentes de caracteres y funciones para dibujar líneas y formas, manejo de mapas de bits, etc.

SDK (Software Development Kit).

Contiene el Win32 API y los programas necesarios para la depuración y para la creación de ficheros de ayuda.

MAPI (Messaging Application Programming Interface).

Librería que permite usar el correo electrónico desde nuestras aplicaciones para enviar mensajes.

MDI (Multiple Document Interface).

Aplicaciones que pueden manejar varios documentos simultáneamente, estos documentos pueden ser todos del mismo tipo, aunque no necesariamente.

SDI (Single Document Interface).

Aplicaciones que sólo pueden manejar un documento.

GUI (Graphic User Interface)

Interface gráfico para el usuario, son las aplicaciones normales en Windows, el otro tipo son las aplicaciones de consola, que emulan una pantalla de texto como las de MS-DOS.

OEM (original equipment manufacturer)

Fabricación del equipo original. Indica parámetros que dependen del fabricante del hardware.