



UNIVERSIDAD POPULAR AUTÓNOMA DEL
ESTADO DE PUEBLA

SISTEMAS COMPUTACIONALES



BIBLIOTECA CENTRAL
USO ÚNICAMENTE EN SALA

DLLS EN VISUAL C++

TESIS

QUE PARA OBTENER EL TÍTULO:
LICENCIADO EN SISTEMAS COMPUTACIONALES

PRESENTA
RODOLFO ORTEGA PAREDES

TE 005.133

#49813

ORT 2000

ORTEGA PAREDES, RODOLFO
DLLS EN VISUAL C++

H. Puebla de Zaragoza, a 31 de marzo de 2000.



UPAEP – Secretaría General

Dirección General de Apoyos Académicos

Dirección del Centro de Recursos para el Aprendizaje y la Investigación.

Biblioteca Central - **Karol Wojtyla**

Tesis Digitales Restricciones de uso:

DERECHOS RESERVADOS ©

PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de textos, imágenes, gráficas, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente de donde la obtuvo mencionando el autor o autores involucrados en el documento.

Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Al menos 80% de las hojas utilizadas hasta la culminación de este trabajo fue papel reciclado.

A la memoria de mi abuelito Antonio y mi abuelita Ausencia. (...)

A mi mamá. (...)

A Karime. (...)

A B-P-M.

Al ente creador del universo, en especial del antaño (más de 10000 años) paraíso, la Tierra.

AGRADECIMIENTOS

Gracias a Marco A. Carrasco, Elí M. Manjarrez, Roberto Salazar y Juan Carlos Alejo por su tiempo.

TABLA DE CONTENIDO

INTRODUCCIÓN	7
CAPÍTULO 1. Programación en Windows	9
El Modelo de Programación de Windows	9
Procesamiento de mensajes	10
La Interfaz de Dispositivos para Gráficos (GDI)	15
Programación Basada en Recursos	18
Manejo de la Memoria	23
Bibliotecas de Enlace Dinámico (DLLs)	23
Interfaz de Programación de Aplicaciones (API) Win32	24
Notación Húngara	24
Diferencias entre Windows de 16 bits y Windows de 32 bits	26
Multitarea	29
Microsoft Visual C ++	30
Historia de Visual C++	31
Visual C++ de 16 y 32 bits	32
CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits	33
Arquitectura de Windows 95	33
Administrador de Máquinas virtuales	33
Dispositivos Virtuales	34
Controladores de Dispositivo	35
Bibliotecas de Enlace Dinámico (DLLS)	35
Aplicaciones basadas en MS-DOS	36
Aplicaciones basadas en Windows	37
Mapas de Bits	37
Tipos de Mapas de Bits	38
CAPÍTULO 3. Bibliotecas de Enlace Dinámico	43
Definición	44
Tipos de Bibliotecas	45
Biblioteca de objetos	45
Biblioteca de importaciones	45
Usos de las bibliotecas	45
Tipos de DLLs	46
DLLs con código	46
DLLs sólo de recursos	46
Usos de los DLLs	46
Tipos de enlace	46
Enlace Estático	47
Enlace Dinámico	47
Determinar que método de enlace usar	53
Operaciones de los DLLs	54
Exportar de un DLL	54
Importar de un DLL a una aplicación usando __declspec(dllimport)	61

Enlazar un ejecutable a un DLL	61
Inicializar un DLL	61
Crear un DLL de recursos	66
Recomendaciones para usar cada tipo de DLL	67
Archivos redistribuibles.....	68
Diferencias Entre Bibliotecas de Enlace Estático y los DLLs de Windows.....	69
Diferencias entre las aplicaciones y los DLLs.....	69
Ventajas y Desventajas de los DLLs.....	70
Trayecto de búsqueda usado por Windows para localizar un DLL.....	72
CAPÍTULO 4. Desarrollo de DLLs.....	73
Desarrollo de DLLs usando Visual C++	73
DLLs regulares.....	74
DLLs Extensión.....	77
DLLs sin MFC	79
Pasos para crear un DLL en Visual C++	81
MFC AppWizard DLL	81
Win32 DLL	81
Pasos para crear una aplicación MFC	82
CAPÍTULO 5. Desarrollo de la aplicación	87
DLLs para mostrar imágenes en una aplicación generada en Visual C++.....	87
Construcción del DLL y de la aplicación CLIENTE	87
CONCLUSIONES	108
Apéndices	109
Apéndice A – RETROSPECTIVA.....	109
Apéndice B - Referencia de DUMPBIN	112
Línea de comandos de DUMPBIN.....	112
Opciones de DUMPBIN	112
Apéndice C – Nombres decorados.....	118
Uso de los nombres decorados	118
Formato de un nombre decorado C++	118
Formato de un nombre decorado en C	119
Forma de ver los nombres decorados	119
Apéndice D – Archivos de Definición de Módulos (.DEF)	120
Reglas para los enunciados de definición de módulos	120
Glosario.....	124
Términos no alfabéticos.....	124
BIBLIOGRAFIA	155

INTRODUCCIÓN

Este trabajo recepcional tiene por objeto dejar una guía para el desarrollo de DLLs. Esta investigación está disponible para aquellos estudiantes que desarrollen software y necesiten modularizar su aplicación y reutilizar código.

Es importante tener en cuenta que para desarrollar aplicaciones en Visual C++ y, peor aún, para desarrollar DLLs en Visual C++, es necesario tener conocimientos de lenguaje C, programación orientada a objetos (si se va a usar MFC) y Windows en general.

El desarrollo del DLL se llevó a cabo usando Visual C++. Esta versión tiene entre sus principales características la creación de aplicaciones de 32 bits que pueden ser usadas con el sistema operativo Windows 9x y versiones posteriores.

La aplicación cliente, donde el DLL se ejecutará, se desarrolló también en Visual C++ y hace llamadas a funciones y recursos guardados en un DLL. El proceso pudiera pasar inadvertido para la persona que ejecute el programa, pero el DLL funciona de manera transparente para el usuario y éste lo notará sólo si el DLL no se encuentra disponible. Por consiguiente, no se ejecutarán las acciones deseadas.

El DLL internamente inserta una imagen que lee de un disco y la despliega en la pantalla. Dentro del DLL se encuentra todo el código necesario para realizar este proceso, por lo que esta acción queda aislada del desarrollo de la aplicación cliente.

Este trabajo consta de los siguientes capítulos:

Capítulo 1: Programación en Windows

En esta parte se explican las características del modelo de programación de Windows, la notación húngara, los recursos de Windows y la diferencia de Windows de 16 y 32 bits. Estos conceptos ayudarán a entender mejor el funcionamiento del sistema operativo.

Capítulo 2: Introducción a los dispositivos virtuales y Mapas de bits

En esta sección se explican los conceptos básicos de las distintas formas de DLLs que existen en Windows, como son los dispositivos virtuales. Dentro de ellos, se encuentran elementos que forman la arquitectura de Windows. Los VMMs, VxDs, Memoria de sólo lectura, DRVs, DLLs, y aplicaciones basadas en MS-DOS. También se incluye una visión general de los Mapas de bits y conceptos básicos de funcionamiento.

Capítulo 3: Bibliotecas de Enlace Dinámico

Aquí se detallan los principios fundamentales de los DLLs, incluyendo tipos, usos, ventajas, etc.

Capítulo 4: Desarrollo de DLLs

Este capítulo describe los pasos que deben seguirse para construir cada tipo de DLL. También, contiene la descripción y el código fuente los DLLs desarrollados para este trabajo y la aplicación desarrollada para llamarlos, junto con el código fuente.

Capítulo 5: Desarrollo de la aplicación

Finalmente, aquí se presenta una descripción de la aplicación generada para la presentación del trabajo recepcional.

CAPÍTULO 1. Programación en Windows

Dado el objetivo principal de esta tesis, que es el de realizar un DLL en Visual C++ para leer imágenes de un disco e ilustrar su funcionamiento, se presentan en este capítulo los conceptos básicos y fundamentales del ambiente de programación en Windows utilizando Visual C++.

EL MODELO DE PROGRAMACIÓN DE WINDOWS

En los casi catorce años desde su liberación en el otoño de 1985, Windows se ha establecido como norma en la industria de ambientes gráficos para computadoras personales construidas para los microprocesadores Intel. El sistema operativo Windows tiene la reputación de ser fácil para los usuarios pero difícil para los programadores. La curva de aprendizaje es larga debido a todas las características que ofrece.

La programación en Windows es diferente del viejo estilo de programación orientado a lotes. Para empezar, es necesario saber algunas características generales en la programación de aplicaciones Windows. Como marco de referencia, se usará el modelo de programación del conocido MS-DOS.

Esta sección contiene los siguientes temas:

- Procesamiento de mensajes
- La Interfaz de Dispositivos para Gráficos (GDI)
- Programación Basada en Recursos
- Manejo de la Memoria
- Bibliotecas de Enlace Dinámico (DLLs)
- Interfaz de Programación de Aplicaciones (API) Win32
- Notación Húngara

PROCESAMIENTO DE MENSAJES

DEFINICIÓN DE MENSAJE

Un mensaje es un paquete de datos, estructura o conjunto de parámetros usado para comunicar o solicitar información. Los mensajes pueden ser usados entre el sistema operativo y una aplicación, diferentes aplicaciones, hilos dentro de una aplicación y ventanas dentro de una aplicación.

FUNCIONAMIENTO DE LOS MENSAJES

En MS-DOS..

Cuando se escribe un programa para MS-DOS en lenguaje C, por ejemplo, sólo se necesita la función *main*. Esta función es la encargada de comenzar la ejecución de las líneas contenidas dentro de sus límites. Cuando se ejecuta un programa, el sistema operativo llama a esta función.

Si un programa necesita captar las teclas que oprime el usuario, llama al sistema operativo para que lo haga. Así mismo, si el programa requiere usar los servicios del sistema operativo, el programa llama a una función contenida en una biblioteca para que realice esta tarea.

En Windows...

Cuando el sistema operativo Windows ejecuta un programa, llama a la función *WinMain*, propia de la programación en Windows que es equivalente a la función *main* del DOS.

Las aplicaciones desarrolladas para correr en Windows no hacen llamadas explícitas a funciones (como *getchar()*) para obtener entrada del usuario. En vez de eso, esperan a que Windows les pase la entrada.

Windows pasa toda la entrada de una aplicación (entrada del usuario) a sus distintas ventanas. Cada ventana tiene una función, llamada *procedimiento de la ventana*, la cual Windows llama cuando tiene entrada para la ventana. Windows pasa la entrada al procedimiento de la ventana en forma de mensajes.

Los mensajes son generados por Windows y por las aplicaciones. Cuando el procedimiento de la ventana termina de procesar la entrada regresa el control a Windows.

CAPÍTULO 1. Programación en Windows

Windows genera un mensaje en cada evento de entrada —por ejemplo, cuando el usuario escribe, mueve el ratón, o cuando oprime un control como una barra de desplazamiento. Windows también genera mensajes en respuesta a cambios en el sistema producidos por cualquier aplicación, como cuando una aplicación cambia el tamaño de una de sus ventanas. Una aplicación puede generar mensajes para ordenar a sus propias ventanas que ejecuten tareas o se comuniquen con ventanas en otras aplicaciones.

Windows manda un mensaje a un procedimiento de la ventana con cuatro parámetros: un apuntador (handle) a una ventana, un identificador del mensaje, y dos valores de 32 bits llamados parámetros del mensaje para determinar a qué procedimiento de ventana va a mandar el mensaje.

Un indicador o apuntador ("handle") es simplemente un número (generalmente de 32 bits) que se refiere a un objeto.

Procedimiento de la ventana

Es una función que procesa todos los mensajes mandados o puestos a la ventana. Todos los aspectos de la apariencia y comportamiento de una ventana dependen de la respuesta del procedimiento de la ventana a estos mensajes.

Cada ventana es un miembro de una clase ventana en particular. La clase de la ventana determina el procedimiento de la ventana predeterminado que una ventana individual usa para procesar sus mensajes. Todas las ventanas que pertenecen a la misma clase usan el procedimiento de la ventana predeterminado para responder a los mensajes.

Identificador de mensaje

Es una constante fija que identifica el propósito del mensaje. Cuando un procedimiento de ventana recibe un mensaje, usa un identificador de mensaje para determinar como procesar el mensaje.

Ejemplo: El identificador de mensaje WM_PAINT le dice al procedimiento de ventana que el área cliente de la ventana ha cambiado y debe ser repintado.

Parámetros del mensaje

Especifican datos o la ubicación de los datos que serán usados por un procedimiento de ventana cuando se vaya a procesar el mensaje. El significado y valor de los parámetros del mensaje dependen del mensaje. El tipo de datos puede ser un entero, banderas de bits empaquetadas, un apuntador a una

estructura conteniendo datos adicionales y otros. Los parámetros pueden llevar datos como las coordenadas del cursor, códigos de teclas y otros.

Cuando un mensaje no usa los parámetros del mensaje, generalmente se igualan a NULL. Un procedimiento de ventana debe revisar el identificador del mensaje para determinar como interpretar los parámetros del mensaje.

ENVÍO DE MENSAJES

Windows usa dos métodos para enviar mensajes a un procedimiento de una ventana:

- Mensajes puestos en una cola. Los mensajes se ponen en una cola FIFO (el primero en entrar, es el primero en salir) llamada *cola de mensajes* para ser procesados según vayan entrando.
- Mensajes no puestos en una cola. Los mensajes se envían directamente al procedimiento de la ventana.

Mensajes puestos en una cola (Queued Messages)

Una cola de mensajes es un objeto de memoria definido por el sistema que guarda mensajes temporalmente. Los mensajes puestos en una cola de mensajes son principalmente el resultado de la entrada del usuario a través del ratón o el teclado, como los mensajes WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN, y WM_CHAR. Otros mensajes puestos en la cola incluyen el temporizador o reloj (timer), mensajes para pintar y salir: WM_TIMER, WM_PAINT y WM_QUIT. La mayoría de los otros mensajes, los cuales son mandados directamente a un procedimiento de la ventana, son llamados mensajes no puestos en la cola.

Con la excepción del mensaje WM_PAINT, Windows siempre pone los mensajes al final de la cola de mensajes. Esto asegura que la ventana reciba sus mensajes de entrada en la secuencia de el primero que entra es el primero que sale. El mensaje WM_PAINT se guarda en la cola y se reenvía al procedimiento de la ventana sólo cuando la cola no contiene otros mensajes. Cuando hay mensajes WM_PAINT múltiples para la misma ventana se combinan en un solo mensaje WM_PAINT, consolidando todas las partes inválidas del área cliente en una sola área. La combinación de mensajes reduce el número de veces que una ventana debe redibujar el contenido del área cliente.

El sistema pone un mensaje en la cola de mensajes de un hilo llenando una estructura MSG y luego copiándolo a la cola de mensajes. La información en la estructura MSG incluye: un apuntador a la ventana para la cual el mensaje se planea, el identificador del mensaje, los dos parámetros del mensaje,

CAPÍTULO 1. Programación en Windows

la hora en que el mensaje se puso, y la posición del cursor del ratón. Un hilo puede poner un mensaje a su propia cola de mensajes o a la cola de otro hilo usando la función `PostMessage` o `PostThreadMessage`.

Una aplicación puede quitar un mensaje de su cola usando la función `GetMessage`. Para examinar un mensaje sin removerlo de su cola, se usa la función `PeekMessage`. Esta función llena una estructura `MSG` con información sobre el mensaje.

Después de quitar un mensaje de su cola, una aplicación puede usar la función `DispatchMessage` para indicar a Windows que mande el mensaje a un procedimiento de la ventana para procesarlo. La función `DispatchMessage` toma un apuntador a una estructura `MSG` que fue llenada por una llamada previa a la función `GetMessage` o `PeekMessage`. La función `DispatchMessage` pasa el apuntador a la ventana, el identificador del mensaje y dos parámetros de mensajes al procedimiento de la ventana, pero no pasa la hora en que el mensaje fue puesto o la posición del cursor del ratón. Una aplicación puede regresar esta información llamando las funciones `GetMessageTime` y `GetMessagePos` mientras se procesa un mensaje.

Windows puede desplegar cualquier tipo de ventanas en un tiempo determinado. Para enviar la entrada del ratón y del teclado a la ventana apropiada, Windows usa colas de mensajes. Cuando el usuario mueve el ratón, oprime los botones del ratón, o escribe en el teclado, el controlador del dispositivo para el ratón o el teclado convierte la entrada en mensajes y los coloca en a cola de mensajes del sistema. Windows quita los mensajes, uno por uno, de la cola de mensajes del sistema, los examina para determinar la ventana de destino, y luego los pone en la cola de mensajes del hilo que creó la ventana de destino. Una cola de mensajes de un hilo recibe todos los mensajes del ratón y del teclado para las ventanas creadas por el hilo. El hilo quita los mensajes de su cola y le ordena a Windows que los mande al procedimiento de ventana apropiado para procesarlos.

Mensajes no puestos en la cola (Nonqueued Messages)

Estos mensajes son enviados inmediatamente al procedimiento de ventana de destino, pasando por alto la cola de mensajes del sistema y la cola de mensaje del hilo. Windows manda estos mensajes para notificarle a una ventana de eventos que pueden afectarla.

Ejemplo: Cuando el usuario activa la nueva ventana de la aplicación, Windows le manda a la ventana una serie de mensajes, incluyendo `WM_ACTIVATE`, `WM_SETFOCUS` y `WM_SETCURSOR`.

Estos mensajes notifican a la ventana que ha sido activada, que la entrada del teclado está siendo dirigida a la ventana, y que el cursor del ratón ha sido movido a los bordes de la ventana. Los mensajes no enviados a la cola también pueden resultar cuando una aplicación llama a ciertas funciones de Windows.

Ejemplo: Windows manda el mensaje WM_WINDOWPOSCHANGED después de que una aplicación usa la función SetWindowPos para mover una ventana.

MANEJO DE MENSAJES

Una aplicación debe quitar y procesar mensajes puestos en la cola de mensajes de sus hilos de ejecución. Una aplicación de un solo hilo generalmente usa un *ciclo de mensajes* en su función *WinMain* para quitar y enviar mensajes a los procedimientos de ventana apropiados para procesarse. Las aplicaciones con múltiples hilos pueden incluir un ciclo de mensajes para cada hilo que cree una ventana.

Ciclo de mensajes

Un ciclo de mensajes simple consiste en un llamado a una función para cada una de estas tres funciones: GetMessage, TranslateMessage y DispatchMessage.

GetMessage: Regresa un mensaje de la cola y lo copia a una estructura MSG. Esta función regresa TRUE a menos que encuentre el mensaje WM_QUIT, en cuyo caso regresa FALSE y termina el ciclo. En una aplicación de un solo hilo, el terminar el ciclo de mensajes es generalmente el primer paso para cerrar la aplicación.

Si se especifica un apuntador (handle) en GetMessage, solamente los mensajes para la ventana especificada se recuperan de la cola.

TranslateMessage: Un ciclo de mensajes de un hilo debe incluir esta función si el hilo va a recibir entrada de caracteres del teclado. Windows genera mensajes de teclas virtuales (WM_KEYDOWN y WM_KEYUP) cada vez que el usuario presione una tecla. Un mensaje de tecla virtual contiene un código de una tecla virtual que identifica la tecla fue presionada, pero no el valor de su carácter. Para recuperar este valor, el ciclo de mensajes debe contener TranslateMessage, el cual convierte el mensaje de la tecla virtual en un mensaje de carácter (WM_CHAR) y lo coloca de regreso en la cola de mensaje de la aplicación. El mensaje de carácter puede ser removido en una iteración subsecuente del ciclo de mensajes y despachado en un procedimiento de ventana.

DispatchMessage: Manda un mensaje al procedimiento de ventana asociando con el apuntador (handle) a la ventana especificado en la estructura MSG. Si el apuntador a la ventana es HWD_TOPMOST, DispatchMessage manda el mensaje los procedimientos de las ventanas de todas las ventanas del nivel más alto en el sistema. Si el apuntador de la ventana es NULL, DispatchMessage hace nada con el mensaje.

LA INTERFAZ DE DISPOSITIVOS PARA GRÁFICOS (GDI)

El GDI es un DLL que procesa llamadas a funciones de gráficos de una aplicación Windows y pasa esas llamadas al controlador del dispositivo apropiado. Los servicios del GDI son proporcionados por los DLLs GDI.EXE y GDI.DLL los cuales contienen las partes de 16 y 32 bits del GDI respectivamente.

Muchos de los programas para MS-DOS escriben directamente a la memoria de video y al puerto de la impresora. La desventaja de esta técnica es la necesidad de proveer un controlador de software para cada tarjeta de video y para cada modelo de impresora.

Windows introdujo una capa de abstracción llamada Interfaz de Dispositivos Gráficos (Graphics Device Interface). El GDI es el componente gráfico de Windows. Se comunica entre la aplicación y el controlador del dispositivo, el cual ejecuta funciones del hardware específicas que generan salida. Al actuar como una memoria intermedia (buffer) entre las aplicaciones y los dispositivos de salida, el GDI presenta una vista del mundo *independiente de dispositivos* para la aplicación, al mismo tiempo, interactúa con el dispositivo en un formato *dependiente del dispositivo*.

En el ambiente GDI existen dos espacios de trabajo —el lógico y el físico. En el *espacio lógico* se encuentran las aplicaciones, este es un mundo "ideal" en el cual todos los colores están disponibles, todas las fuentes son escalables y la resolución de la salida es fenomenal. Por otro lado, el *espacio físico*, es el mundo real de los dispositivos, con colores limitados, formatos de salida extraños y capacidades de dibujo diferentes. En Windows, una aplicación no tiene que entender la peculiaridad de un dispositivo nuevo. El código del GDI trabaja en el dispositivo nuevo si el dispositivo tiene un controlador.

Windows proporciona los controladores de video e impresoras, de esta manera los programas y los desarrolladores de aplicaciones no necesitan saber el tipo de tarjeta de video que está unida al sistema. En vez de direccionar el hardware, el programa llama a las funciones del GDI que hacen referencia a una estructura de datos llamada contexto del dispositivo (device context). La principal ventaja es la independencia de dispositivos.

CONTEXTO DEL DISPOSITIVO

El contexto del dispositivo (Device context o DC) es una estructura que define un conjunto de objetos gráficos y sus atributos asociados, y los modos gráficos que afectan la salida.

Los objetos gráficos incluyen:

- Plumas para dibujar líneas
- Brochas para pintar y llenar
- Mapas de bits para copiar o desplazar partes de la pantalla
- Paletas para definir el conjunto de colores disponibles
- Regiones para recortar y otras operaciones

A diferencia de la mayoría de las estructuras Win32, una aplicación nunca tiene acceso directo al contexto del dispositivo, en vez de eso, la aplicación hace operaciones con la estructura indirectamente llamando varias funciones.

Windows mapea la estructura del contexto del dispositivo a un dispositivo físico y emite las instrucciones de entrada/salida apropiadas. El GDI es casi tan rápido como el acceso directo al video, además, permite a diferentes aplicaciones escritas para Windows compartir la pantalla.

Las aplicaciones basadas en Win32 pueden dibujar e imprimir salida en una variedad de dispositivos. El software que soporta esta independencia de dispositivos está contenido en dos bibliotecas de enlace dinámico. Una es GDI.DLL o GDI.EXE.

El nombre de la segunda depende del dispositivo donde la aplicación dibuja la salida. Por ejemplo, si una aplicación dibuja la salida en un área cliente de su ventana en un monitor VGA, la biblioteca se llama VGA.DLL, si la aplicación imprime a una impresora Epson FX-80, la biblioteca se llamara EPSON9.DLL.

PRIMITIVAS DEL GDI

El GDI proporciona funciones y estructuras relacionadas que puede usar una aplicación para generar salida gráfica a monitores, impresoras y otros dispositivos. Los conceptos GDI mapeados entre el espacio lógico y el físico son primitivas y otros conceptos listados a continuación:

CAPÍTULO 1. Programación en Windows

- **Líneas y curvas:** Las líneas son la base de cualquier sistema de dibujo de gráficas de vectores. El GDI soporta líneas rectas, rectángulos, elipses (incluyendo el círculo), arcos que son curvas parciales de la circunferencia de una elipse y curvas Bezier
- **Áreas rellenas:** Cuando una serie de curvas o líneas encierra un área, esa área puede ser rellena con el objeto brocha actual del GDI. Esta brocha puede ser de color sólido, un patrón, o una imagen de bits que se repite vertical u horizontalmente dentro del área.
- **Mapas de bits:** Las imágenes de este formato pueden ser usadas para mostrar imágenes del mundo real que deben ser pintadas rápidamente.
- **Texto:** Puede contener fuentes True Type o fuentes basadas en mapas de bits.

Otros aspectos del GDI que no son tan fáciles de clasificar incluyen:

- **Modos de mapeo y transformaciones:** Por defecto se dibuja en píxeles, pero el GDI permite dibujar en pulgadas o en milímetros.
- **Metaarchivos:** Es una colección de comandos GDI almacenados de forma binaria. Los metaarchivos se usan principalmente para transferir representaciones de dibujos de gráficas de vectores a través del portapapeles.
- **Regiones:** Una región es un área compleja de cualquier forma, y se define generalmente como una combinación booleana de regiones más simples. Las regiones se guardan por dentro en el GDI como una serie de líneas rastreadas de cualquier combinación de líneas que pueden haber sido usadas para definir la región original.
- **Trayectos (paths):** Es una colección de líneas rectas y curvas guardadas internamente en el GDI. Los trayectos pueden ser usados para dibujar, rellenar y recortar. También pueden ser convertidos en regiones.
- **Recortes (clipping):** El dibujo puede estar limitado a una sección en particular del área cliente. Esto se conoce como recorte, y el recorte puede ser un área rectangular o un área que se puede definir matemáticamente como una serie de líneas cortas. El recorte se define por una región o un trayecto.
- **Paletas:** Estructuras que permiten manejar diferentes combinaciones de colores en el sistema operativo.

El color y el estilo de los elementos dependen del objeto con que se dibujen. Se pueden crear y utilizar plumas, fuentes y brochas para cambiar la apariencia de los objetos. Las plumas sirven para dibujar líneas y curvas, las brochas para llenar el interior de las figuras cerradas y las fuentes para escribir texto.

CAPÍTULO 1. Programación en Windows

Los objetos se convierten de objetos lógicos a objetos físicos usando un proceso de realización. Por ejemplo, una aplicación crea una pluma lógica llamando a la función `CreatePen` con los parámetros apropiados. Cuando la pluma lógica se selecciona en un contexto del dispositivo (DC) usando `SelectObject`, el GDI convierte la pluma lógica en un objeto físico que se usa para comunicarse con el dispositivo. El GDI pasa el objeto lógico al dispositivo, y el dispositivo crea un objeto específico del dispositivo que contiene información específica del dispositivo. Durante la conversión (realización), el color (lógico) requerido se mapea a los colores disponibles, las fuentes se igualan a las mejores fuentes disponibles, y los patrones se preparan para la salida. La selección de la fuente es más compleja que otras conversiones (realizaciones), y el GDI, no el controlador, ejecuta la mayoría del trabajo de conversión (realización). De la misma manera, la conversión de la paleta (usando `RealizePalette`, no `SelectPalette`) se hace dentro del GDI.

Las primitivas de salida son pasadas similarmente como solicitudes lógicas al controlador del dispositivo, el cual dibuja la primitiva lo mejor posible según su resolución y características. Si el controlador no puede manejar una cierta primitiva, por ejemplo, no puede dibujar una elipse, el GDI simula la operación. Para una llamada a `Ellipse`, el GDI calcula un polígono que representa una elipse digitalizada. Luego, el polígono resultante puede ser simulado como una polilínea. Aunque la aplicación no se ocupa de qué componentes del sistema hagan el trabajo, la primitiva se dibuja.

PROGRAMACIÓN BASADA EN RECURSOS

Para crear una programación manejada por datos en MS-DOS, se debe codificar los datos como constantes de inicialización o proporcionar archivos de datos separados del programa para leer. Cuando se programa en Windows, se almacenan los datos en un archivo de recursos usando un número de formatos preestablecidos. El Enlazador (linker) combina este archivo de recursos binarios con la salida del compilador de C++ para generar un programa ejecutable. Los archivos de recursos pueden incluir mapas de bits, iconos, definiciones de menús, esquemas de cajas de diálogo, y cadenas. Incluso pueden incluir formatos de recursos personalizados que uno defina. Para editar estos recursos se utiliza un editor de recursos generalmente incluido con el compilador.

RECURSOS DE WINDOWS

Los recursos son datos binarios que un compilador de recursos o un desarrollador guarda en un archivo ejecutable de una aplicación o en una Biblioteca de Enlace Dinámico. Los recursos pueden ser utilizados para guardar datos específicos y características propias para la interfaz de una aplicación.

Además, el uso de recursos permite que los datos permanezcan ocultos al usuario y en determinado momento integrar versiones actualizadas de los recursos.

Un recurso puede ser de dos tipos:

- Personalizado
- Predeterminado

Un recurso definido por el usuario, también llamado recurso personalizado, es conveniente para agregar datos de cualquier tipo a un archivo ejecutable o a un DLL y obtener acceso a los datos dentro del programa. Los datos pueden estar en cualquier formato, ya sea texto o binario. Las funciones de Windows usadas para acceder los recursos definidos por el usuario regresan un apuntador a los datos cuando Windows los carga en memoria. Se puede hacer lo que se desee

Ciertos tipos de recursos tienen formatos preestablecidos y reconocidos por Windows. Se pueden crear y editar estos recursos usando los editores de recursos incluidos en el SDK de Windows, en Visual C++ y en otros editores que soporten el formato del recurso.

Windows tiene soporte interno para los siguiente tipos de recursos:

- Iconos
- Cursores
- Mapas de bits
- Plantillas para cajas de diálogo
- Tablas de cadenas
- Plantillas para Menús
- Tabla de teclas aceleradoras

Iconos

Un icono es una pequeña imagen gráfica que representa una aplicación cuando su ventana principal está minimizada. Para la aplicación un icono es un tipo de recurso. Antes que los recursos sean compilados, cada icono es un archivo separado que contiene un conjunto de bits formando una imagen. Cuando una aplicación requiere un icono primero lo llama por su nombre, enseguida busca el recurso con

el nombre con el cual fue llamado y si hay más de un icono, Windows selecciona el que mejor se acomode a las preferencias del usuario.

Pueden especificarse iconos para una clase de ventana específica, desplegarse iconos personalizados o desplegar iconos en una caja de diálogo.

Cúrsores

El cursor es un mapa de bits especial que muestra al usuario donde tomarán lugar las acciones iniciadas por el ratón. Para la mayoría de las aplicaciones corriendo bajo Windows, el usuario hace selecciones, escoge comandos, y dirige otras acciones usando el ratón o el teclado. Es posible controlar el tamaño del cursor, usar cursores contenidos en Windows, usar una clase de cursor, desplegar diferentes cursores durante diferentes acciones, o usar cursores específicos cuando no hay ratón instalado, así como usar un cursor propio.

Mapas de bits

El término mapa de bits se refiere a una imagen formada por un conjunto de bits. Existen dos tipos de mapas de bits en Windows: mapas de bits dependientes del dispositivo y mapas de bits independientes del dispositivo.

Los mapas de bits dependientes del dispositivo (DDBs, Device Dependent Bitmaps) son un conjunto de bits en memoria, que pueden ser desplegados en un dispositivo de salida. Debido a que hay una correlación estrecha entre los bits de memoria y los píxeles en la pantalla, se dice que un mapa de bits en memoria es dependiente del dispositivo. Para tales mapas de bits, la manera en que los bits son puestos en memoria depende del dispositivo de salida.

El segundo tipo de mapas de bits, los mapas de bits independientes del dispositivo (DIBs, Device Independent Bitmaps) describen la apariencia de una imagen, más que la manera en que la imagen es representada internamente por un dispositivo de salida en particular.

En el capítulo 2 se describen los mapas de bits con más detalle.

Cajas de diálogo

Las cajas de diálogos son ventanas emergentes que interactúan con el usuario. Las cajas de diálogos son usadas generalmente para desplegar o pedir información necesaria para completar un

CAPÍTULO 1. Programación en Windows

comando. Una caja de diálogo puede contener uno o más controles (ventanas hijo) con los cuales el usuario puede introducir texto, escoger opciones o dirigir la acción de un comando en particular.

Windows proporciona muchas funciones, mensajes, y controles predefinidos para ayudar a crear y administrar las cajas de diálogo, así se facilita el desarrollo de la interfaz del usuario para una aplicación.

Windows también proporciona muchas cajas de diálogo predefinidas o comunes que soportan comandos, como la caja de diálogo común File Open y File Print. Las aplicaciones que usan estos comandos deben usar las cajas de diálogo comunes para pedir la entrada del usuario, sin importar el tipo de aplicación que lleva a cabo los comandos.

Tabla 1: Cajas de diálogo comunes en Win32

Color	Despliega todos los colores disponibles y permite que el usuario seleccione colores básicos o cree colores personalizados.
Fuente	Despliega listas de nombres de fuentes, tamaños de la fuente (en puntos), y otros atributos que el usuario puede seleccionar.
Abrir	Despliega nombres de archivos, directorios y unidades, y permite que el usuario seleccione una combinación de nombres que identifican el archivo que se va a abrir.
Guardar como	Despliega nombres de archivos, directorios y unidades, y permite que el usuario seleccione una combinación de nombres que identifican el archivo que se va a guardar.
Imprimir	Despliega y permite que el usuario seleccione las opciones para imprimir, como rango de páginas a imprimir y número de copias.
Configurar página	Despliega y permite que el usuario seleccione las opciones de configuración de la página, como orientación del papel, tamaño, fuente del papel, y márgenes.
Buscar	Despliega y permite que el usuario seleccione una cadena a buscar y opciones de búsqueda, como la dirección de la búsqueda.
Reemplazar	Despliega y permite que usuario seleccione cadenas a buscar y reemplazar y opciones para reemplazar, como la dirección de la búsqueda y alcance del reemplazo.

Fuente: MSDN Development Library – CD 12, July 1995.

49813

Tablas de cadenas

Los recursos tablas de cadenas o conjunto de cadenas (frases o enunciados) son usados principalmente para facilitar la traducción de un programa a otros idiomas. Si se usa una tabla de cadenas en vez de colocar las cadenas directamente en el código fuente, entonces todo el texto que usa el programa está en un archivo, el listado de recursos. Si el texto en este listado de recursos es traducido, todo lo que se necesita para crear una versión del programa en un idioma extranjero, es volver a ligar el programa y agregar los recursos traducidos a un archivo ejecutable. Este método es más seguro que modificar el código fuente con las cadenas incluidas. Una segunda razón para usar tablas de cadenas es reducir el espacio de memoria.

Menús

La mayoría de las aplicaciones de Windows usan menús para que el usuario pueda seleccionar y llevar a cabo comandos o acciones.

Un menú es una lista de elementos que son los comandos de una aplicación para un usuario. Cada elemento del menú puede ser desplegado como texto o como mapa de bits. Al escoger un elemento del menú (con el ratón o con el teclado), el usuario le dice a la aplicación que ejecute el comando asociado con ese elemento. Windows responde a esta acción mandando un mensaje a la aplicación que identifica el comando que escogió el usuario.

Para usar un menú en una aplicación, se siguen los siguientes pasos:

1. Definir el menú en el archivo de definición de recursos.
2. Especificar el menú en el código de la aplicación. Hay dos maneras de hacer esto. Cuando se registra la clase de la ventana, se especifica un menú (llamado por la clase menú) para esa clase de ventana. La segunda manera, cuando se crea una ventana y se especifica un menú para esa ventana.
3. Inicializar el menú si es necesario.
4. Puede especificarse un menú para una ventana específica, usarse mapas de bits como elementos de un menú, agregar teclas aceleradoras para menús y para sus elementos y desplegar menús en cascada.

Tabla de teclas aceleradoras

Es una tabla que contiene una lista de teclas o combinación de teclas que generan mensajes WM_COMMAND o WM_SYSCOMMAND. Se llaman así porque con solo oprimir la combinación de teclas, se ejecuta el comando asociado sin necesidad de seleccionarlo en la barra de menús, especialmente si está contenido dentro de varios submenús. A menudo, los programas usan tablas de teclas aceleradoras para duplicar la acción de las opciones en un menú (sin embargo las tablas de teclas aceleradoras pueden también ejecutar funciones no pertenecientes a un menú). Generalmente se usan las teclas Ctrl y Alt.

Los aceleradores están relacionados estrechamente con los menús —ambos proporcionan al usuario acceso a un conjunto de comandos de una aplicación.

MANEJO DE LA MEMORIA

Con cada versión de Windows, el manejo de la memoria se hace más fácil. Ahora, simplemente se asigna la memoria que se necesita y Windows se encarga de los detalles. Los esquemas de memoria segmentada no existen.

Características del manejo de la memoria en Windows de 32 bits:

- Independencia de arquitectura x86 en el manejo de la memoria.
- La memoria virtual es una vista lineal de la memoria física.
- Los programas de 32 bits pueden direccionar hasta 4 GB de memoria.
- Cada proceso tiene su propio espacio de direcciones privado, de esta manera los programas no pueden escribir fuera de su propio espacio de direcciones.

BIBLIOTECAS DE ENLACE DINÁMICO (DLLS)

En el ambiente MS-DOS, todos los módulos de un programa se enlazan estáticamente durante el proceso de construcción de la aplicación. Windows permite el enlace dinámico, el cual significa que bibliotecas especiales se construyen y se pueden cargar y enlazar en tiempo de ejecución. Muchas aplicaciones pueden compartir los DLLs, lo cual ahorra memoria y espacio en disco. El enlace dinámico incrementa modularidad de una aplicación porque se pueden compilar y probar los DLLs por separado (un DLL necesita de una aplicación para ejecutarse).

En el capítulo 3 se describen los DLLs con detalle.

INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API) WIN32

El API (Interfaz de Programación de Aplicaciones, Application Programming Interface) es un conjunto de rutinas que una aplicación usa para pedir y realizar servicios de bajo nivel ejecutados por el sistema operativo. En Windows, el API también apoya a las aplicaciones en el manejo de ventanas, menús, iconos y otros elementos de la interfaz gráfica de usuario.

Mientras que el API ya sea Win32 o Win64 es una interfaz de programación en C, la biblioteca MFC es una interfaz de programación en C++ para Windows. La primera usa programación normal en C y la segunda es un enfoque del API orientado a objetos.

El API Win32 ofrece muchas funciones nuevas, incluyendo funciones para la Entrada/Salida, la cual era manejada anteriormente por llamadas a MS-DOS.

MFC (Fundación de Clases Microsoft) es una biblioteca de clases que simplifica el tiempo de programación al crear una aplicación totalmente funcional usando un modelo Documento-Vista el cual separa los datos que maneja la aplicación en una clase Documento del despliegue de los datos en la pantalla a partir de una clase Vista.

NOTACIÓN HÚNGARA

La programación de una aplicación de tamaño considerable contiene identificadores de muchos caracteres, por ejemplo, los mensajes del sistema operativo Windows. Los desarrolladores deben usar nombres para lo que quieran identificar que no pertenezcan al sistema operativo y que a la vez les denoten las características principales. Esto obliga a tener una norma que distinga a varios elementos del sistema llevando un cierto orden.

Una forma de enfrentar este problema es hacer uso de una notación que permita identificar un nombre rápidamente. Existe una notación muy común en la programación en Windows: la notación húngara. Se nombró así en honor a un programador de Microsoft, Charles Simonyi, haciendo referencia a su nacionalidad.

La notación húngara es una convención de nombramiento de variables que hace más entendible el código a la hora de analizarlo, de esta manera, es más sencillo recordar el tipo de dato y hacer nombramientos aún cuando el nombre contenga dos tipos distintos.

CAPÍTULO 1. Programación en Windows

La notación húngara es sencilla, el nombre de la variable comienza con una letra minúscula o letras que denotan el tipo de dato de la variable. Por ejemplo, el prefijo *lpsz* en *lpszCmdParam* denota "apuntador largo a cadena terminada en cero (long pointer to a string terminated by zero)".

El prefijo *h* en *hInstance* y *hPrevInstance* denota "handle", el prefijo *n* en *nCmdShow* denota "number", y generalmente especifica un número entero. Dos de los parámetros de *WndProc* también usan la notación húngara: *wParam* es un UINT (la "w" denota "word" y UINT denota unsigned integer) y *lParam* es un LONG.

Cuando se nombran variables para una estructura, se puede usar el nombre de la estructura (o una abreviación del nombre de la estructura) en minúsculas o como un prefijo del nombre de la variable o como todo el nombre de la variable. Por ejemplo, en algunos programas la variable *msg* es una estructura de tipo MSG, otro ejemplo, *wndclass* es una estructura de tipo WINDCLASS.

La notación húngara ayuda a evitar errores en el código porque existe un orden para nombrar tipos de datos específicos antes de que se conviertan en basura debido a que el nombre de la variable describe el uso y tipo de dato. De esta manera, se pueden evitar muchos errores al codificar cuando se trata de tipos de datos diferentes. El uso de la notación húngara hace que sea más sencillo recordar variables y nombres para procedimientos, estructuras y parámetros. La notación húngara facilita la creación de nombres que describen a variables o tipos de datos.

Algunos prefijos usados en el libro Programación en Windows de Charles Petzold, se listan en la siguiente tabla:

Tabla 2: Prefijos de la Notación Húngara.

Prefijo	Tipo de Dato
c	char
by	BYTE (unsigned char)
n	short o int
i	int
x, y	short (usado como coordenada x o coordenada y)
cx, cy	short (usado como longitud x o y; la c denota "count")
b	BOOL (int)
w	UINT (unsigned int) o WORD (unsigned word)
l	LONG (long)
dw	DWORD (unsigned long)
fn	Function
s	string
Sz	string terminada en 0 bytes

Fuente: MSDN Development Library – CD 12, July 1995.

DIFERENCIAS ENTRE WINDOWS DE 16 BITS Y WINDOWS DE 32 BITS

Windows 3.x y Windows para Grupos de Trabajo son sistemas operativos de 16 bits los cuales pueden ser corridos en una PC con un microprocesador de 8 bits o de 16 bits, el más adecuado es un procesador de 16 bits porque aprovecha al máximo el ancho del bus de datos.

La diferencia más importante entre un microprocesador de 16 bits y uno de 32 bits radica en el tamaño del bus de datos. El bus de datos determina cuantos bytes son transferidos entre el microprocesador y la memoria al mismo tiempo; el bus de direcciones determina cuanta memoria es direccionada por cada microprocesador. Generalmente, un bus de datos más grande da como resultado un microprocesador más rápido. En la siguiente tabla se listan algunas características de algunos procesadores de la familia Intel en cuanto al tamaño del bus de datos.

Tabla 3: Comparación de procesadores.

Procesador	Ancho del Bus de Datos	Direccionalamiento máximo de memoria
8048	8	2K interno
8051	8	8k interno
8085A	8	64K
8086	16	1M
8088	8	1M
8096	16	8K interno
80186	16	1M
80188	8	1M
80286	16	16M
80386DX	32	4G
80386SL	16	32M
80386SX	16	16M
80486DX	32	4G
80486SX	32	4G
PENTIUM	32/64	4G

Fuente: MSDN Development Library – CD 12, July 1995.

Windows 95 y Windows NT son versiones de 32 bits de Windows. Ambos están diseñados para microprocesadores con un bus de datos de 32 bits, en el cual pueden explotar todo el poder de esta arquitectura. Windows 95 puede ejecutarse en arquitecturas de 16 bits, pero el rendimiento es inferior.

Windows 95 y Windows NT usan el nuevo API de 32 bits. Las aplicaciones modificadas para este API pueden aprovechar características avanzadas que se listan a continuación:

Tabla 4: Características de Windows de 32 bits.

Arquitectura del sistema de archivos en modo protegido y arquitectura de dispositivos
Controladores de dispositivos cargables y descargables para reducir el conjunto de trabajo
Gráficas nuevas, comunicaciones, y componentes de redes

CAPÍTULO 1. Programación en Windows

Programación preferente de aplicaciones de 32 bits y manejo de hilos
Colas de entrada asíncronas con soporte de entrada por prioridad.
Soporte para múltiples hilos de ejecución
Soporte para diferentes procesadores
Proceso distribuido
Redes
Mayor seguridad
Soporte para aplicaciones basadas en caracteres, ejecutándose en una ventana
GDI mejorado para dibujar curvas complejas

Fuente: MSDN Development Library – CD 12, July 1995.

Una de las principales ventajas del API de 32 bits es el rompimiento de la barrera de los 64 KB. Antes de su creación, el software y los sistemas operativos nunca se habían igualado al desempeño del hardware. Cuando se desarrolla una aplicación Win32 se usa todo el potencial del hardware en cuanto al ancho del bus de datos. Un programa Win32 puede direccionar hasta 4GB de memoria.

En la siguiente tabla se detallan características específicas que soporta cada sistema operativo de 16 y 32 bits.

Tabla 5: Win32 en los diferentes sistemas operativos

	Windows 3.1 / WFW 3.1 Con Win32s	Windows 95	Windows NT
Win16 (extendido a 32 bits)	√	√	√
Administración de memoria de 32 bits	√	√	√
Mapeo de archivos	√	√	√
Redes (NetBIOS, Windows sockets)	√	√	√
OLE 2.0	√	√	√
Mailslots and Named Pipes		√	√
Win32 threading		√	√
Advanced GDI (beziers and paths)		√	√
Llamadas a procedimientos remotos (RPCs)		√	√
Transformaciones GDI			√
Seguimiento de eventos (event logging)			√
Seguridad			√
UNICODE			√

Fuente: MSDN Development Library – CD 12, July 1995.

El API de 32 bits de Windows soporta archivos de mapeo a memoria. Esta poderosa facilidad permite a un archivo guardado en disco ser asignado a un rango de direcciones de memoria virtual. El contenido del archivo puede ser leído o escrito simplemente accediendo a la memoria asignada. El archivo permanece residente en disco hasta que se acceda en realidad. Cuando se acceda, si una sección no está presente, una pieza pequeña del archivo es paginado a memoria, esta operación es invisible a la aplicación.

MULTITAREA

La característica más importante entre Windows de 16 bits y Windows de 32 bits, como en el caso de Windows NT, es la multitarea. Aunque Windows de 16 bits puede ejecutar diferentes aplicaciones simultáneamente, al ejecutar las aplicaciones no lo hace de una manera preferente. Existen dos tipos de multitarea que a continuación se describen:

MULTITAREA COOPERATIVA

Tipo de multitarea en la cual el proceso que controla el CPU actualmente debe ofrecer control a otros procesos. Dicho proceso o programa, puede controlar el CPU durante todo el tiempo que lo necesite. Si un programa no está usando el CPU, de cualquier modo, puede permitir a otro programa usar el CPU temporalmente. Se llama cooperativo porque todos los programas deben cooperar para trabajar. Si un programa no coopera, puede acaparar el CPU. En contraste, la multitarea preferente o preventiva fuerza a las aplicaciones a compartir el CPU, lo quieran o no. Los sistemas operativos Windows 3.x y el MultiFinder (para la computadoras Macintosh) usan multitarea cooperativa.

Los desarrolladores de aplicaciones para Windows de 16 bits reconocen esto y tratan de implementar sus aplicaciones para que las tareas se puedan ejecutar en cascada. Por ejemplo, un programa para dar formato a discos pudiera formatear una pista de un disco y luego regresar el control a Windows. Una vez que Windows tiene el control, puede responder a otras tareas para el usuario. Cuando el usuario se encuentre inactivo, Windows regresará el control al programa que da formato y de esta manera a otra pista se le puede dar formato.

MULTITAREA PREFERENTE

El término *multitarea preferente* se refiere a una técnica de compartir tiempo de procesamiento con todos los programas de ejecución. Esto es, una aplicación debe decirle a Windows que ha terminado de procesar antes que el planeador de Windows pueda asignar a otra aplicación tiempo de ejecución, lo cual crea problemas para los usuarios y los desarrolladores.

Este tipo de multitarea es la característica más importante de Windows 95 (Windows NT es un sistema operativo más robusto que soporta características más avanzadas como soporte para varios procesadores, entre otras).

Para los usuarios, esto significa que el control del sistema es perdido por un periodo de tiempo arbitrario que es establecido por la aplicación (no por el usuario).

MICROSOFT VISUAL C ++

El término Visual en Visual C++ puede ser engañoso. Si no se conoce el producto, al principio se piensa que es parecido a Visual Basic, pero no es así. Para desarrollar aplicaciones en Visual C++, se

debe entender y escribir código C/C++, si se utiliza una biblioteca como MFC o C si uno utiliza el API de Windows. Al usar MFC por ejemplo, involucra saber programación orientada a objetos.

Al usar MFC, Visual C++ tiene asistentes (Wizards) que permiten crear el esqueleto de la nuestra aplicación pero se debe entender el código que generan los asistentes, la estructura de la biblioteca MFC y algunos conceptos internos del sistema operativo Windows.

HISTORIA DE VISUAL C++

Visual C++ es el resultado de la evolución del lenguaje C desarrollado por parte de Microsoft. En sus inicios, tuvo el nombre de Microsoft C. Después, en abril de 1992, apareció Microsoft C/C++ 7.0, el cual incluía el desarrollo de la biblioteca de clases propia de Microsoft llamada MFC.

En febrero de 1993 se lanzó Visual C++ 1.0 (16 bits) el cual incluía MFC versión 2.0. En agosto de 1993 apareció la Visual C++ (versión 32 bits) con MFC 2.0. En versiones subsecuentes se incluyeron características más sofisticadas.

Para los programadores de C++, el reuso de componentes generalmente significa la técnica de derivar nuevas clases de C++ de una clase base preexistente. El sistema de desarrollo Microsoft Visual C++ 1.0 usó esta técnica, con clases base proporcionadas en MFC, y Wizards para hacer fácil la derivación de clases. El éxito de Visual C++ 1.0 ha hecho este el mecanismo de reuso más popular para los programadores de C++. Versiones subsecuentes de Visual C++ antes de la versión 2.0 incluían mejoras a MFC.

Visual C++ versión 2.0 fue la primera versión de 32 bits, la cual incluía soporte a Windows NT 3.5. Con cada versión nueva, a Visual C++ y la biblioteca MFC se le añaden nuevas clases que incorporan soporte a características recientes según las tecnologías más recientes desarrolladas por Microsoft.

La Galería de componentes es otra característica de Visual C++ donde se guardan y administran los componentes, incluyendo clases de C++, controles OLE, y docenas de componentes de terceros que permiten agregar características avanzadas.

Una forma de acelerar el desarrollo de aplicaciones es usar la biblioteca Fundación de Clases Microsoft (MFC). MFC en su versión 4.0 contiene más de 150 clases, incluyendo controles basados en Windows para crear aplicaciones que aprovechen la interfaz de Windows 95.

Para aplicaciones de bases de datos, se incluye clases de Objetos de Acceso a Datos (DAO) para un acceso rápido al motor de base de datos orientada a objetos JET. Y apoyo a ODBC para crear aplicaciones de bases de datos independientes.

VISUAL C++ DE 16 Y 32 BITS

Las versiones recientes de Visual C++ son herramientas diseñadas para ayudar a acelerar el desarrollo de una aplicación para los ambientes Windows 95, Windows NT, y Windows 3.1.

La diferencia entre las versiones de 16 y 32 bits de Visual C++ radican en la interfaz gráfica, y en el soporte para 16 y 32 bits. Mientras que la versión de 16 bits permite utilizar el API Win16, la versión de 32 bits permite incorporar las características de Windows de 32 bits descritas con anterioridad.

La versión 4.0 de Visual C++ contiene la versión de 32 bits y la última versión de 16 bits, la 1.52c. Mientras que la versión de 16 bits permite crear aplicaciones para Windows 3.x, Windows 95 y Windows NT, la versión de 32 bits crea aplicaciones que sólo se ejecutan con Windows con soporte para 32 bits (Windows 3.x con Win32s, Windows 95 y Windows NT).

En el apéndice A se describe con más detalle lo que ha cambiado en Visual C++ hasta la fecha de impresión de este trabajo.

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

Dado el objetivo principal de esta tesis, que es el de realizar un DLL en Visual C++ para leer imágenes de un disco e ilustrar su funcionamiento, se presentan en este capítulo los conceptos básicos y fundamentales de las diferentes formas de DLLs que existen en Windows así como los conceptos fundamentales de los mapas de bits.

ARQUITECTURA DE WINDOWS 95

A continuación se listan los tipos de componentes que pueden desarrollarse para Windows 95 y las características que pueden aprovechar de Windows 95. El ambiente operativo para Windows 95 consiste de dispositivos de hardware de la computadora y los siguientes componentes de software:

1. Administrador de Máquinas virtuales (VMM)
2. Dispositivos virtuales (VxDs)
3. Controladores de Dispositivos
4. Bibliotecas de Enlace Dinámico (DLLs) de Windows de 16 y 32 bits
5. Aplicaciones basadas en MS-DOS
6. Aplicaciones basadas en Windows de 16 y 32 bits

ADMINISTRADOR DE MÁQUINAS VIRTUALES

MÁQUINAS VIRTUALES

Una máquina virtual (VM) es una tarea ejecutable que consiste de una aplicación, que soporta software (como MS-DOS y ROM BIOS), memoria, registros del procesador, espacio de direcciones y espacio para E/S de puertos. Las aplicaciones en una máquina virtual corren en un modo virtual 8086 (también llamado modo V86). En el modo V86, las aplicaciones se ejecutan como si estuvieran corriendo

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

en un Intel 8086 o un procesador compatible (o en modo real en un 80286, 80386 o en un 80486). El modo V86 proporciona los registros, instrucciones y 1 MB de espacio de dirección que las aplicaciones en modo real pueden usar. Esto significa que una aplicación MS-DOS en una máquina virtual correrá satisfactoriamente sin modificaciones. El modo V86 también permite la ejecución de aplicaciones en modo protegido en una máquina virtual. En Windows, las aplicaciones se cambian al modo protegido usando la interfaz de modo protegido de MS-DOS (DPML). Tales aplicaciones corren como aplicaciones DPML en modo real en un 80386. De hecho, el núcleo de Windows, KRNL386.EXE usa DPML para correr en modo protegido.

El Administrador de Máquinas Virtuales (VMM, Virtual Machine Manager) es el sistema operativo de 32 bits en modo protegido en el núcleo de Windows 95. Su responsabilidad principal es crear, ejecutar, monitorear y terminar máquinas virtuales para Windows y para cada aplicación no Windows. El VMM provee servicios que administran la memoria, procesos, interrupciones y excepciones como errores de protección general. El VMM trabaja con dispositivos virtuales y módulos de 32 bits en modo protegido, para permitir a los dispositivos virtuales interceptar las interrupciones y los errores para controlar el acceso que una aplicación tiene a los dispositivos del hardware y software instalado.

El VMM mapea ROM BIOS, MS-DOS, controladores de dispositivo y programas residentes en memoria (TSRs) al espacio de direcciones de la máquina virtual para que la aplicación tenga acceso a las funciones MS-DOS del sistema y las rutinas de ROM BIOS.

El VMM proporciona multitarea por prioridad. El VMM ejecuta múltiples aplicaciones simultáneamente. Esto se logra compartiendo el tiempo del procesador entre las máquinas virtuales en las cuales corren las aplicaciones.

DISPOSITIVOS VIRTUALES

Los Dispositivos Virtuales (VxDs, Virtual Devices) son programas de 32 bits que soportan el VMM independiente del dispositivo, administrando los dispositivos del hardware y ayudando al software. Los VxDs soportan todos los dispositivos de hardware de una computadora, incluyendo el Controlador de Interrupciones Programable (PIC), el temporizador (timer), dispositivo de Acceso Directo a Memoria (DMA), controlador de disco, puertos seriales, puertos paralelos, teclado, y adaptador de pantalla. Se requiere un VxD para cualquier dispositivo de hardware que tiene modos de operación configurables o retiene datos por un periodo de tiempo.

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

Un VxD puede también dar apoyo a funciones del API para aplicaciones corriendo en modo virtual 8086 o modo protegido. Estas funciones pueden dar a las funciones acceso directo a las características del VxD.

Los modos virtual 8086 y protegido anteriormente mencionados se refieren al modo de operación del procesador. Para la familia 8086 de Intel existen dos modos de operación: el modo real y el modo protegido. Existe un tercer modo que mediante rutinas del sistema operativo permite emular el modo real más de una vez en el sistema.

CONTROLADORES DE DISPOSITIVO

Un controlador de dispositivo (Device Driver) para Windows es un DLL que usa Windows para interactuar con un dispositivo de hardware, como una pantalla o teclado. En vez de acceder los dispositivos directamente, Windows carga los dispositivos controladores y llama funciones en los controladores para llevar a cabo acciones en el dispositivo. Cada controlador de dispositivos exporta un conjunto de funciones, Windows llama estas funciones para completar una acción, como dibujar un círculo o traducir un código del teclado. Las funciones del controlador también contienen el código del dispositivo específico necesario para realizar las acciones en el dispositivo. Cada controlador de dispositivo exporta un conjunto de funciones. Windows llama a estas funciones para completar una acción, como dibujar un círculo o traducir un código del teclado. Las funciones del controlador también contienen el código del dispositivo específico necesario para realizar las acciones del dispositivo.

Windows requiere de controladores de dispositivos para la pantalla, el teclado, y los puertos de comunicaciones. Otros controladores pueden también ser requeridos si se agregan dispositivos opcionales al sistema.

BIBLIOTECAS DE ENLACE DINÁMICO (DLLS)

El enlace dinámico provee un mecanismo para enlazar aplicaciones a bibliotecas de funciones en tiempo de ejecución. Las bibliotecas residen en sus propios archivos y no son copiadas a un archivo ejecutable de una aplicación como sucede con el enlace estático. Estas bibliotecas son enlazadas dinámicamente porque su código se une a una aplicación cuando ésta se carga en memoria y se ejecuta en vez de hacerlo cuando se genera el código para la aplicación. Cuando una aplicación usa un DLL, el sistema operativo carga el DLL en memoria, resuelve las referencias a funciones en el DLL para que puedan ser llamados por la aplicación, y descarga el DLL cuando no se necesita más.

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

Muchas aplicaciones corriendo en distintos procesos pueden usar el código en el DLL, con esto, solamente una copia del código está residente en el sistema. Un DLL también tiene la ventaja de actualizarse sin cambiar las aplicaciones que lo usan, mientras la interfaz de las funciones en el DLL no cambie. Esto se logra sustituyendo un DLL anterior con uno reciente en el cual sus funciones pudieran estar mejor implementadas siempre y cuando no cambien los nombres de las funciones ni su número de parámetros.

El capítulo 3 trata con más detalle todo lo referente a DLLs.

APLICACIONES BASADAS EN MS-DOS

Windows 95 soporta aplicaciones escritas para MS-DOS. Cada aplicación basada en MS-DOS puede correr en pantalla completa o como ventana sobre el escritorio de Windows 95.

El sistema puede ejecutar varias aplicaciones basadas en MS-DOS al mismo tiempo. Para poder hacerlo, el sistema crea una máquina virtual separada para cada aplicación basada en MS-DOS y comparte el microprocesador entre las VMs de MS-DOS y la VM del sistema (la cual contiene todas las aplicaciones basadas en Windows). Una VM puede ejecutar una aplicación basada en MS-DOS en modo virtual 8086 o en modo protegido del microprocesador.

Muchas de las aplicaciones MS-DOS se ejecutan bien en una ventana o en pantalla completa, hay algunas que no. Para asegurar una absoluta compatibilidad hacia atrás de las aplicaciones MS-DOS, Windows 95 proporciona un modo de operación separado llamado "modo de aplicación MS-DOS individual". Cuando se ejecuta este modo, Windows 95 corre sólo una aplicación MS-DOS.

Existen programas como herramientas para disco y otras aplicaciones que modifican directamente estructuras del sistema de archivos, como las entradas de directorio. Tales aplicaciones deben solicitar el uso exclusivo del volumen antes de hacer modificaciones a las estructuras. Windows 95 proporciona un conjunto de funciones de control para entrada y salida (IOCTL) para hacer uso exclusivo del volumen. El uso exclusivo impide que las aplicaciones cambien inadvertidamente el sistema de archivos cuando una utilería de disco está tratando de modificarlo.

APLICACIONES BASADAS EN WINDOWS

Windows 95 soporta aplicaciones de 16 bits escritas para Windows 3.x así como aplicaciones de 32 bits que usan el API de Win32 o Win32s. Para aplicaciones de 16 bits, Windows 95 conserva el modelo cooperativo de multitarea usado en Windows 3.x; esto es, todas las aplicaciones de 16 bits

comparten el mismo espacio de dirección virtual, la misma cola de mensajes, y el mismo hilo de ejecución. En contraste, cada aplicación de 32 bits basada en Windows tiene su propio espacio de dirección, una cola de mensajes privada, y uno o más hilos de ejecución.

MAPAS DE BITS

Un mapa de bits es un objeto gráfico usado para crear, manipular (escalar, desplazar, rotar, girar, y dibujar), y guardar imágenes en forma de archivo.

El mapa de bits es uno de siete objetos posibles que puede ser seleccionado dentro de un contexto del dispositivo. Existen otros seis elementos los cuales también pueden seleccionarse de la misma manera. Tales objetos son la pluma, la brocha, las fuentes, la región, la paleta lógica, y la trayectoria.

El contexto del dispositivo (device context, DC) es una estructura de datos mantenida internamente por el GDI. Un DC está asociado con un dispositivo de salida en particular, como es la impresora, un graficador o la pantalla. Para una pantalla, el DC está generalmente asociado con una ventana en particular.

En el API de Win32, las aplicaciones del panel de control de Windows son ejemplos de aplicaciones que usan mapas de bits. Por ejemplo, cuando un usuario selecciona un papel tapiz para el escritorio, el elemento seleccionado es un mapa de bits, el cual usa Windows como el fondo del escritorio.

Windows crea este patrón dibujado repetidamente un molde de 32 por 32 píxeles en el escritorio. Este figurín es guardado como un mapa de bits en un archivo dado. Supongamos que dicho nombre es Fondo.Bmp.

Desde el punto de vista de los usuarios, un mapa de bits es un rectángulo de píxeles que forman una imagen. Sin embargo, desde la perspectiva de los desarrolladores, un mapa de bits es una colección de estructuras que especifica o contiene los siguientes elementos:

- Un encabezado que describe la resolución del dispositivo en el cual el rectángulo de los píxeles fue creado, las dimensiones del rectángulo, el tamaño de la matriz de bits, e información adicional.
- Una paleta lógica.

- Una matriz de bits que define la relación entre píxeles en la imagen. Esta muestra una matriz de paleta, un rectángulo de 32 por 32 píxeles, y la matriz de índices que mapea los colores de la paleta a píxeles en el rectángulo.

TIPOS DE MAPAS DE BITS

Hay dos tipos de mapas de bits: dependiente del dispositivo (DDBs) e independiente del dispositivo (DIBs). Los DDBs fueron comunes en versiones anteriores de Windows. En realidad, ellos son los únicos mapas de bits disponibles para los desarrolladores. Sin embargo, como una mejora de la tecnología de pantallas y como la variedad de pantallas se incrementó, ciertos problemas inherentes surgieron, los cuales pudieron resolverse usando DIBs. Por ejemplo, no había método de almacenar (o de extraer) la resolución del tipo de pantalla en el cual un mapa de bits fue creado, así que una aplicación de diseño no pudo determinar rápidamente si un mapa de bits era conveniente para el tipo de despliegue de video en el cual la aplicación era corrida.

MAPAS DE BITS DEPENDIENTES DEL DISPOSITIVO (DDBS)

Nota: Los DDBs son soportados solamente por compatibilidad con las aplicaciones escritas para versiones de Windows de 16 bits anteriores a la 3.0. Las aplicaciones desarrolladas para versiones posteriores de Windows, o las que se porten de una versión anterior de Windows a la plataforma de 32 bits, deben usar DIBs.

Los DDBs son descritos usando una sola estructura, la estructura BITMAP. Los miembros de esta estructura especifican el ancho y alto de la región rectangular, en píxeles, el ancho de la matriz que mapea entradas de la paleta del dispositivo a píxeles; y el formato del color del dispositivo, en términos de los planos de color y bits por pixel. Una aplicación puede regresar el color del formato de un dispositivo llamando la función `GetDeviceCaps` y especificando las constantes apropiadas.

Una aplicación puede crear un DDB de un DIB inicializando las estructuras requeridas y llamando la función `CreateDIBitmap`. Al especificar la bandera `CBM_INIT` en la llamada a `CreateBitmap` es equivalente a llamar la función `CreateCompatibleBitmap` para crear un DDB en el formato del dispositivo y luego llamar la función `SetDIBits` para traducir los bits del DIB a los del DDB. Para determinar si un dispositivo soporta la función `SetDIBits`, se llama la función `GetDeviceCaps`, especificando la bandera `RC_DI_BITMAP` como `RASTERCAPS`.

MAPAS DE BITS INDEPENDIENTES DEL DISPOSITIVO (DIBS)

Los mapas de bits que contienen una tabla de colores son independientes del dispositivo. Una tabla de colores describe como los valores de un pixel corresponden a los valores de color RGB. RGB es un modelo para describir colores que se producen al emitir luz. Un DIB contiene la siguiente información de color y dimensión:

- El formato de color del dispositivo, en el cual la imagen rectangular fue creada.
- La resolución del dispositivo, en el cual la imagen rectangular fue creada.
- La paleta para el dispositivo, en el cual la imagen fue creada.
- Un arreglo de bits que mapea un conjunto de rojos, verdes y azules a pixeles en una imagen rectangular.
- Un identificador de compresión de datos, el cual indica el esquema de compresión de datos (si hay alguno) usado para reducir el tamaño del arreglo de bits.

La información de dimensión y color se guarda en una estructura BITMAPINFO.

La estructura BITMAPINFO contiene una estructura con información del encabezado del mapa de bits (BITMAPINFOHEADER) seguida de dos o más estructuras RGBQUAD. La estructura BITMAPINFOHEADER especifica las dimensiones del rectángulo de pixeles, describe el color de la tecnología del dispositivo, e identifica los esquemas de compresión usados para la reducción del tamaño del mapa de bits. La estructura RGBQUAD identifica los colores que aparecen el rectángulo de pixeles.

Hay dos variantes de DIBs:

- DIBs "bottom-up": El origen está la esquina inferior izquierda,
- DIBs "top-down": El origen está en la esquina superior izquierda.

Si la altura de un DIB, como se indica por el miembro de la estructura de información del encabezado del mapa de bits biHeight, es un valor positivo, es es un bottom-up DIB; si la altura es un valor negativo, entonces es un DIB de arriba hacia abajo. DIBs de arriba hacia abajo no pueden ser comprimidos.

El formato del color se especifica en términos de un total de planos de colores y bits de color. El total de planos de colores es siempre 1: el número de bits de colores es 1 para los mapas de bits monocromáticos, mapas de bits de 4 para VGA, y 8, 16, o 32 para mapas de bits en otros dispositivos de

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

color. Una aplicación regresa un número de bits de color en un dispositivo particular usado para llamar la función `GetDeviceCaps`.

La resolución de un dispositivo de despliegue es especificado en pixeles por metro. Una aplicación puede reducir la resolución horizontal para un despliegue de video, o impresora, los siguiente son tres procesos:

- Llama la función `GETDEVICECAPS`, especificando `HORZRES` como el segundo argumento.
- Llama `GETDEVICECAPS` un segundo tiempo, especificando `HORZSIZE` como el segundo argumento.
- Divide el primer valor de regreso por el segundo valor de regreso.

La aplicación puede regresar la resolución vertical usando el mismo proceso de tres pasos con diferentes parámetros: `VERTRES` en lugar de `HORZRES`, y `VERTSIZE` en lugar de `HORZSIZE`.

La paleta está representada por un arreglo de estructura `RGBQUAD` que especifica los componentes de intensidad del rojo, verde y azul para cada color en un despliegue de dispositivos de la paleta de color. Cada índice de color en el arreglo de la paleta mapea a un pixel específico en la región rectangular asociada con el mapa de bits. El tamaño de este arreglo, en bits, es equivalente a la anchura de el rectángulo, en pixeles, multiplicado por la altura de el rectángulo, en pixeles, multiplicado por el número de color de bits para el dispositivo. Una aplicación puede regresar el tamaño de la paleta de los dispositivos para llamar a la función `GETDEVICECAPS`, especificando el constante `NUMCOLORS` como el segundo argumento.

El API de Win32 soporta la compresión de la matriz de la paleta para los bottom-up DIBs de 8 bits por pixel y 4 bits por pixel. Estas matrices pueden ser comprimidas usando el esquema RLE (codificación de longitud de ejecución). El esquema RLE usa valores de 2 bytes, el primer byte especifica el número de pixeles consecutivos que usan un índice de color y el segundo byte especifica el índice.

Una aplicación puede crear un DIB de un DDB mediante la inicialización de las estructuras requeridas y llamando la función `GetDIBits`. Para determinar si un dispositivo soporta esta función, se llama la función `GetDeviceCaps`, especificando la bandera `RC_DI_BITMAP` como `RASTERCAPS`.

Una aplicación puede usar un DIB para establecer pixeles en el dispositivo de despliegue llamando la función `SetDIBitsToDevice` o la función `StretchDIBits`. Para determinar si un dispositivo soporta la función `SetDIBitsToDevice`, se llama la función `GetDeviceCaps`, especificando la bandera

CAPÍTULO 2. Dispositivos virtuales y Mapas de Bits

RC_DIBTODEV como RASTERCAPS. Especifique EC_STRETCHDIB como la bandera RASTERCAPS para determinar si el dispositivo soporta StretchDIBits.

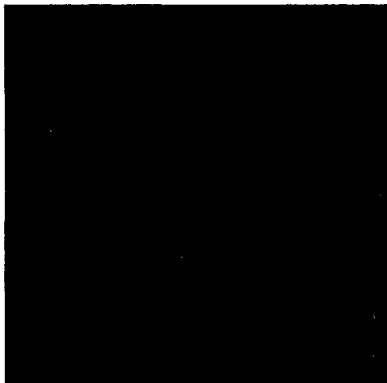
Una aplicación que simplemente necesita desplegar un DIB existente puede usar la función SetDIBitsToDevice.

Ejemplo de un mapa de bits:

Palette-index

```
row 0, scanline 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
row 1, scanline 30 00 00 00 00 00 00 00 00 09 00 00 00 00 00 00 00
row 2, scanline 29 11 11 01 19 11 01 10 10 09 09 01 09 11 11 01 90
.
.
.
row 31, scanline 0 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 90
```

Pixel rectangle



Color-palette

0 Black
1 Dark red
2 Dark green
3 Dark yellow
4 Dark blue
5 Dark magenta
6 Dark cyan
7 Dark gray
8 Light gray
9 Light red
A Light green
B Light yellow
C Light blue
D Light magenta
E Light cyan
F White

En el ejemplo anterior, el rectángulo de píxeles fue creado en un monitor VGA usando una paleta de 16 colores. Una paleta de 16 colores requiere índices de 4 bits; por consiguiente, la matriz que mapea los colores de la paleta de colores de los píxeles es compuesta de índices de 4 bits. Note que en el mapa de bits de arriba; Windows mapea índices a píxeles empezando con la línea de búsqueda de abajo en la región rectangular y finalizando con la línea de búsqueda de arriba. (Una línea de búsqueda es una fila de píxeles adyacentes en un despliegue.) Por ejemplo, la primera fila de la matriz (fila 0) corresponde a los píxeles de la fila del fondo, la línea de búsqueda 31. Esto es porque el mapa de bits de arriba es un mapa de bits independiente del dispositivo (DIB) de abajo hacia arriba. En los DIBs de arriba a abajo y en los mapas de bits dependientes del dispositivo (DDBs), Windows mapea índices a píxeles empezando con la línea de búsqueda de arriba.

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

Dado el objetivo principal de esta tesis, que es el de realizar un DLL en Visual C++ para leer imágenes de un disco e ilustrar su funcionamiento, se presentan en este capítulo los conceptos básicos y fundamentales de los DLLs.

Las Bibliotecas de Enlace Dinámico (DLLs) han sido el pilar de Windows desde su primera versión. Todas las funciones del API de Windows están contenidas en DLLs. Existen tres DLLs en el sistema operativo Windows que son los más importantes.

- **KERNEL32.DLL:** Contiene funciones para el manejo de memoria
- **USER32.DLL:** Contiene funciones para llevar a cabo tareas de la interfaz del usuario como la creación de ventanas y envío de mensajes
- **GDI32.DLL:** Agrupa funciones para el dibujo imágenes gráficas y despliegue de texto

Además de estos, Windows contiene muchos otros DLLs que contienen funciones para ejecutar tareas más especializadas. Algunos de ellos son:

- **OLE32.DLL:** Biblioteca para el enlace e incrustación de objetos
- **COMDLG32.DLL:** Contiene los diálogos comunes (como FileOpen y FileSave)
- **LZ32.DLL:** Agrupando funciones para la descompresión de archivos
- **ADVAPI32.DLL :** Contiene funciones para seguridad de objetos, manipulación del registro y seguimiento de eventos

Un programa en Windows es un archivo ejecutable que generalmente crea una o más ventanas y usa un ciclo de mensajes para recibir la entrada del usuario. Las Bibliotecas de Enlace Dinámico no son ejecutables directamente, y generalmente no reciben mensajes.

Un sistema operativo convencional, contiene rutinas a otros programas que puede llamar para hacer una tarea específica. En el caso de Windows, al escribir un DLL, se está escribiendo una extensión a Windows. Desde otro punto de vista, los DLLs (incluyendo los que forman Windows) son extensiones de un programa.

DEFINICIÓN

Un DLL es un archivo ejecutable que actúa como una biblioteca compartida de funciones y recursos para una aplicación. El enlace dinámico proporciona una forma a un proceso o a un DLL de llamar a una función o a un recurso que no es parte de su código ejecutable. Un DLL es llamado por una aplicación en tiempo de ejecución. Las funciones y los recursos dentro de una biblioteca de enlace dinámico son unidos hasta que la aplicación se ejecuta, esto quiere decir que el código del DLL no es agregado al archivo ejecutable cuando se construye la aplicación. Al momento de ejecutar el código del DLL, éste se une al código del archivo ejecutable pero en la memoria. En un DLL no existe código para ciclos de mensajes o creación de ventanas internas.

Un programa es un archivo EXE que se puede ejecutar (o correr) desde Windows. Una vez que un programa se está ejecutando, es un proceso. Un proceso posee su propia memoria, apuntadores a archivo y otros recursos del sistema.

Un DLL consiste de datos globales, funciones compiladas, y recursos, que forman parte de un proceso. Un DLL se compila para cargarse a una dirección base, y si no hay conflicto con otros DLLs, el archivo se mapea a la misma dirección virtual del proceso que lo llamó. El DLL tiene varias funciones exportadas, y el programa cliente (el programa que cargó primero el DLL) importa esas funciones. Windows iguala las importaciones a las exportaciones cuando carga el DLL.

Un DLL contiene una tabla de funciones exportadas. Estas funciones son identificadas por el mundo exterior por sus nombres simbólicos y (opcionalmente) por enteros llamadas números ordinales. La tabla de funciones también contiene las direcciones de las funciones dentro del DLL. Cuando un programa cliente carga primero el DLL, no conoce las direcciones de las funciones que necesita llamar, pero conoce los símbolos u ordinales. Luego, el proceso de enlace dinámico construye una tabla que conecta las llamadas del programa cliente a las direcciones de las funciones en el DLL.

TIPOS DE BIBLIOTECAS

Parte de la confusión que envuelve a las Bibliotecas de Enlace Dinámico resulta de la presencia de la palabra biblioteca en diferentes contextos. Además de las Bibliotecas de Enlace Dinámico, también existen las Bibliotecas de Objetos y las Bibliotecas de Importaciones.

BIBLIOTECA DE OBJETOS

Es un archivo con extensión LIB que contiene código que se agrega al archivo EXE de un programa cuando se ejecuta el enlazador durante el enlace estático.

Ejemplo: En Visual C++, la biblioteca de objetos en tiempo de ejecución de C (C runtime object library) que se enlaza con un programa se llama LIBC.LIB.

BIBLIOTECA DE IMPORTACIONES

Es una forma especial de archivo de biblioteca de objetos. Como las bibliotecas de objetos, las bibliotecas de importación tienen la extensión LIB y son usadas por el enlazador para resolver llamados de funciones en el código fuente. Sin embargo, las bibliotecas de importación no contienen código. En vez de eso, proporcionan al enlazador información necesaria para ajustar tablas de relocación dentro del archivo EXE para el enlace dinámico.

Ejemplo: Los archivos KERNEL32.LIB, USER32.LIB y GDI32.LIB incluidos con el compilador Microsoft son bibliotecas de importación para las funciones de Windows. Si se llama a la función Rectangle en un programa, GDI32.LIB le dice a LINK que esta función está en la Biblioteca de Enlace Dinámico GDI32.DLL. Esta información va en el archivo EXE para que Windows pueda llevar a cabo el enlace dinámico con la biblioteca GDI32.DLL cuando el programa se ejecute.

USOS DE LAS BIBLIOTECAS

Las bibliotecas de importación y las bibliotecas de objetos son usadas solamente durante el desarrollo del programa.

Las bibliotecas de enlace dinámico se usan durante el tiempo de ejecución. Un DLL debe estar presente en el disco cuando un programa que usa la biblioteca se ejecuta.

TIPOS DE DLLS

DLLS CON CÓDIGO

Los DLLs no son ejecutables directamente, y generalmente no reciben mensajes. Son archivos separados que contienen funciones que pueden ser llamadas por programas y otros DLLs para ejecutar ciertas tareas. Un DLL entra en acción solamente cuando un proceso o un DLL llama una de las funciones en la biblioteca.

DLLS SÓLO DE RECURSOS

Algunos DLLs (como archivos de fuentes) son llamados "DLLs sólo de recursos". Contienen solamente datos (generalmente en la forma de recursos) pero no contienen código.

USOS DE LOS DLLS

Los DLLs son utilizados para tres propósitos generales:

- Compartir componentes con múltiples aplicaciones, tanto funciones como recursos. El contenido de un DLL se aísla de la aplicación principal en el DLL, el cual puede ser utilizado por diferentes aplicaciones.
- Encapsulamiento de código y datos. Con esta característica, se puede escribir software modular. Al usar clases C++ en DLLs, éstos son modulares al momento de la ejecución. De lo contrario, las clases contenidas en archivos ejecutables, son modulares al momento de la construcción en los archivos ejecutables.
- Realización de operaciones en los niveles del sistema.

TIPOS DE ENLACE

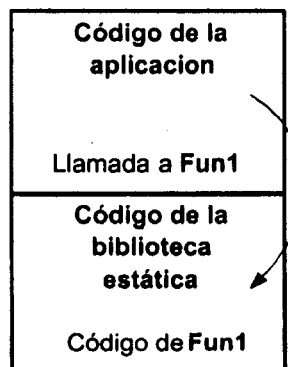
La palabra *enlace* se refiere a la forma en que un archivo ejecutable crea un vínculo con un DLL cuando necesita parte del contenido del DLL.

Esta sección define y contrasta los principios generales de dos tipos de enlace, el enlace estático y el dinámico.

ENLACE ESTÁTICO

En los lenguajes de programación de alto nivel como C, Pascal o FORTRAN, el código fuente de una aplicación se compila y se enlaza a varias bibliotecas para crear un archivo ejecutable. Estas bibliotecas contienen archivos objeto de funciones previamente compiladas que son llamadas para efectuar tareas comunes, como calcular raíces cuadradas. Cuando estas funciones de biblioteca son enlazadas a la aplicación, pasan a formar parte permanente del archivo ejecutable de la aplicación. Todas las llamadas de las funciones de biblioteca son resueltas en tiempo de enlace, por eso el nombre de enlace estático. La siguiente figura ilustra una aplicación llamando a una biblioteca de enlace estático.

Figura 1: Llamado de una función en una biblioteca de enlace estático



Fuente: MSDN, Julio 1995

En la figura anterior, el enlazador ha añadido el código de Fun1 de la biblioteca estática al código de la aplicación. Además del código, cualquier dato de la función de biblioteca Fun1 será agregado a los datos de la aplicación.

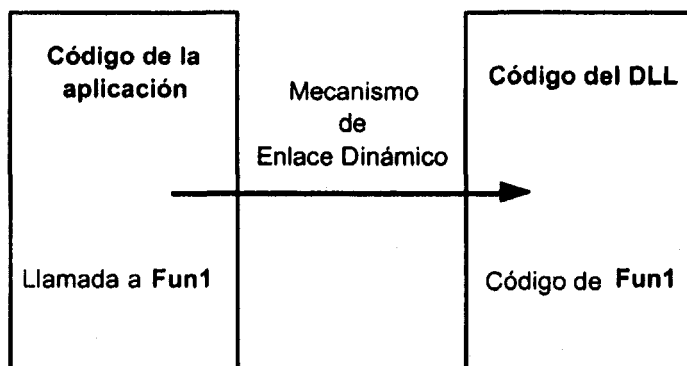
ENLACE DINÁMICO

El Enlace Dinámico proporciona un mecanismo para enlazar aplicaciones a bibliotecas en tiempo de ejecución. Las bibliotecas se encuentran en sus propios archivos ejecutables y no son copiadas al código de los archivos ejecutables de la aplicación como sucede en el enlace estático. Estas bibliotecas son llamadas bibliotecas de enlace dinámico (DLL) para enfatizar que son enlazadas a una aplicación cuando es cargada y ejecutada, en vez de cuando es ligada. Cuando una aplicación usa un DLL, el sistema operativo carga el DLL en memoria, resuelve las referencias a funciones en el DLL para que sean llamadas por la aplicación, y descarga el DLL cuando no se necesita más. El mecanismo de enlace

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

dinámico puede ser efectuado explícitamente por aplicaciones o implícitamente por el sistema operativo. La siguiente figura ilustra una aplicación llamando a una función en una biblioteca de enlace dinámico.

Figura 2: Llamado de una función en una biblioteca de enlace dinámico

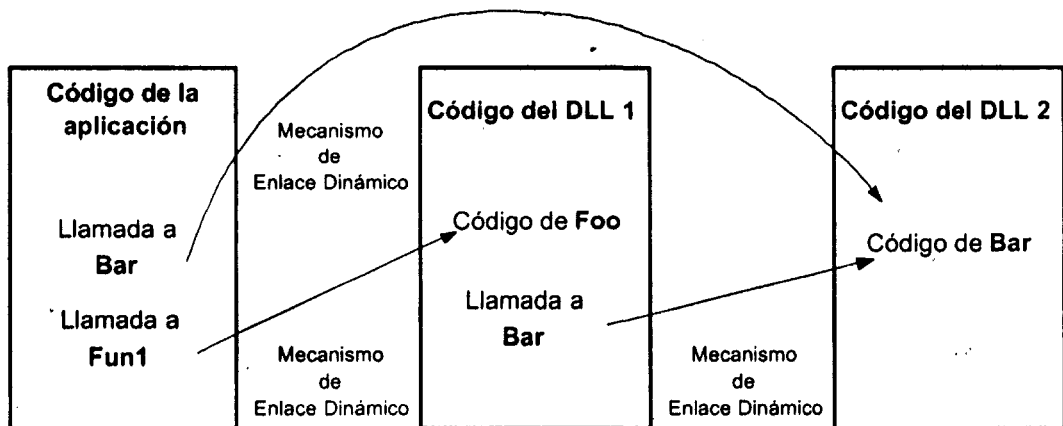


Fuente: MSDN, Julio 1995

En este ejemplo, el código para la función Fun1 no está contenido dentro de la aplicación; en vez de eso, permanece en el archivo ejecutable DLL y es cargado en memoria en tiempo de ejecución. La llamada de la aplicación a Fun1 es diferente a una de sus propias funciones porque la dirección destino de Fun1 no se conoce hasta el tiempo de ejecución, mientras que la dirección de las propias funciones de la aplicación (y de cualquier función de biblioteca estática que use) se conocidas durante el tiempo de enlace. Cualquier dato estático usado por Fun1 reside en el DLL en vez de la aplicación.

Cuando una aplicación depende de un DLL, decimos que la aplicación usa el DLL. Los DLLs pueden ser usados por otros DLLs así como por aplicaciones. Note que esta relación es de un solo lado, aunque las aplicaciones pueden usar DLLs, los DLLs no pueden usar aplicaciones. Esto es, aunque un DLL puede ser implícitamente cargado como una consecuencia de cargar una aplicación o DLL, una aplicación nunca será implícitamente cargada como una consecuencia de cargar un DLL. La siguiente figura ilustra esta relación.

Figura 3: Los DLLs enlazando dinámicamente otros DLLs



Fuente: MSDN, Julio 1995

En la figura anterior, el DLL 1 es usado por la aplicación, pero el DLL 2 es usado por el DLL 1 y por la aplicación.

TIPOS DE ENLACE DINÁMICO

Existen dos tipos de enlace dinámico:

- Enlace dinámico implícito
- Enlace dinámico explícito

Enlace dinámico implícito

Al enlace implícito también se le conoce como enlace dinámico en tiempo de carga o enlace dinámico en tiempo de carga estático.

Cuando el código fuente del archivo ejecutable que contiene llamados a funciones se compila, los llamados a funciones en un DLL generan una referencia de funciones externa en el código objeto. Para resolver esta referencia externa, la aplicación debe enlazarse con un archivo de importación (LIB) proporcionado por quien construyó el DLL.

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

Cuando se construye un DLL, el enlazador produce un archivo compañero de importación con extensión LIB (biblioteca de importación), el cual contiene todos los símbolos de exportación y (opcionalmente) números ordinales, pero no contiene código. El archivo LIB es un sustituto para el DLL que se agrega al proyecto del programa cliente. Para resolver referencias externas a DLLs, el enlazador simplemente agrega la información al programa ejecutable y le dice al sistema donde encontrar el código del DLL cuando inicie el proceso.

Cuando se construye el programa cliente utilizando enlace estático, los símbolos importados (del programa cliente) se igualan a los símbolos exportados en el archivo LIB, y esos símbolos (o enteros ordinales) se unen en el archivo EXE.

El archivo LIB contiene el nombre del DLL (pero no su nombre completo), el cual se guarda dentro del archivo EXE. Cuando el cliente se carga, el programa que usa el DLL se enlaza al archivo LIB proporcionado por quien construyó el DLL.

Después, cuando el sistema (Windows) inicia un programa que contiene referencias enlazadas dinámicamente, usa la información en el archivo ejecutable para localizar los DLLs requeridos. Luego, los enlaza dinámicamente por símbolo o por ordinal. El programa cliente llama a las funciones exportadas del DLL como si las funciones estuvieran contenidas en el archivo ejecutable.

Si el sistema encuentra el DLL, mapea los módulos en el espacio de direcciones del proceso. Si cualquiera de los DLLs tiene una función de punto de entrada (código para inicializar y terminar), el sistema operativo llama a la función. Uno de los parámetros que se pasa a la función de punto de entrada especifica un código que indica que el DLL se está agregando al proceso. Si la función no regresa el valor de TRUE, el sistema termina el proceso y reporta un error.

Para enlazar implícitamente a un DLL, los archivos ejecutables deben obtener lo siguiente de quien construyó el DLL:

- Un archivo de encabezados (archivo con extensión H) que contenga las declaraciones de las funciones exportadas y/o clases C++.
- Una biblioteca de importación (archivo LIB) con la cual enlazarse (el enlazador crea la biblioteca de importación cuando se construye el DLL).
- El archivo DLL

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

Los archivos ejecutables que usan el DLL deben incluir el archivo de encabezados que contenga las funciones exportadas (o clases C++) en cada archivo fuente que contenga llamadas a las funciones exportadas.

Para construir el archivo ejecutable que llame a las funciones, se debe enlazar con la biblioteca de importación. Si se utiliza el ambiente de desarrollo Visual C++, se debe especificar el nombre de la biblioteca de importación en la caja de diálogo en Object/Library Modules de la ficha Link en el menú Project/Settings.

El archivo DLL debe estar en ciertos límites de búsqueda. Buscar en "Trayecto de búsqueda usado por Windows para localizar un DLL" más adelante.

Si el sistema no puede encontrar el DLL, termina el proceso y despliega un mensaje de error.

Enlace dinámico explícito

Al enlace dinámico explícito también se le conoce como enlace dinámico en tiempo de carga dinámica o enlace dinámico en tiempo de ejecución.

Con el enlace explícito, las aplicaciones deben hacer una llamada a una función para cargar explícitamente el DLL en tiempo de ejecución. Para enlazar explícitamente a un DLL, una aplicación debe:

- Llamar a LoadLibrary (o una función similar) para cargar el DLL y obtener un apuntador al archivo.
- Llamar a GetProcAddress para obtener un apuntador de función para cada función que la aplicación quiera llamar. Debido a que las aplicaciones llaman a las funciones en el DLL a través de un apuntador, el compilador no genera referencias externas, por lo que no hay necesidad de enlazar al DLL con una biblioteca de importación.
- Llamar a FreeLibrary cuando se termine de usar el DLL.

El siguiente fragmento de código ilustra el proceso:

```
// Definición de tipo para apuntador a función
// correspondiente al prototipo de la función exportada
typedef UINT (CALLBACK* LPFNDDLLFUNC1) (DWORD,UINT);
.
.
.
HINSTANCE hDLL;           // Apuntador al DLL
LPFNDDLLFUNC1 lpfnDllFunc1; // Apuntador a una función
DWORD dwParam1;
```

```

UINT  uParam2, uReturnVal;

hDLL = LoadLibrary("MyDLL");
if (hDLL != NULL)
{
    lpfnDllFunc1 = (LPFNDDLFUNC1)GetProcAddress(hDLL, "DLLFunc1");
    if (!lpfnDllFunc1)
    {
        // handle the error
        FreeLibrary(hDLL);
        return SOME_ERROR_CODE;
    }
    else
    {
        // call the function
        uReturnVal = lpfnDllFunc1(dwParam1, uParam2);
    }
}

```

LOADLIBRARY Y AFXLOADLIBRARY

Un proceso debe llamar a la función `LoadLibrary` (o `AfxLoadLibrary`) para enlazarse explícitamente a un DLL. Si todo sale bien, la función mapea el DLL especificado en el espacio de direcciones del proceso y regresa un apuntador al DLL que se puede usar con otras funciones en el enlace explícito como es el caso de `GetProcAddress` y `FreeLibrary`.

`LoadLibrary` intenta lo encontrar el DLL usando el "Trayecto de búsqueda usado por Windows para localizar un DLL" descrito más adelante.

Si el sistema no puede encontrar el DLL o si la función de punto de entrada regresa un valor `FALSE`, `LoadLibrary` regresa `NULL`.

Si la llamada a `LoadLibrary` especifica un DLL que ya se mapeo en el espacio de direcciones del proceso que lo llamó, la función simplemente regresa un apuntador al DLL e incrementa el contador de referencia del módulo.

Si por alguna razón el sistema no puede cargar el DLL, el proceso intenta recuperarse del error.

GETPROCADDRESS

Los procesos que se enlazan explícitamente a un DLL deben llamar `GetProcAddress` para obtener la dirección de una función exportada en el DLL. Se usa el apuntador a la función de regreso para llamar a la función en el DLL. `GetProcAddress` requiere de parámetros como el apuntador al DLL (regresado por `LoadLibrary`) y, o el nombre de la función que se quiere llamar o el ordinal de exportación de la función.

Debido a que la función en el DLL se está llamando a través de un apuntador, no hay verificación de tipos durante la compilación, por lo tanto, los parámetros deben ser correctos para no saturar la memoria asignada a la pila y causar una "violación de acceso". Una manera de asegurarse de los tipos correctos, es usar definición de tipos para las funciones exportadas. Ver el fragmento de código perteneciente a la sección "Enlace dinámico implícito" descrito anteriormente.

FREELIBRARY Y AFXFREELIBRARY

Los procesos que se enlazan explícitamente a un DLL deben llamar la función `FreeLibrary` cuando el DLL no se necesite más. Esta función decrementa el contador de referencia del módulo y si el contador es cero, lo borra del espacio de direcciones del proceso.

Las aplicaciones MFC deben usar `AfxFreeLibrary` en vez de `FreeLibrary` para descargar un DLL extensión. La interfaz (prototipo de la función) para `AfxFreeLibrary` es la misma de `FreeLibrary`.

DETERMINAR QUE MÉTODO DE ENLACE USAR

La mayoría de las aplicaciones usan el enlace implícito porque es el método más fácil de usar. Sin embargo, hay veces que el enlace explícito es necesario. La siguiente lista presenta razones comunes para usar el enlace explícito:

- La aplicación no conoce el nombre del DLL que tendrá que cargar hasta cuando se ejecute el programa. Por ejemplo, la aplicación podría necesitar obtener el nombre del DLL y de las funciones exportadas de un archivo de configuración.
- Un proceso que use enlace implícito es terminado por el sistema operativo si el DLL no se encuentra cuando el proceso inicia. Un proceso que use enlace explícito no es terminado en esta situación y puede intentar recuperarse del error. Por ejemplo, el proceso podría notificar al usuario del error y esperar que el usuario especifique otro trayecto para el DLL.
- Un proceso que usa enlace implícito también es terminado si cualquiera de los DLLs a los que se enlaza tiene una función `DllMain` que falla. Un proceso que usa enlace explícito es terminado en esta situación.
- Una aplicación que se enlaza implícitamente a muchos DLLs puede ser lenta al iniciar porque Windows carga todos los DLLs cuando la aplicación se carga. Para mejorar el desempeño al iniciar, una aplicación se puede enlazar implícitamente a aquellos DLLs que se necesiten inmediatamente después de cargar y esperar a enlazarse explícitamente a los otros DLLs cuando se vayan a ocupar.

- El enlace explícito elimina la necesidad de enlazar la aplicación con una biblioteca de importación. Si los cambios en el DLL hacen que los ordinales de exportación cambien, las aplicaciones que usan enlace explícito no tienen que reenlazarse (asumiendo que llaman a `GetProcAddress` con un nombre de una función y no con un valor ordinal), mientras que las aplicaciones que usan enlace implícito deben reenlazarse a la nueva biblioteca de importación.

OPERACIONES DE LOS DLLS

EXPORTAR DE UN DLL

Un DLL tiene una distribución interna muy similar a un archivo EXE, con una diferencia importante —un DLL tiene una tabla de exportación. La tabla de exportación contiene el nombre de cada función que exporta el DLL a otros ejecutables. Estas funciones son los puntos de entrada en el DLL; sólo las funciones en la tabla de exportación pueden accesarse por otros ejecutables. Cualquier otra función en el DLL es privada al DLL. La tabla de exportación puede verse usando un programa llamado `DUMPBIN` (incluido con Visual C++) con la opción `/EXPORTS`.

Se pueden exportar funciones de un DLL usando dos métodos:

- Crear un archivo de definición de módulo (.DEF) y usar el archivo DEF cuando se construye el DLL.
- Usando la palabra reservada `__declspec(dllexport)` en la definición de la función.

Cuando se exportan funciones usando cualquier método, es necesario usar la convención de llamada `_stdcall`.

EXPORTAR DE UN DLL USANDO ARCHIVOS .DEF

Un archivo de definición de módulo (.DEF) es un archivo de texto que contiene uno o más enunciados que describen varios atributos de un DLL. Si no se usa la palabra reservada `__declspec(dllexport)` para exportar las funciones del DLL, entonces se requiere el archivo .DEF.

El archivo .DEF más simple debe contener los siguientes enunciados de definición de módulo:

- El primer enunciado en el archivo debe ser el enunciado `LIBRARY`. Este enunciado identifica el archivo .DEF como perteneciente a un DLL. El enunciado `LIBRARY` es precedido por el nombre del DLL. El enlazador coloca este nombre en la biblioteca de importación del DLL.

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

- El enunciado EXPORTS lista los nombres y, opcionalmente, los valores ordinales de las funciones exportadas por el DLL. Se asigna a la función un valor ordinal seguido por el nombre de la función con un signo "@" y un número. Cuando se especifican valores ordinales, deben estar dentro del rango 1 a N, donde N es el número de funciones exportadas por el DLL.
- Aunque no se requiere, un archivo .DEF puede contener el enunciado DESCRIPTION que describe el propósito del DLL.

Por ejemplo, un DLL que contiene el código para implementar un árbol de búsqueda binaria podría verse como lo siguiente:

```
LIBRARY      BTREE
DESCRIPTION  "Implementa un árbol binario."
EXPORTS
    Insert    @1
    Delete    @2
    Member @3
    Min       @4
```

Si se usa AppWizzard para crear un DLL MFC, AppWizzard crea un archivo .DEF esqueleto y automáticamente lo agrega al proyecto. Luego, se deben agregar los nombres de las funciones a exportarse al archivo .DEF. Para los DLLs sin MFC, se debe crear un archivo .DEF y agregarlo al proyecto manualmente.

Si se exportan funciones en un archivo C++, se tendrán que colocar los nombres decorados en el archivo .DEF o definir las funciones exportadas con enlace común estilo C usando la expresión extern "C". Si se necesitan colocar los nombres decorados en el archivo .DEF, los nombres se obtienen usando la herramienta DUMPBIN o usar la opción /MAP del enlazador.

Nota: Los nombres decorados producidos por el compilador son específicos al compilador. Si se colocan los nombres decorados producidos por el compilador de Visual C++ en un archivo .DEF, las aplicaciones que se enlacen al DLL deben también ser construidas usando la misma versión de Visual C++ para que los nombres decorados en la aplicación cliente sean iguales a los nombres exportados en el archivo .DEF del DLL.

Si se construye un DLL extensión, y se exporta el contenido en un archivo .DEF, se debe colocar el siguiente código al principio y al final de los archivos de encabezados que contienen las clases exportadas:

```
#undef AFX_DATA
#define AFX_DATA AFX_EXT_DATA
// <cuerpo del archivo de encabezados>
#undef AFX_DATA
```

```
#define AFX_DATA
```

Estas líneas aseguran que las variables MFC usadas internamente o que se agregan a las clases se exporten (o importen) del DLL extensión. Por ejemplo, cuando se deriva una clase usando DECLARE_DYNAMIC, esta macro se expande para agregar una variable miembro CruntimeClass a la clase. Si se omiten estas cuatro líneas puede causar que el DLL se compile o se enlace incorrectamente o puede causar un error cuando la aplicación cliente se enlace al DLL.

Cuando se construye el DLL, el enlazador usa el archivo .DEF para crear un archivo de exportación (.EXP) y una biblioteca de (.LIB). Luego, el enlazador usa el archivo de exportación para construir el DLL. Los ejecutables que se enlazan implícitamente al DLL se enlazan a la biblioteca de importación cuando se construyen.

Nota: MFC usa archivos .DEF para exportar funciones y clases de MFCx0.DLL.

EXPORTAR DE UN DLL USANDO __DECLSPEC(DLLEXPORT)

Se pueden exportar datos, funciones, clases o funciones miembro de una clase desde un DLL usando la palabra reservada __declspec(dllexport). Si se usa __declspec(dllexport), no se necesita un archivo .DEF para exportar.

Para exportar funciones, la palabra reservada __declspec(dllexport) debe aparecer a la izquierda de la palabra reservada con la cual se hace la llamada, si una palabra reservada se especifica. Por ejemplo:

```
void __declspec(dllexport) __cdecl Función1(void);
```

Para exportar todos los miembros públicos y funciones miembro en una clase, la palabra reservada debe aparecer a la izquierda del nombre de la clase como sigue:

```
class __declspec(dllexport) CexampleExport : public Cobject
{ .. definición de la clase ... };
```

Cuando se construye un DLL, generalmente se crea un archivo de encabezados que contiene los prototipos de las funciones y/o clases que se exportan, y se agrega __declspec(dllexport) a las declaraciones en el archivo de encabezados. Para hacer el código más legible, es recomendable definir una macro para __declspec(dllexport) y luego usar la macro con cada símbolo que se exporta:

```
#define DllExport __declspec(dllexport)
```

Nota: Cuando se porta el código fuente de Win16 a Win32, se debe reemplazar cada instancia de __export con __declspec(dllexport).

EXPORTAR E IMPORTAR DE UN DLL USANDO AFX_EXT_CLASS

Los DLLs extensión usan la macro AFX_EXT_CLASS para exportar clases; los ejecutables que se enlazan a DLLs extensión usan la macro para importar clases. Con la macro AFX_EXT_CLASS, los mismos archivos de encabezados para construir el DLL extensión pueden usarse con los ejecutables que se enlazan al DLL.

N el archivo de encabezados del DLL, se agrega la palabra reservada AFX_EXT_CLASS a la declaración de la clase como se muestra a continuación:

```
class AFX_EXT_CLASS CMiClase : public Cdocument
{
// < cuerpo de la clase >
};
```

Esta macro es definida por MFC como __declspec(dllexport) cuando se definen los símbolos del preprocesador _AFXDLL y _AFXEXT. Pero la macro se define como __declspec(dllimport) cuando se define _AFXDLL y _AFXEXT no se define. Cuando se define, el símbolo del preprocesador _AFXDLL indica que la versión compartida de MFC está siendo usado por el ejecutable destino (un DLL o una aplicación). Cuando ambos _AFXDL y _AFXEXT se definen, esto indica que el ejecutable destino es un DLL extensión.

La siguiente tabla resume la descripción anterior cuando se define la macro AFX_EXT_CLASS:

Tabla 6 - Definición de la macro AFX_EXT_CLASS

_AFXDLL	_AFXEXT	Equivalente a	Tipo de archivo destino
X	X	__declspec(dllexport)	DLL Extensión
X		__declspec(dllimport)	DLL o aplicación

Fuente: Ayuda de Visual C++

Debido a que AFX_EXT_CLASS se define como __declspec(dllexport) cuando se exporta de un DLL extensión, se pueden exportar clases enteras sin colocar los nombres decorados para todos los símbolos de las clases en el archivo .DEF.

Aunque se puede evitar crear un archivo .DEF y todos los nombres decorados para la clase con este método, crear un archivo .DEF es más eficiente porque los nombres pueden ser exportados por ordinal. Para usar el método de exportación del archivo .DEF, se debe colocar el siguiente código al principio y al final del archivo de encabezados:

```
#undef AFX_DATA
#define AFX_DATA AFX_EXT_DATA
// < cuerpo del archivo de encabezados >
#undef AFX_DATA
#define AFX_DATA
```

Exportación de miembros individuales en una clase

Algunas veces no se necesita exportar una clase completa, en este caso, se pueden exportar sólo miembros individuales de una clase. Para exportar miembros individuales de una clase, por ejemplo, una clase derivada de `CDialog`, sólo se necesita exportar el constructor y la llamada a `DoModal`. Se puede usar `AFX_EXT_CLASS` en los miembros individuales que se requieren exportar. Por ejemplo:

```
class CDialogoEjemplo : public CDialog
{
public:
    AFX_EXT_CLASS CDialogoEjemplo();
    AFX_EXT_CLASS int DoModal();
    ..
    // resto de la definición de la clase
    ...
};
```

EXPORTAR FUNCIONES C++ PARA USARSE EN EJECUTABLES EN LENGUAJE C

Si se tienen funciones en un DLL escrito en C++ que se quieran acceder de un módulo de lenguaje C, se debe declarar estas funciones con enlace C en vez de enlace C++. A menos que se especifique lo contrario, el compilador C++ usa nombramiento de tipos seguro C++ (C++ type-safe naming o decoración de nombres) y convenciones de llamada C++, las cuales pueden ser difíciles de llamar desde C.

Para especificar enlace C, se especifica `extern "C"` para las declaraciones de las funciones. Por ejemplo:

```
extern "C" __declspec(dllexport) int MiFunción (long parámetro1);
```

EXPORTACIÓN DE FUNCIONES ESTILO C PARA USARSE EN EJECUTABLES EN LENGUAJES C O C++

Si se tienen funciones en un DLL escritas en C que se quieran acceder de un módulo en lenguaje C o C++, se debe usar la macro `__cplusplus` del preprocesador para determinar el lenguaje usado para compilar, y luego declarar estas funciones con enlace C si se está usando desde un módulo en lenguaje

C++. Si se usa esta técnica y se proporcionan archivos de encabezados para el DLL, estas funciones pueden usarse por usuarios de C y C++ sin cambios.

El siguiente código muestra el archivo de encabezados el cual puede usarse por aplicaciones cliente C y C++:

```
// MiFuncion.h
#ifdef __cplusplus
extern "C" { // sólo es necesario para exportar la interfaz
            // si es usado por el código fuente C++
#endif

__declspec( dllimport ) void MiFuncion();
__declspec( dllimport ) void OtraFuncion();

#ifdef __cplusplus
}
#endif
```

Si se necesitan enlazar funciones en C a un ejecutable C++ y la declaración de la función en los archivos de encabezados no usa la técnica anterior, en el archivo de código fuente C++, agregar lo siguiente para evitar que el compilador decore los nombres de las funciones en C:

```
extern "C"
#include "MiCArchivoEncabezados.h"
}
```

DETERMINAR QUE MÉTODO DE EXPORTACIÓN USAR

A continuación se listan las ventajas y las desventajas de exportar desde un DLL usando tanto archivos .DEF como la directiva `__declspec(dllexport)`.

Uso de los archivos .DEF

VENTAJAS

- Se pueden controlar los ordinales de exportación. Cuando se agregan funciones exportadas adicionales al DLL, se les puede asignar valores ordinales (mayores que cualquier otra función exportada). Cuando se hace lo anterior, las aplicaciones que usan enlace implícito no tienen que enlazarse de nuevo con la biblioteca de importación nueva que contiene las funciones nuevas. Esto es muy importante, por ejemplo, si se diseña un DLL de terceros para usarse con muchas aplicaciones. Se puede continuar mejorando el DLL al agregar funcionalidad adicional mientras que se asegura que las aplicaciones que lo usan continuarán trabajando adecuadamente con el DLL. Los DLLs MFC se construyen usando archivos .DEF.

- Se pueden exportar funciones usando el atributo `NONAME`, el cual coloca sólo el ordinal en la tabla de exportación en el DLL. Para los DLLs con un número grande de funciones exportadas, el uso del atributo `NONAME` puede reducir el tamaño del DLL.

DESVENTAJAS

Si se exportan funciones en un archivo C++, se tienen que:

- colocar los nombres decorados en el archivo `.DEF` ó
- definir las funciones exportadas con enlace C común usando `extern "C"` para evitar la decoración de nombres hecha por el compilador.

Uso de `__declspec(dllexport)`

VENTAJAS

- No se debe mantener dar mantenimiento a un archivo `.DEF` y obtener los nombres decorados de las funciones exportadas.

DESVENTAJAS

- No se tiene control sobre los ordinales de exportación que genera el compilador.
- Si se reconstruye el DLL con nuevas exportaciones, también se tiene que reconstruir la aplicación.

IMPORTAR DE UN DLL A UNA APLICACIÓN USANDO `__declspec(DLLIMPORT)`

Un programa que usa símbolos públicos definidos por un DLL se dice que los importa. Cuando se crean archivos de encabezados para las aplicaciones que usan los DLLs, se debe usar `__declspec(declspec(dllimport))` en las declaraciones de los símbolos públicos. La palabra reservada `__declspec(dllimport)` funciona con archivos `.DEF` o con la palabra reservada `__declspec(dllexport)`.

Para hacer el código más legible, definir una macro para `__declspec(dllimport)` y luego usar la macro para declarar cada símbolo exportado de la siguiente manera:

```
#define DllImport __declspec(dllimport)

DllImport int j;
DllImport void fun();
```

El uso de `__declspec(dllimport)` es opcional en la declaración de las funciones, pero el compilador produce código más eficiente si se usa esta palabra reservada. Sin embargo, se debe usar

`__declspec(dllimport)` para que el ejecutable que importa los elementos tenga acceso a los símbolos y objetos de los datos. Los usuarios del DLL aún tendrán que enlazar con una biblioteca de importación.

Se puede usar el mismo archivo de encabezados para la aplicación cliente y para el DLL.

ENLAZAR UN EJECUTABLE A UN DLL

Un archivo ejecutable se enlaza (o se carga) a un DLL de dos formas:

- Enlace implícito
- Enlace explícito

Ver la sección "Tipos de enlace" antes en este capítulo.

INICIALIZAR UN DLL

Generalmente, un DLL tiene código de inicialización (como asignación de memoria) que debe ejecutarse cuando el DLL se carga. Cuando se usa Visual C++, el lugar donde se agregue el código para inicializar el DLL depende del tipo de DLL que se construya. Si no se necesita código de inicialización o de terminación, no se requiere algo especial para construir el DLL. Si por el contrario se necesita inicializar un DLL, la siguiente tabla describe donde agregar el código.

Tabla 7 - Ubicación del código para inicializar un DLL

Tipo de MFC	Lugar para agregar el código de inicialización y terminación
DLL regular	En <code>InitInstance</code> y <code>ExitInstance</code> del objeto <code>CWinApp</code> del DLL
DLL extensión	En la función <code>DllMain</code> generada por AppWizard
DLL sin MFC	En una función llamada <code>DllMain</code> que se proporcione.

Fuente: Ayuda de Visual C++

En Win32, todos los DLLs pueden contener un punto de entrada de las funciones (generalmente llamado *DllMain*) que se llama para inicialización y terminación. Esto permite asignar o liberar recursos adicionales mientras se necesitan. Windows llama a la función de punto de entrada en cuatro situaciones: anexo de un proceso, separación de un proceso, anexo de un hilo y en la separación de un hilo.

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

La biblioteca de tiempo de ejecución de C proporciona una función de punto de entrada llamada *_DllMainCRTStartup* y esta llama a *DllMain*. Dependiendo del tipo de DLL, o se tiene una función *DllMain* en el código fuente o se usa *DllMain* proporcionada en la biblioteca MFC.

INICIALIZAR DLLS REGULARES

Debido a que los DLLs tienen un objeto *CWinApp*, deber de hacer las tareas de inicialización y terminación en el mismo lugar que una aplicación MFC: en las funciones miembro *InitInstance* y *ExitInstance* de la clase derivada de *CWinApp* del DLL. Debido a que MFC proporciona una función *DllMain* que es llamada por *DllMainCRTStartup* para *PROCESS_ATTACH* y *PROCESS_DETACH*, no se debe escribir una función *DllMain* propia. La función *DllMain* proporcionada por MFC llama a *InitInstance* cuando se carga el DLL y llama a *ExitInstance* antes de que se descargue el DLL.

El siguiente código muestra el esqueleto generado por el asistente MFC AppWizard utilizando como nombre del proyecto "DLLREG":

```
// DLLREG.h : main header file for the DLLREG DLL
//

#if !defined(AFX_DLLREG_H__BA0FC9E5_404E_11D3_B27D_003751C14E42__INCLUDED_)
#define AFX_DLLREG_H__BA0FC9E5_404E_11D3_B27D_003751C14E42__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

////////////////////////////////////
// CDLLREGApp
// See DLLREG.cpp for the implementation of this class
//

class CDLLREGApp : public CWinApp
{
public:
    CDLLREGApp();

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDLLREGApp)
    //}}AFX_VIRTUAL

    //{{AFX_MSG(CDLLREGApp)
    // NOTE - the ClassWizard will add and remove member functions here.
```

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

```
// DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before
the previous line.

#endif // !defined(AFX_DLLREG_H__BA0FC9E5_404E_11D3_B27D_003751C14E42__INCLUDED_)

// DLLREG.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "DLLREG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//
//      Note!
//
//      If this DLL is dynamically linked against the MFC
//      DLLs, any functions exported from this DLL which
//      call into MFC must have the AFX_MANAGE_STATE macro
//      added at the very beginning of the function.
//
//      For example:
//
//      extern "C" BOOL PASCAL EXPORT ExportedFunction()
//      {
//          AFX_MANAGE_STATE(AfxGetStaticModuleState());
//          // normal function body here
//      }
//
//      It is very important that this macro appear in each
//      function, prior to any calls into MFC. This means that
//      it must appear as the first statement within the
//      function, even before any object variable declarations
//      as their constructors may generate calls into the MFC
//      DLL.
//
//      Please see MFC Technical Notes 33 and 58 for additional
//      details.
//

/////////////////////////////////////////////////////////////////
// CDLLREGApp

BEGIN_MESSAGE_MAP(CDLLREGApp, CWinApp)
```

```

//{{AFX_MSG_MAP(CDLLREGApp)
// NOTE - the ClassWizard will add and remove mapping macros here.
//      DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDLLREGApp construction

CDLLREGApp::CDLLREGApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

////////////////////////////////////
// The one and only CDLLREGApp object

CDLLREGApp theApp;

```

INICIALIZAR DLLS EXTENSIÓN

Debido a que los DLLs extensión no tienen un objeto derivado de CWinApp (como los DLLs regulares), se debe agregar el código de inicialización y terminación a la función DllMain que genera AppWizard.

El siguiente código muestra el esqueleto de un DLL extensión generado por el asistente MFC AppWizard utilizando como nombre del proyecto "DLLEXT":

```

// DLLEXT.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include <afxdll.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

static AFX_EXTENSION_MODULE DLLEXTDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    // Remove this if you use lpReserved
    UNREFERENCED_PARAMETER(lpReserved);

    if (dwReason == DLL_PROCESS_ATTACH)
    {
        TRACE0("DLLEXT.DLL Initializing!\n");

        // Extension DLL one-time initialization
    }
}

```

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

```
    if (!AfxInitExtensionModule(DLLEXTDLL, hInstance))
        return 0;

    // Insert this DLL into the resource chain
    // NOTE: If this Extension DLL is being implicitly linked to by
    // an MFC Regular DLL (such as an ActiveX Control)
    // instead of an MFC application, then you will want to
    // remove this line from DllMain and put it in a separate
    // function exported from this Extension DLL. The Regular DLL
    // that uses this Extension DLL should then explicitly call that
    // function to initialize this Extension DLL. Otherwise,
    // the CDynLinkLibrary object will not be attached to the
    // Regular DLL's resource chain, and serious problems will
    // result.

    new CDynLinkLibrary(DLLEXTDLL);
}
else if (dwReason == DLL_PROCESS_DETACH)
{
    TRACE0("DLLEXT.DLL Terminating!\n");
    // Terminate the library before destructors are called
    AfxTermExtensionModule(DLLEXTDLL);
}
return 1;    // ok
}
```

INICIALIZAR DLLS SIN MFC

Para inicializar DLLs sin MFC, el código del DLL debe contener una función llamada DllMain. El siguiente código presenta un esqueleto básico que muestra la definición de DllMain:

```
BOOL APIENTRY DllMain(HANDLE hModule),
                     DWORD ul_reason_for_call,
                     LPVOID lpReserved)
{
    switch( ul_reason_for_call) {
        case DLL_PROCESS_ATTACH:
            .
            .
            .
        case DLL_THREAD_ATTACH:
            .
            .
            .
        case DLL_THREAD_DETACH:
            .
            .
            .
        case DLL_PROCESS_DETACH:
            .
            .
            .
    }
    return TRUE;
}
```

CREAR UN DLL DE RECURSOS

Un DLL de recursos es un DLL que contiene sólo recursos, tales como iconos, mapas de bits, cadenas, y cajas de diálogo. Usar DLLs de recursos es una buena forma de compartir el mismo conjunto de recursos entre múltiples programas.

Para crear un DLL de recursos, se debe crear un proyecto DLL Win32 (sin MFC) y agregar los recursos al proyecto.

Pasos para crear un proyecto de recursos:

- Seleccionar *New* del menú *File* y luego seleccionar la carpeta *Projects*. Escoger *Win32 Dynamic Link Library* como el tipo de proyecto.
- Crear un guión nuevo de recursos que contenga los recursos (como una cadena o un menú) para el DLL y guardar el archivo .RC.
- En el menú *Project*, seleccionar *Add To Project* y luego seleccionar los archivos. Usar la caja de diálogo *Insert Files into Project* para insertar el nuevo archivo .RC en el proyecto.
- Crear un archivo fuente en el directorio del DLL que contiene la siguiente función de inicialización mínima:

RECOMENDACIONES PARA USAR CADA TIPO DE DLL

La siguiente tabla resume las características (ventajas y desventajas) principales de cada tipo de DLL que puede ser construido usando Visual C++:

Tabla 8 - Características y ventajas de los diferentes tipos de DLLs

Tipo de DLL	Ventajas	Desventajas
Win32 DLL	• El tamaño del DLL es muy pequeño, lo cual se traduce en un ahorro en espacio en disco y en memoria. • No depende de DLL extras de la biblioteca MFC.	• El tipo de funciones que puede contener deben ser del estilo C común.
DLL regular	• Acceso a todas las características de la	• El tamaño del DLL es

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

enlazado estáticamente a la biblioteca MFC	biblioteca MFC. • Puede ser usado por aplicaciones construidas con MFC o sin MFC. • Puede exportar funciones estilo C común. • Puede ser usado por otras aplicaciones como Visual Basic, Pascal y Fortran. El código del DLL puede ser contenido en un solo archivo, a diferencia de los DLLs regulares enlazados dinámicamente a MFC.	mucho mayor y potencialmente consumirá memoria extra porque carga su propia copia del código de la biblioteca MFC. • Durante la construcción, el enlace es más lento debido a que cada vez que se construye el DLL se agrega también código de la biblioteca MFC.
DLL regular enlazado dinámicamente a la biblioteca MFC	• Acceso a todas las características de la biblioteca MFC. • Puede ser usado por aplicaciones construidas con MFC o sin MFC. • Puede exportar funciones estilo C común. • Se enlaza dinámicamente a MFC, lo cual significa un DLL de menor tamaño. • Puede ser usado por otras aplicaciones como Visual Basic, Pascal y Fortran.	• Se deben distribuir los DLLs MFCx0.DLL y MSVCRT.DLL (o archivos similares) junto con el DLL. • Si la versión distribuida no corresponde a la versión con la cual se construyó el DLL, cuando se use el DLL no se ejecutará.
DLL extensión de MFC	• Acceso a todas las características de la biblioteca MFC. • Se enlaza dinámicamente a MFC, lo cual significa un DLL de menor tamaño. • Soporta la interfaz C++. • Puede exportar clases y usar clases reusables derivadas de MFC, así como pasar objetos derivados de MFC entre la aplicación y el DLL.	• Las aplicaciones deben ser aplicaciones MFC.

Fuente: Ayuda de Visual C++

ARCHIVOS REDISTRIBUIBLES

Las aplicaciones construidas con Visual C++ pueden requerir un número de DLLs, dependiendo de la manera en que la aplicación se construyó.

A continuación se presenta una lista de los archivos MFC distribuibles:

Tabla 9 - Archivos distribuibles para el funcionamiento de la MFC

Archivo	Descripción
CTL3D2.DLL	Controles 3D que soportan un DLL
MFCx0.DLL	Código del núcleo MFC (habilitado para MBCS)
MFCx0U.DLL	Código del núcleo MFC para UNICODE
MSVCRT.DLL	DLL de la versión compartida de la biblioteca C en tiempo de ejecución.

Fuente: Ayuda de Visual C++

DIFERENCIAS ENTRE BIBLIOTECAS DE ENLACE ESTÁTICO Y LOS DLLS DE WINDOWS

Los DLLs de Windows se diferencian considerablemente de las bibliotecas de enlace estático. Las diferencias principales entre las dos son:

Diferencias entre las bibliotecas de enlace estático y los DLLs

	Bibliotecas de Enlace Estático	Bibliotecas de Enlace Dinámico
Ubicación de los archivos y extensión	Residen en archivos con extensión LIB y se ubican en el directorio de la aplicación o en subdirectorios	Generalmente tienen extensión DLL y residen en archivos separados tanto en el directorio de la aplicación, el directorio de Windows y en el directorio Sistema (System) de Windows
Manejo de las bibliotecas por parte de las aplicaciones	Cada aplicación tiene su propia copia de la biblioteca	Muchas aplicaciones pueden hacer uso de una sola copia de un DLL
Contenido	Código y datos porque son guardadas como una colección de archivos de objetos	Contienen código, datos y recursos como mapas de bits, iconos y cursores
Segmento de datos	Deben usar el segmento de datos de la aplicación	Pueden tener sus propios segmentos de datos

Fuente: Documentación de DLLs

DIFERENCIAS ENTRE LAS APLICACIONES Y LOS DLLS

Aunque las aplicaciones y los DLLs son módulos de programas ejecutables, difieren en varias maneras. Para el usuario final, la diferencia es que los DLLs no son programas que se puedan ejecutar directamente. Desde la perspectiva del sistema, hay dos diferencias fundamentales entre las aplicaciones y los DLLs:

Tabla 10 - Diferencia entre una instancia y un DLL

	Una aplicación	Un DLL
Instancias	Puede tener múltiples instancias de sí misma corriendo en el sistema simultáneamente	Sólo una instancia
Objetos a los que tiene acceso	Pila, memoria global, referencias a archivos y una cola de mensajes	Ninguno

Fuente: Documentación de DLLs

VENTAJAS Y DESVENTAJAS DE LOS DLLS

Tabla 11 - Ventajas y desventajas de los DLLs

	Ventajas de usar DLLs	Desventajas de usar bibliotecas de enlace estático
Memoria	Ahorran memoria y reducen la necesidad de escribir parte de la memoria al disco duro cuando la memoria física se agota. Muchos procesos pueden usar un DLL simultáneamente, compartiendo una sola copia del DLL en memoria.	El sistema operativo debe cargar una copia del código de la biblioteca en memoria para cada aplicación que se construya con una biblioteca de enlace estático.
Espacio en disco	Si un DLL es usado por varias aplicaciones se elimina la redundancia de código por cada aplicación.	Cada aplicación tiene el código de la biblioteca enlazado a su imagen ejecutable como una copia separada.
Actualizaciones	Cuando las funciones en un DLL cambian, las aplicaciones que las usan no necesitan ser recompiladas o reconstruidas siempre que los argumentos de las funciones y sus valores de regreso no cambien.	El código objeto enlazado estáticamente requiere que la aplicación se construya de nuevo cuando las funciones cambien.
Soporte a usuarios	Proporciona soporte posterior a los usuarios. Por ejemplo, un controlador de un dispositivo en un	Se necesita distribuir de nuevo la aplicación principal

CAPÍTULO 3. Bibliotecas de Enlace Dinámico

	DLL puede ser modificado para soportar un monitor que no estaba disponible cuando la aplicación se construyó.	completa.
Soporte de lenguajes	Los programas escritos en distintos lenguajes de programación pueden llamar la misma función en un DLL mientras que los programas sigan la misma convención de llamado de las funciones. Los programas y las funciones en el DLL deben ser compatibles en las siguientes formas: El orden en el cual las funciones esperan que los argumentos se pasen a la pila. Si las funciones o la aplicación es responsable de limpiar la pila. Si algún argumento se pasan en registros.	Las funciones no pueden recuperarse entre programas escritos en otros lenguajes.
POO	Proporcionan un mecanismo para extender la biblioteca de clases MFC. Se pueden derivar clases de las clases MFC existentes y ponerlas luego en un DLL extensión para usarse por aplicaciones MFC.	No es posible hacer lo mismo.
Internacionalización	Facilita la creación de versiones internacionales. Al colocar los recursos en un DLL, se pueden colocar las cadenas para cada idioma en un DLL recurso separado y hacer que las versiones del idioma se carguen con los recursos apropiados.	No puede hacerse lo mismo.
Modularidad	El software puede ser dividido en partes funcionales y cada archivo puede utilizarse por varios módulos.	

Fuente: Documentación de DLLs

Una desventaja potencial es que la aplicación no está contenida en sí misma, depende de la existencia de un archivo separado. También, los DLLs pueden ser más difíciles de desarrollar y significan más archivos en un disco.

TRAYECTO DE BÚSQUEDA USADO POR WINDOWS PARA LOCALIZAR UN DLL

Si un DLL se enlaza explícitamente con LoadLibrary, se puede especificar el trayecto completo del DLL. Si se enlaza implícitamente, Windows usa la siguiente secuencia de búsqueda para localizar el DLL:

- El directorio que contiene el archivo EXE
- El directorio actual del proceso
- El directorio de sistema de Windows (generalmente \Windows\System)
- El directorio de Windows (generalmente \Windows)
- Los directorios listados en la variable de entorno PATH

CAPÍTULO 4. Desarrollo de DLLs

Dado el objetivo principal de esta tesis, que es el de realizar un DLL en Visual C++ para leer imágenes de un disco e ilustrar su funcionamiento, se presentan en este capítulo los pasos para construir DLLs en Visual C++.

DESARROLLO DE DLLS USANDO VISUAL C++

En Visual C++ se pueden crear dos tipos de DLLs. La siguiente tabla muestra una descripción general de los dos tipos de DLLs que pueden crearse en Visual C++. Más adelante se describe con detalle cada tipo de DLL disponibles en Visual C++.

Tabla 12 - Características generales de los DLLs

Tipo de DLL	Subtipo	Características generales
Usando MFC	DLL Regular	• Puede usar clases C++ dentro y también MFC • Se enlaza estática o dinámicamente a la biblioteca MFC • Puede ser usado por cualquier aplicación MFC o sin MFC • Puede exportar funciones estilo C
	DLL Extensión	Se enlaza dinámicamente a la biblioteca MFC
DLL Win32 puro	Puede ser usado por aplicaciones con MFC o sin MFC	

Fuente: Ayuda de Visual C++

DLLS REGULARES

Los DLLs regulares son un tipo de DLLs que puede construirse en Visual C++. Este tipo de DLLs tiene la característica de usar la interfaz común de C así como clases C++. Pueden ser usados por aplicaciones MFC o por aplicaciones construidas sin enlace a MFC. El programa cliente puede estar escrito en cualquier lenguaje que soporte el uso de DLLs, tal como C, C++, Pascal, Visual Basic, etc.

Los DLLs regulares se enlazan a la biblioteca MFC internamente.

DLLS REGULARES ENLAZADOS ESTÁTICAMENTE A MFC

Esta opción sólo está disponible en las versiones profesional y empresarial de Visual C++.

Como su nombre lo describe, este tipo de DLLs se construye usando la versión de enlace estático de la biblioteca MFC. Las funciones se exportan generalmente usando la interfaz común de C.

Las características más importantes de estos DLLs son:

- El DLL se puede enlazar a las mismas bibliotecas MFC de enlace estático usadas por las aplicaciones. Ya no hay una versión separada de las bibliotecas de enlace estático para los DLLs.
- El término USRDLL usado en versiones anteriores a la 4.0 ya no se usa más.

Un DLL regular enlazado estáticamente a MFC requiere de lo siguiente:

- Debe instanciar una clase derivada de CWinApp
- Usa la función DllMain proporcionada por MFC. El código de inicialización del DLL se debe incluir en la función InitInstance y el código de terminación en ExitInstance como se haría en una aplicación MFC normal.
- Aunque el término "USRDLL" es obsoleto, todavía se debe definir la opción "_USRDLL" en la línea de comandos del compilador. Esta definición determina las declaraciones que se agregaran a los archivos de encabezados de MFC. Al usar los asistentes del ambiente integrado de Visual C++, esta opción se define automáticamente en las opciones de compilación del proyecto.

CAPÍTULO 4. Desarrollo de DLLs

Un DLL regular debe tener una clase derivada de CWinApp y un solo objeto de esa clase como una aplicación MFC. Sin embargo, el objeto CWinApp del DLL no tiene una cola de mensajes principal, como un objeto CWinApp de una aplicación.

El mecanismo CWinApp::Run no aplica a un DLL porque la aplicación posee la cola de mensajes principal.

La declaración de una función exportada de un DLL puede ser de la siguiente manera:

```
extern "C" __declspec(dllexport) Función();
```

Todas las asignaciones de memoria en un DLL regular deben permanecer dentro del DLL; el DLL no debe pasar o recibir del archivo ejecutable algo de lo siguiente:

- Apuntadores a objetos MFC
- Apuntadores a memoria asignada por MFC

Si se necesita algo de lo arriba mencionado, o si se necesita pasar objetos derivados de MFC entre el ejecutable que hace el llamado y el DLL, entonces se debe construir un DLL extensión.

Un DLL que se enlaza estáticamente a MFC no se puede enlazar dinámicamente a los DLLs MFC.

DLLS REGULARES ENLAZADOS DINÁMICAMENTE A MFC

Como su nombre lo describe, este tipo de DLL se construye usando la versión de enlace dinámico de la biblioteca MFC (también conocida como la versión compartida de MFC). Este es un tipo nuevo de DLL presentado en Visual C++ 4.0

Un DLL regular, enlazado dinámicamente a MFC tiene las siguientes características:

- A diferencia de un DLL regular enlazado estáticamente, este tipo de DLL se enlaza dinámicamente a los DLLs de MFC (también conocidos como DLLs compartidos de MFC).
- La biblioteca de importación MFC enlazada a este tipo de DLL es la misma usada para los DLLs extensión o aplicaciones usando DLLs MFC: MFCxx(D).LIB.

Un DLL regular, enlazado dinámicamente a MFC requiere de lo siguiente:

- Estos DLLs se compilan con la opción _AFXDLL definida, como cualquier otro ejecutable el cual se enlaza dinámicamente a los DLLs MFC. Tal opción es definida automáticamente por el ambiente

integrado de Visual C++ al construir el DLL regular usando los asistentes. También, se define `_USRDLL`, como un DLL regular enlazado estáticamente a MFC.

- Este tipo de DLL debe instanciar una clase derivada de `CWinApp`.
- Este tipo de DLL usa `DllMain` proporcionada por MFC. El código de inicialización específica del DLL se debe colocar en la función miembro `InitInstance` y el código de terminación en `ExitInstance` como en una aplicación MFC normal.

Debido a que este tipo de DLL usa la versión de enlace dinámico de MFC, se debe asignar explícitamente el estado actual del módulo a uno para el DLL. Para hacerlo, se debe usar la macro `AFX_MANAGE_STATE` al principio de cada función exportada del DLL.

Los DLLs regulares deben tener una clase derivada de `CWinApp` y un solo objeto de esa clase, como en una aplicación MFC. Sin embargo, el objeto `CWinApp` del DLL no tiene una cola de mensajes principal, como un objeto `CWinApp` de una aplicación.

El mecanismo `CWinApp::Run` no aplica a un DLL porque la aplicación posee la cola de mensajes principal.

Con este tipo de DLLs, se deben distribuir los DLLs compartidos `MFCx0.DLL` y `MSVCRT.DLL` (o archivos similares) con la aplicación. Ver "Archivos redistribuibles al final de este capítulo".

Un DLL que se enlaza dinámicamente a MFC no puede también enlazarse estáticamente a MFC.

Los símbolos se exportan de un DLL regular generalmente usando la interfaz común de C. La declaración de una función exportada de un DLL regular es de la siguiente manera:

```
extern "C" __declspec(dllexport) Función();
```

Todas las asignaciones de memoria en un DLL regular deben permanecer dentro del DLL, el DLL no debe pasar o recibir del archivo ejecutable cualquiera de lo siguiente:

- Apuntadores a objetos MFC
- Apuntadores a memoria asignada por MFC

Si se necesita hacer algo de lo anterior, o si se necesita pasar objetos derivados de MFC entre el archivo ejecutable y el DLL, entonces se debe construir un DLL extensión.

Se debe agregar la macro `AFX_MANAGE_STATE` al principio de todas las funciones exportadas en este tipo de DLLs para cambiar el estado del módulo actual a uno para el DLL. Esto se hace agregando la siguiente línea de código al principio de las funciones exportadas del DLL:

```
AFX_MANAGE_STATE(AfxGetStaticModuleState())
```

La macro `AFX_MANAGE_STATE` no debe usarse en DLLs regulares que se enlazan estáticamente a MFC o en DLLs extensión.

DLLS EXTENSIÓN

Un DLL extensión de MFC es un DLL que generalmente implementa clases reusables derivadas de las clases existentes de la biblioteca MFC. Los DLLs extensión se construyen usando la versión de enlace dinámico de la biblioteca MFC (también conocida como la versión compartida de MFC). Solamente los ejecutables MFC (ya sea aplicaciones o DLLs regulares) que se construyen con la versión compartida de MFC pueden usar un DLL extensión. Con un DLL extensión se pueden derivar clases nuevas de MFC y luego presentar esta versión "extendida" de MFC a las aplicaciones que llaman al DLL.

Los DLLs extensión se pueden usar también para pasar objetos derivados de MFC entre una aplicación y un DLL. Las funciones miembro asociadas con el objeto pasado existen en el módulo donde se creó el objeto. Debido a que estas funciones se exportan apropiadamente cuando se usan DLLs de la versión compartida de MFC, se puede pasar libremente apuntadores a objetos MFC o a objetos derivados de MFC entre una aplicación y los DLLs extensión que carga.

En versiones anteriores de Visual C++ a un DLL extensión se le llamaba `AFXDLL`. A partir de la versión 4.0, el término `AFXDLL` no se usa en la documentación de Visual C++. Un DLL extensión tiene las mismas características que el antiguo `AFXDLL`.

Un DLL extensión de MFC tiene las siguientes características y requerimientos:

El programa cliente debe ser una aplicación MFC compilada con la opción `_AFXDLL` definida.

- Un DLL extensión puede también ser usado por un DLL regular el cual se enlaza dinámicamente a MFC.
- Los DLLs extensión deben ser compilados con la opción `_AFXEXT` definida. Esto forzará que la opción `_AFXDLL` también sea definida y asegura que las declaraciones apropiadas serán incluidas de los archivos de encabezados MFC. Esto también asegura que la declaración `AFX_EXT_CLASS`

será definida como `__declspec(dllexport)` mientras se construye el DLL, lo cual es necesario si se usa esta macro para declarar las clases en un DLL extensión.

- Los DLLs extensión no deben instanciar una clase derivada de `CWinApp`, pero deben confiar en la aplicación cliente (o DLL) para proporcionar este objeto.
- Los DLLs extensión deben, sin embargo, contener una función llamada `DllMain` y hacer cualquier inicialización necesaria aquí.

Los DLLs extensión se construyen usando la versión de enlace dinámico de MFC (también conocido como la versión compartida de MFC). Sólo los archivos ejecutables (ya sea aplicaciones o DLLs regulares) que se construyen con la versión compartida de MFC pueden usar un DLL extensión. Ambos, el programa cliente y el DLL extensión deben usar la misma versión de `MFCx0.DLL`.

Un DLL extensión de MFC usa la versión compartida de MFC de la misma manera que una aplicación usa la versión de DLL compartida de MFC, con algunas consideraciones:

- No tiene un objeto derivado de `CWinApp`. Debe trabajar con el objeto derivado de `CWinApp` del programa cliente.
- Llama a la función `AfxInitExtensionModule` en su función `DllMain`. El valor de regreso de esta función debe ser verificado. Si `AfxInitExtensionModule` regresa 0, entonces se debe regresar este mismo valor de la función `DllMain`.
- Crea un objeto `CDynLinkLibrary` durante la inicialización si el DLL extensión desea exportar objetos de la clase `C_RUNTIME_CLASS`.

Las bibliotecas de importación para la versión compartida de MFC se nombran de acuerdo a la convención descrita en el tema "Archivos redistribuibles al final de este capítulo".

Si se exporta usando un archivo `DEF`, colocar el siguiente código al principio y al final del archivo de encabezados:

```
#undef AFX_DATA
#define AFX_DATA AFX_EXT_DATA
// <cuerpo del archivo de encabezados>
#undef AFX_DATA
#define AFX_DATA
```

Las líneas anteriores aseguran que el código será compilado correctamente para un DLL extensión. Si se omiten las líneas anteriores el DLL podría compilarse o enlazarse incorrectamente.

DLLS SIN MFC

Este tipo de DLLs no usa la biblioteca MFC internamente, y las funciones exportadas en el DLL pueden ser llamadas por aplicaciones MFC o sin MFC. Las funciones son exportadas generalmente de un DLL sin MFC usando la interfaz común de C.

Los DLL sin MFC o Win32 DLL se escriben en C o en C++ y usa llamadas al API Win32 en vez de hacerlas a la biblioteca MFC.

Uno de los cambios más significantes en los DLLs para Win32 es el lugar en memoria donde el código y los datos de un DLL residen. En Win32, cada aplicación se ejecuta dentro del contexto de su propio espacio de direcciones lineal de 32 bits. De esta manera, cada aplicación tiene su propio espacio de direcciones privado que solamente puede ser direccionado por el código dentro de ese proceso. (En Win32 cada espacio de direcciones privada se le refiere como el *proceso* para la aplicación.) Todo el código, los datos, recursos y memoria dinámica de una aplicación también residen dentro del proceso de la aplicación.

No es posible para una aplicación direccionar datos o código fuera de su propio proceso. Debido a esto, cuando un DLL es cargado debe residir de alguna manera en el proceso de la aplicación que lo cargó, y si más de una aplicación lo carga, debe residir en el proceso de cada aplicación.

Para satisfacer el requerimiento de arriba, es decir, que sea reentrante (accesible de más de un hilo al mismo tiempo) y tener solamente una copia física del DLL cargada en memoria, Win32 usa *mapeo de memoria*. Por medio del mapeo de memoria, Win32 puede cargar el DLL una vez en el montículo global (global heap) y luego mapear el rango de direcciones del DLL en el espacio de direcciones de cada aplicación que lo carga. La siguiente figura muestra como un DLL se mapea simultáneamente al espacio de direcciones de dos aplicaciones diferentes.

Note que el DLL está físicamente cargado en el montón global (global heap) solamente una vez, aunque ambas aplicaciones comparten el código y los recursos del DLL.

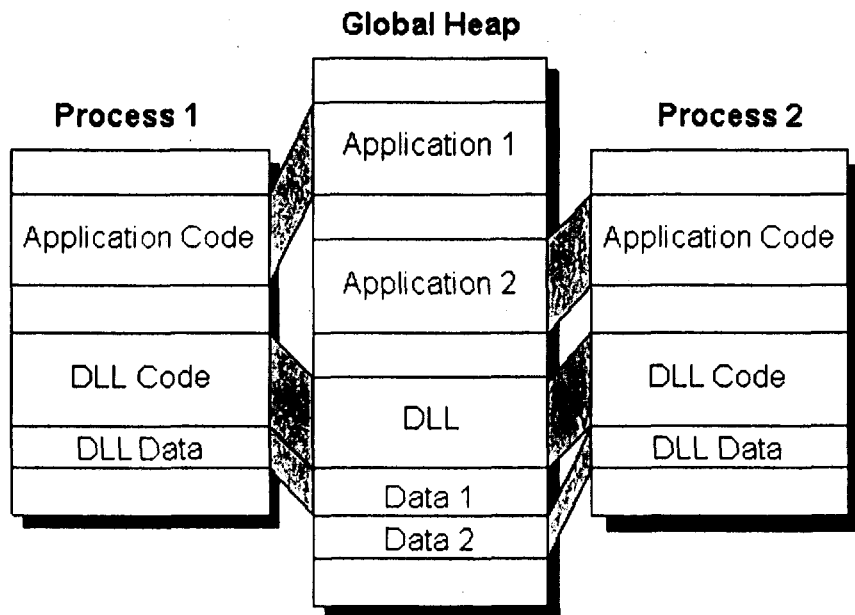
En Windows 3.x y anteriores, los DLLs tenían un conjunto de datos común que se compartía entre todas las aplicaciones que cargaban el DLL. Este comportamiento es uno de los obstáculos más difíciles para escribir código en los DLLs.

Win32 rompe esta barrera al permitir que los DLLs tengan conjuntos separados de datos, uno para cada aplicación que carga el DLL. Estos conjuntos de datos también son ubicados en el montón

(heap) global, pero no son compartidos entre las aplicaciones. La figura siguiente muestra cada conjunto de datos del DLL y como está mapeado a la aplicación a la cual pertenece. Esta nueva característica hace más fácil escribir código para los DLLs porque el DLL no tiene que cuidarse de que dos aplicaciones accedan la(s) misma(s) variable(s) global(es).

Debido a que un DLL se mapea en el proceso de la aplicación que lo cargó, puede existir alguna confusión sobre quién posee el proceso actual cuando se ejecuta código en un DLL. Llamar a la función `GetModuleHandle` con un nombre de módulo NULO como parámetro regresa el apuntador al módulo para la aplicación, no el DLL, cuando se llama desde el código de un DLL. Es una buena idea mantener una copia del apuntador del módulo del DLL en una variable estática para referencia futura cuando se accedan los recursos del DLL (El apuntador al módulo se pasa a la función `DllEntryPoint`).

Figura 4: Los DLLs son mapeados al espacio de direcciones de las aplicaciones que los cargaron.



Fuente: MSDN, Julio 1995

PASOS PARA CREAR UN DLL EN VISUAL C++

MFC APPWIZARD DLL

Para crear un DLL usando MFC Application Wizard se deben seguir los siguientes pasos:

En el menú *File*, seleccionar *New...*

Existen dos opciones en este paso:

- MFC Application Wizard (DLL) ó,
- Win32 Dynamic Link Library

La primera opción permite crear un DLL regular que soporta la biblioteca MFC incluida, o compartida o un DLL extensión de MFC. Al seleccionar *MFC AppWizard (dll)*, aparece la siguiente pantalla donde se pregunta por *MFC Regular DLL*, ya sea enlazado estáticamente o dinámicamente; o *MFC Extension DLL*.

El diálogo anterior permite especificar el tipo, nombre, ubicación y plataforma del proyecto. Al seleccionar la primera opción, *MFC AppWizard (dll)*, y completar los demás cuadros de diálogo, aparece el segundo diálogo:

1. Seleccionar el tipo de DLL, Regular o Extensión.

El diálogo anterior es el último paso para crear el DLL. Al dar un click en el botón *Finish* se crean los archivos necesarios para empezar el desarrollo.

WIN32 DLL

Los siguientes pasos describen la forma de crear un DLL Win32:

1. En el menú *File*, seleccionar *New* y luego la ficha *Projects*.
2. Especificar el nombre en el cuadro *Project Name*, la ubicación en *Location*, el ambiente de trabajo en *Workspace*, las dependencias en *Dependency*, y seleccionar la plataforma en *Platforms*. Luego dar doble click en el icono de *Dynamic Link Library*.
3. El asistente no crea archivos iniciales para este tipo de proyecto como en el caso de los DLL Regulares o Extensión. En este caso, se debe contar con archivos de inicialización ó crear archivos

nuevos que contengan el código mínimo requerido por un DLL Win32. En el capítulo 3, en la sección inicializar DLLs sin MFC, hay un ejemplo del código mínimo para compilar un DLL Win32, el cual es el punto de partida para desarrollar el DLL.

PASOS PARA CREAR UNA APLICACIÓN MFC

A continuación se listan los pasos para crear una aplicación MFC.

Seleccionar File y luego New... El diálogo siguiente muestra el diálogo donde se especifica el tipo de aplicación, la ubicación, y la plataforma.

El siguiente paso es seleccionar el tipo de aplicación que se necesita así como el idioma en que aparecerán los textos de los recursos.

En este paso hay dos opciones para una aplicación MFC AppWizard:

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

Una aplicación SDI tiene, desde el punto de vista del usuario, solamente una ventana. Si la aplicación depende en documentos de disco, solamente un documento puede ser cargado a la vez. La aplicación Cuaderno de notas de Windows es un ejemplo de una aplicación SDI.

Una aplicación MDI tiene múltiples ventanas hijo, cada una con un documento individual. Microsoft Word es una aplicación MDI en donde la aplicación permite mantener abiertos a la vez más de un documento.

El siguiente paso permite agregar características de bases de datos.

El siguiente paso permite especificar si se desea un documento compuesto y soporte para controles OLE y/o automatización.

El siguiente paso permite agregar características a la aplicación, tales como: barras flotantes, barra de estado inicial, capacidad de impresión y de vista preliminar, ayuda contextual y controles con aspecto 3D; y otras características avanzadas.

En el penúltimo paso se agregan comentarios al proyecto, opcionalmente, y se indica la manera en que la biblioteca MFC se enlazará a la aplicación.

CAPÍTULO 4. Desarrollo de DLLs

En el último paso se ven los nombres de las clases principales del proyecto y los nombres de los archivos correspondientes. También permite cambiar la clase base de la clase Vista de la aplicación.

El último diálogo muestra la información final sobre el proyecto y después de presionar el botón OK AppWizard generará los archivos necesarios para el proyecto y la aplicación estará lista para ejecutarse por primera vez.

Tabla 13: Archivos principales generados por proyectos MFC AppWizard (exe) y (dll)

Nombre del archivo	Función
ARCHIVOS DEL PROYECTO Y ARCHIVOS DE CONSTRUCCIÓN (MAKE)	
PROYECTO.DSW	Archivo del espacio de trabajo usado con el ambiente de desarrollo. Organiza todos los proyectos en un solo espacio de trabajo.
PROYECTO.OPT	Archivo de opciones del espacio de trabajo usado con el ambiente de desarrollo. Guarda todas las opciones del usuario que se crean para el espacio de trabajo, para que cada vez que se abra el espacio de trabajo del proyecto tenga la vista y las opciones que se seleccionaron la última vez.
PROYECTO.DSP	Este archivo se usa con el ambiente de desarrollo. En las versiones previas de Visual C++, la extensión de este archivo era MAK. Guarda la información específica del proyecto. Hay un archivo con extensión DSP para cada proyecto que se crea. Los archivos DSP no son compatibles con NMAKE. Se debe exportar un archivo de construcción (makefile) para construir con NMAKE.
PROYECTO.CLW	Este archivo es usado por ClassWizard y guarda la información de las clases contenidas en el proyecto.
PROYECTO.ODL	Contiene el código fuente del Lenguaje de Descripción de Objetos para una biblioteca de control de tipos. Este archivo es usado por Visual C++ para generar una biblioteca de tipos. La biblioteca generada expone la interfaz del control a otros clientes de automatización.
PROYECTO.NCB	Este archivo contiene información generada por el analizador el cual es usado por ClassView, WizardBar, y la Galería de Componentes.

CAPÍTULO 4. Desarrollo de DLLs

README.TXT	Este archivo describe cada archivo en el proyecto usando los nombres reales creados por AppWizard o ControlWizard.
ARCHIVOS FUENTE DEL PROGRAMA O DE UN CONTROL Y ARCHIVOS DE ENCABEZADOS	
PROYECTO.H	Este es el archivo principal de inclusión para un programa o un DLL. Contiene todos los símbolos globales y directivas #include para otros archivos de encabezado. Deriva la clase CProyectoApp de CWinApp y declara una función miembro InitInstance. Para un control, la clase CProyectoApp se deriva de COleControlModule.
PROYECTO.CPP	Este archivo es el archivo principal del código fuente del programa principal. Crea un objeto de la clase CProyectoApp (el cual se deriva de CWinApp) y redefine la función miembro InitInstance.
PROYECTODLG.CPP PROYECTODLG.H	Estos archivos se crean si se selecciona una aplicación basada en diálogos. Los archivos derivan e implementan la clase diálogo, llamada CProyectoDlg, e incluye esqueletos de funciones miembro para inicializar un diálogo y realizar un intercambio de datos del diálogo (DDX). La clase del diálogo Acerca de también se coloca en estos archivos en vez de hacerlo en Proyecto.CPP.
PROYECTODOC.CPP, PROYECTODOC.H	Estos archivos derivan e implementan la clase documento, llamada CProyectoDoc, e incluyen esqueletos de las funciones miembro que inicializan, serializan (guardar y cargar) un documento, e implementan el diagnóstico de depuración.
PROYECTOSET.H/ .CPP	Estos archivos se crean si el programa tiene soporte para bases de datos y contiene la clase recordset.
PROYECTOVIEW.CPP, PROYECTOVIEW.H	Estos archivos derivan e implementan la clase vista, llamada CProyectoView, que se usa para mostrar e imprimir los datos del documento. Esta clase se deriva de CView o alguna de las clases derivadas de CView (como CScrollView) y tiene esqueletos de funciones miembro para dibujar la vista e implementar el diagnóstico de depuración. Si se habilita el soporte para impresión, las entradas del mapa de mensajes se agregan para los mensajes de comandos imprimir, configurar imprimir, e impresión preliminar. Estas entradas llaman a la función miembro correspondiente en la clase vista padre.

CAPÍTULO 4. Desarrollo de DLLs

PROYECTO.PPG.H, PROYECTO.PPG.CPP	Estos archivos declaran e implementan la clase CProyectoPropPage. Esta clase se deriva de COlePropertyPage y el esqueleto de la función miembro DoDataExchange se proporciona para implementar el intercambio de datos y validación.
MAINFRM.CPP, MAINFRM.H	Estos archivos derivan la clase CMainFrame de CFrameWnd (para aplicaciones SDI) o CMDIFrameWnd (para aplicaciones MDI). La clase CMainFrame maneja la creación de botones en la barra de herramientas y la barra de estado, si las opciones correspondientes se seleccionaron en la página de opciones de la aplicación de AppWizard (paso 4 de 6).
CHILDFRM.CPP, CHILDFRM.H	Estos archivos derivan la clase CChildFrame de CMDIChildWnd. Esta clase se usa para las ventanas MDI. Estos archivos se crean si se selecciona la opción MDI.
ARCHIVOS DE ENCABEZADOS PRECOMPILADOS	
STDAFX.CPP, STDAFX.H	Estos archivos se usan para construir un archivo de encabezados precompilados PROYECTO.PCH y el archivo de tipos precompilados STDAFX.OBJ. Es importante que no se defina ni se quite la definición de cualquiera de las macros AFX_NO_XXX en stdafx.h.
ARCHIVOS DE RECURSOS	
PROYECTO.RC, RESOURCE.H	Este es el archivo de recursos para el proyecto y su archivo de encabezados. El archivo de recursos contiene la definición del menú predeterminado, las tablas de atajos y de cadenas para una aplicación MFC genérica. También especifica la caja Acerca de predeterminada y un archivo con un icono (RES\PROYECTO.ICO). El archivo de recursos incluye el archivo AFXRES.RC para los recursos de la MFC. Si se especificó una barra de herramientas, también especifica el archivo mapa de bits (RES\TOOLBAR.BMP).
PROYECTO.RC2	Este archivo puede ser incluido arriba del archivo RC en un proyecto. Un archivo RC2 es útil para incluir recursos usados por distintos proyectos. En vez de tener que crear los recursos muchas veces para distintos proyectos, se colocan en un archivo RC2 y se incluye el archivo RC2 en el archivo RC principal.

CAPÍTULO 4. Desarrollo de DLLs

PROYECTO.DEF	Si el proyecto es para un DLL, este es el archivo de definición de módulo para el proyecto. Para un control, este archivo proporciona el nombre y la descripción del control, así como el tamaño de la pila (heap) de tiempo de ejecución.
RES\PROYECTO.ICO	Contiene el icono del programa o del control MFC. El icono aparece cuando la aplicación se minimiza y también en la caja de diálogo Acerca de.
RES\PROYECTO.BMP	Este archivo se usa para representar un programa o un control en una barra de herramientas o paleta. Este archivo se incluye en el archivo de recursos del proyecto. La barra de herramientas inicial se construye en la clase CMainFrame.

Fuente: MS Visual C++

CAPÍTULO 5. Desarrollo de la aplicación

Dado el objetivo principal de esta tesis, que es el de realizar un DLL en Visual C++ para leer imágenes de un disco e ilustrar su funcionamiento, se presentan aquí los pasos para construir la aplicación de este trabajo recepcional.

DLLS PARA MOSTRAR IMÁGENES EN UNA APLICACIÓN GENERADA EN VISUAL C++

Para mostrar el funcionamiento de los DLLs, esta sección describe el DLL desarrollado para muestra del trabajo recepcional.

Todos los DLLs requieren de una aplicación o de un DLL que haga uso de sus características. La aplicación cliente que hace uso de los DLLs es una aplicación Windows tradicional. Esta aplicación generada en Visual C++, hace uso de funciones en un DLL Win32. Cuando la aplicación obtiene los datos necesarios, manda los datos al contexto del dispositivo en uso (DC) y despliega la imagen en pantalla.

CONSTRUCCIÓN DEL DLL Y DE LA APLICACIÓN CLIENTE

DLL WIN32

El DLL contiene un funciones diversas necesarias para leer datos de un archivo de imagen con formato de mapa de bits y regresar información que utilizará la aplicación cliente para desplegar la imagen en pantalla.

Construcción

Para construir el DLL Win32, se deben seguir los pasos descritos con anterioridad en "Pasos para crear un DLL en Visual C++".

Características:

- MFC Win32

Código fuente

DIBUTIL.H

```
//
// Dibutil.h
//

#define PALVERSION 0x300

// Macros útiles

// WIDTHBYTES hace la alineación de las líneas de rastreo
// del DIB (scanlines). El parámetro "bits" es la cuenta de bits
// para la línea de rastreo (biWidth * biBitCount), y esta macro
// regresa el número de bytes DWORD alineados necesarios para
// guardar (mantener) esos bits.
// WIDTHBYTES toma el # de bits en una línea (scan line)
// y la redondea a la palabra más cercana

#define WIDTHBYTES(bits) (((bits) + 31) / 32 * 4)

// Dado un apuntador a un encabezado DIB,
// regresa TRUE si es un DIB estilo Windows 3.0
// Falso de lo contrario (DIB estilo PM).

#define IS_WIN30_DIB(lpbi) ((* (LPDWORD) (lpbi)) == sizeof (BITMAPINFOHEADER))

// Número de bytes para leer durante cada operación de lectura
#define MAXREAD 32768

// Banderas para _lseek
#define SEEK_CUR 1
#define SEEK_END 2
#define SEEK_SET 0

HANDLE FAR PASCAL DIBLoad (LPSTR lpszDIBName);
DWORD PASCAL lread (int fh, VOID far *pv, DWORD ul);
DWORD FAR PASCAL DIBHeight (LPSTR lpDIB);
DWORD FAR PASCAL DIBWidth (LPSTR lpDIB);
WORD FAR PASCAL DIBNumColors (LPSTR lpbi);
LPSTR FAR PASCAL FindDIBBits (LPSTR lpbi);
WORD FAR PASCAL PaletteSize (LPSTR lpbi);
HPALETTE FAR PASCAL CreateDIBPalette (HANDLE hDIB);
```

DIBUTIL.CPP

```
// DIBUTIL.cpp : Defines the entry point for the DLL application.
//

// #include "stdafx.h"
```

```

//-----
// DIB.C
//-----

#include <windows.h>
#include <memory.h>
#include <stdio.h>
#include "dibutil.h"

// Archivo de inclusión para quitar las declaraciones Near y Far. Win32 no
// distingue en tre direcciones far y near. Los tipos FAR y NEAR están
// definidos en WINDEF.H. Por lo tanto estos tipos se manejar por el
// archivo de inclusión y se redefinen como cadenas vacías para Win32.
// #include <windef.h>

// Habilitar comprobación estricta de tipos
// #define STRICT

//-----
// DIBLoad
//
// Abre el archivo indicado por lpszDIBName y lee y copia los
// datos del DIB a un bloque de memoria asignada globalmente.
//
// lpszName - Apuntador a una cadena (nombre DOS válido) que
//             contiene el nombre del DIB.
//
// Regresa un apuntador (handle) a un bloque de memoria global
// si no hay errores o NULL si los hay.
//-----

HANDLE FAR PASCAL DIBLoad (LPSTR lpszDIBName)
{
    unsigned          fh;
    LPBITMAPINFOHEADER lpbi;
    OFSTRUCT          of;
    BITMAPFILEHEADER  bf;
    WORD              nNumColors;
    WORD              offBits;
    HANDLE            hDIBInfo = NULL;
    char *            szMsg = NULL;
    HANDLE            result = NULL;          // Error al leer los datos

    // Abrir el archivo y obtener un apuntador (handle) a su
    // BITMAPINFO

    hDIBInfo = GlobalAlloc(GMEM_MOVEABLE,
        (DWORD)(sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD)));
    if (hDIBInfo == NULL) {
        wsprintf((LPSTR) szMsg, "Memoria insuficiente para cargar '%s'",
            lpszDIBName);
        return NULL;
    }

    // Porqué no se puede omitir la sig. instrucción?
    lpbi = (LPBITMAPINFOHEADER)GlobalLock(hDIBInfo);

    fh = OpenFile (lpszDIBName, &of, OF_READ);
    if (fh == -1) {

```

```

        wsprintf((LPSTR) szMsg, "Can't open file '%s'", lpzDIBName);
        return NULL;
    }

    // Leer el BITMAPFILEHEADER
    if (sizeof (bf) != _lread (fh, (LPSTR)&bf, sizeof (bf))) {
        _lclose(fh);
        GlobalUnlock(hDIBInfo);
        return result;
    }

    if (bf.bfType != 0x4d42) { // 'BM' 0x4d42 equivale a 19778 en decimal
        _lclose(fh);
        GlobalUnlock(hDIBInfo);
        return result;
    }

    if (sizeof(BITMAPINFOHEADER) != _lread (fh,
                                             (LPSTR)lpbi,
                                             sizeof(BITMAPINFOHEADER))) {
        _lclose(fh);
        GlobalUnlock(hDIBInfo);
        return result;
    }

    // No soportar OS/2 bitmaps
    if (!IS_WIN30_DIB(lpbi)) {
        wsprintf((LPSTR) szMsg, "BMPS OS/2 (%s) no soportados!", lpzDIBName);

        _lclose(fh);
        GlobalUnlock(hDIBInfo);
        return result;
    }

    if ((nNumColors = (WORD)lpbi->biClrUsed))
    {
        // No hay tabla de colores para 24 bits,
        // se asume tamaño predeterminado
        if (lpbi->biBitCount != 24)
            nNumColors = 1 << lpbi->biBitCount; // standard size table
    }

    // Llenar algunos valores predeterminados si son cero
    if (lpbi->biClrUsed == 0)
        lpbi->biClrUsed = (DWORD)nNumColors;

    if (lpbi->biSizeImage == 0)
    {
        lpbi->biSizeImage = (((lpbi->biWidth * (DWORD)lpbi->biBitCount) + 31) & ~31)
>> 3) * lpbi->biHeight;
    }

    // Obtener un "buffer" de tamaño adecuado para el encabezado,
    // la tabla de colores y los bits
    GlobalUnlock(hDIBInfo);
    hDIBInfo = GlobalReAlloc(hDIBInfo, lpbi->biSize + nNumColors * sizeof(RGBQUAD) +
lpbi->biSizeImage, 0);

    // No se pudo redimensionar el "buffer"

```

```

if (!hDIBInfo) {
    wsprintf((LPSTR) szMsg, "Not enough memory to load '%s'", lpzDIBName);

    _lclose(fh);
    GlobalUnlock(hDIBInfo);
    return result;
}

lpbi = (LPBITMAPINFOHEADER)GlobalLock(hDIBInfo);

// Leer la tabla de colores
_lread(fh, (LPSTR)(lpbi) + lpbi->biSize, nNumColors * sizeof(RGBQUAD));

// Desplazamiento (offset) a los bits desde el inicio
// del encabezado del DIB
offBits = (WORD)lpbi->biSize + nNumColors * sizeof(RGBQUAD);

if (bf.bfOffBits != 0L)
{
    _llseek(fh, bf.bfOffBits, SEEK_SET);
}

if (lpbi->biSizeImage == _lread(fh, (LPSTR)lpbi + offBits, lpbi->biSizeImage))
    result = hDIBInfo;

_lclose(fh);
GlobalUnlock(hDIBInfo);
return result;
}

// Rutina privada para leer más de 64K
//
// _lread(int fh, VOID FAR *pv, DWORD ul)
//
// Lee los datos en incrementos de 32k hasta que todos los
// datos ya se leyeron
//
// Regresa 0 - Si la lectura no se completó correctamente
// El número de bytes si no hubo errores.

DWORD PASCAL _lread (int fh, void *pv, DWORD ul)
{
    DWORD    ult = ul;
    BYTE     *hp;

    hp = (BYTE*) pv;

    while (ul > (DWORD)MAXREAD) {
        if (!_lread(fh, (LPSTR)hp, (WORD)MAXREAD) != MAXREAD)
            return 0;
        ul -= MAXREAD;
        hp += MAXREAD;
    }
    if (_lread(fh, (LPSTR)hp, (WORD)ul) != (WORD)ul)
        return 0;

    return ult;
}

//-----

```

```

// DIBHeight
//
// Regresa el alto del DIB (como DWORD porque un Win30 DIB
// puede tener un DWORD en su campo alto, pero en Win30 el
// el primer valor (high order) de la palabra no se usa).
//-----

DWORD FAR PASCAL DIBHeight (LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;
    LPBITMAPCOREHEADER lpbmc;

    lpbmi = (LPBITMAPINFOHEADER) lpDIB;
    lpbmc = (LPBITMAPCOREHEADER) lpDIB;

    if (lpbmi->biSize == sizeof (BITMAPINFOHEADER))
        return lpbmi->biHeight;
    else
        return (DWORD) lpbmc->bcHeight;
}

//-----
// DIBwidth
//
// Regresa el ancho del DIB (como DWORD porque un Win30 DIB
// puede tener un DWORD en su campo alto, pero en Win30 el
// el primer valor (high order) de la palabra no se usa).
//-----

DWORD FAR PASCAL DIBwidth (LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;
    LPBITMAPCOREHEADER lpbmc;

    lpbmi = (LPBITMAPINFOHEADER) lpDIB;
    lpbmc = (LPBITMAPCOREHEADER) lpDIB;

    if (lpbmi->biSize == sizeof (BITMAPINFOHEADER))
        return lpbmi->biWidth;
    else
        return (DWORD) lpbmc->bcWidth;
}

//-----
// FindDIBbits
//
// lpbi - Apuntador al encabezado del DIB (BITMAPINFOHEADER)
//         o apuntador al DIB
// Regresa un apuntador a los bits del DIB
//-----
LPSTR FAR PASCAL FindDIBbits (LPSTR lpbi)
{
    return (lpbi + *(LPDWORD)lpbi + PaletteSize (lpbi));
}

//-----
// DIBNumColors
//
// Regresa el número de colores en la tabla de colores del DIB

```

```

//      lpbi - Apuntador al encabezado del DIB (BITMAPINFOHEADER)
//-----

WORD FAR PASCAL DIBNumColors (LPSTR lpbi)
{
    WORD wBitCount;

    //      Si es un DIB de Windows, el número de colores en la
    //      tabla de colores puede ser menor que el número de bits
    //      por pixel.
    //      Esto permite que a lpbi->biClrUsed se le pueda asignar
    //      un valor. Si esto sucede, regresa el valor apropiado.

    if (IS_WIN30_DIB (lpbi))
    {
        DWORD dwClrUsed;

        dwClrUsed = ((LPBITMAPINFOHEADER) lpbi)->biClrUsed;

        if (dwClrUsed)
            return (WORD) dwClrUsed;
    }

    // Calcular el número de colores en la tabla de colores
    // basado en el número de bits por pixel del DIB

    if (IS_WIN30_DIB (lpbi))
        wBitCount = ((LPBITMAPINFOHEADER) lpbi)->biBitCount;
    else
        wBitCount = ((LPBITMAPCOREHEADER) lpbi)->bcBitCount;

    switch (wBitCount)
    {
        case 1:
            return 2;

        case 4:
            return 16;

        case 8:
            return 256;

        default:
            return 0;
    }
}

//-----
//      PaletteSize
//
//      Regresa el número de bytes de la tabla de colores del DIB
//
//      lpbi - Apuntador al encabezado del DIB (BITMAPINFOHEADER)
//-----

WORD FAR PASCAL PaletteSize (LPSTR lpbi)
{
    if (IS_WIN30_DIB (lpbi))
        return (DIBNumColors (lpbi) * sizeof (RGBQUAD));
}

```

```

    else
        return (DIBNumColors (lpbi) * sizeof (RGBTRIPLE));
}

//-----
// CreateDIBPalette
//
// Dado un apuntador (handle) a un DIB, construye una paleta
// lógica y regresa un apuntador (handle) a esta paleta.
//
// hDIB - HANDLE a memoria global con un encabezado DIB
// (BITMAPINFOHEADER)
//-----

HPALETTE FAR PASCAL CreateDIBPalette (HANDLE hDIB)
{
    LPLOGPALETTE    lpPal;
    HANDLE          hLogPal;
    HPALETTE        hPal = NULL;
    int             i, wNumColors;
    LPSTR           lpbi;
    LPBITMAPINFO    lpbmi;
    LPBITMAPCOREINFO lpbmc;
    BOOL            bWinStyleDIB;

    if (!hDIB)
        return NULL;

    lpbi      = (LPSTR) GlobalLock (hDIB);
    lpbmi     = (LPBITMAPINFO) lpbi;
    lpbmc     = (LPBITMAPCOREINFO) lpbi;
    wNumColors = DIBNumColors (lpbi);
    bWinStyleDIB = IS_WIN30_DIB (lpbi);

    if (wNumColors)
    {
        hLogPal = GlobalAlloc (GHND,
                               sizeof (LOGPALETTE) + sizeof (PALETTEENTRY) * wNumColors);

        if (!hLogPal) {
            GlobalUnlock (hDIB);
            return NULL;
        }

        lpPal = (LPLOGPALETTE) GlobalLock (hLogPal);

        lpPal->palVersion = PALVERSION;
        lpPal->palNumEntries = wNumColors;

        for (i = 0; i < wNumColors; i++) {
            if (bWinStyleDIB) {
                lpPal->palPalEntry[i].peRed   = lpbmi->bmiColors[i].rgbRed;
                lpPal->palPalEntry[i].peGreen = lpbmi->bmiColors[i].rgbGreen;
                lpPal->palPalEntry[i].peBlue  = lpbmi->bmiColors[i].rgbBlue;
                lpPal->palPalEntry[i].peFlags = 0;
            }
            else {
                lpPal->palPalEntry[i].peRed   = lpbmc->bmciColors[i].rgbtRed;
                lpPal->palPalEntry[i].peGreen = lpbmc->bmciColors[i].rgbtGreen;
            }
        }
    }
}

```

```

        lpPal->palPalEntry[i].peBlue = lpbmc->bmciColors[i].rgbtBlue;
        lpPal->palPalEntry[i].peFlags = 0;
    }

    hPal = CreatePalette (lpPal);

    if (!hPal) {
        GlobalUnlock (hLogPal);
        GlobalFree (hLogPal);
        return NULL;
    }

    GlobalUnlock (hDIB);
    GlobalFree (hLogPal);
    GlobalUnlock (hDIB);

    return hPal;
}

BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    return TRUE;
}

```

APLICACIÓN CLIENTE

La aplicación cliente desarrollada para este trabajo, está construida en Visual C++. También existe otra aplicación cliente en Visual Basic descrita más adelante. Para construir la aplicación cliente en Visual C++, se siguen los pasos descritos anteriormente en este capítulo para construir una aplicación. Tal aplicación tiene las siguientes características:

- Aplicación Win32

La aplicación cliente usa el DLL descrito anteriormente. La aplicación despliega una imagen, en formato Windows Bitmap, que se puede seleccionar de un archivo en disco. Las instrucciones para mostrar la imagen están contenidas el DLL.

A continuación se describe el proceso para completar la aplicación cliente junto con el código fuente.

Desarrollo

Las aplicaciones en Windows, como se menciona en el capítulo 1, reciben mensajes que sistema operativo (Windows) les manda para procesar ciertas acciones. Los principales mensajes utilizados por la aplicación son WM_COMMAND y WM_PAINT.

A continuación se lista el código fuente de la aplicación.

Código fuente

DIBEXE.H

```
// Dibexe.h
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_DIBEXE_H__1A86BEA8_DACF_11D3_B01F_E01A52C10000__INCLUDED_)
#define AFX_DIBEXE_H__1A86BEA8_DACF_11D3_B01F_E01A52C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif //_MSC_VER > 1000

#include "resource.h"

#endif // !defined(AFX_DIBEXE_H__1A86BEA8_DACF_11D3_B01F_E01A52C10000__INCLUDED_)

// Funciones usadas en DIBEXE.cpp

BOOL MenuCommand (HWND hWnd, WPARAM wParam);
```

DIBEXE.CPP

```
// Dibexe.cpp
//

// DIBEXE.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include <string.h>

#include "commdlg.h"
#include "dib.h"
#include "dibexe.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                // current instance
```

```

TCHAR szTitle[MAX_LOADSTRING]; // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING]; // The title bar text

HPALETTE hPalette = NULL, hOldPalette = NULL;
HBITMAP hBitmap = NULL; // Handle para el BMP actual */
// HANDLE hCompatDC = NULL; // Handle para el la estructura info del BMP actual*/
HGLOBAL hDIBBuffer;
LPVOID lpDIBBuffer;
static UINT bmpWidth, bmpHeight;
BOOL bRegreso, bDraw;

// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    // TODO: Place code here.
    MSG msg;
    HACCEL hAccelTable;

    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_DIBEXE, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_DIBEXE);

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// PURPOSE: Registers the window class.
//
// COMMENTS:

```

CAPÍTULO 5. Desarrollo de la aplicación

```
// This function and its usage is only necessary if you want this code
// to be compatible with Win32 systems prior to the 'RegisterClassEx'
// function that was added to Windows 95. It is important to call this function
// so that the application will get 'well formed' small icons associated
// with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = (WNDPROC)WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_DIBEXE);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = (LPCSTR)IDC_DIBEXE;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

    return RegisterClassEx(&wcex);
}

//
// FUNCTION: InitInstance(HANDLE, int)
//
// PURPOSE: Saves instance handle and creates main window
//
// COMMENTS:
//
//     In this function, we save the instance handle in a global variable and
//     create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;

    hInst = hInstance; // Store instance handle in our global variable

    hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

//
// FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
```

```

//
// PURPOSE: Processes messages for the main window.
//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    // int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc, hCompatDC;
    TCHAR szHello[MAX_LOADSTRING];
    LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

    LPBITMAPINFOHEADER lpbi;
    HGDIOBJ hTemp;

    switch (message)
    {
        case WM_CREATE:
            break;

        case WM_COMMAND:
            /* Process menu commands */
            return MenuCommand(hWnd, LOWORD(wParam));
            break;

        case WM_PAINT:

            hdc = BeginPaint(hWnd, &ps);

            if (bDraw) {
                // Seleccionar la paleta
                hOldPalette = SelectPalette(hdc, hPalette, FALSE);
                bRegreso = RealizePalette(hdc);

                // Asegurar el bloque de memoria
                lpbi = (LPBITMAPINFOHEADER) GlobalLock ((HANDLE) hDIBBuffer);

                // Crear un DDB
                hBitmap = CreateDIBitmap(hdc,
                    (LPBITMAPINFOHEADER) lpbi,
                    CBM_INIT,
                    FindDIBBits ((LPSTR) lpDIBBuffer),
                    (LPBITMAPINFO) lpbi,
                    DIB_RGB_COLORS);

                if (hPalette) {
                    hOldPalette = SelectPalette(hdc, hPalette, FALSE);
                    bRegreso = RealizePalette(hdc);
                }

                // Crear un DC de memoria del cual BitBlt la imagen.
                // Seleccionarlo y realizar la paleta, luego seleccionarlo en el DDB
                hCompatDC = CreateCompatibleDC(hdc);
                hOldPalette = SelectPalette(hdc, hPalette, FALSE);
                bRegreso = RealizePalette(hCompatDC);
            }
    }
}

```

CAPÍTULO 5. Desarrollo de la aplicación

```

        hTemp = SelectObject (hCompatDC, hBitmap);

        // Mostrar la imagen
        bRegreso = BitBlt (hdc, 0, 0,
                           bmpWidth, bmpHeight, hCompatDC, 0, 0, SRCCOPY);

        // Liberar recursos que ya no necesitamos
        bRegreso = DeleteDC (hCompatDC);
        if (!bRegreso)
            MessageBox (hWnd, "No se pudo borrar hCompatDC", "DIB Exe", MB_OK);

        bRegreso = DeleteObject (hBitmap);
        if (!bRegreso)
            MessageBox (hWnd, "No se pudo borrar hBitmap", "DIB Exe", MB_OK);

        bRegreso = GlobalUnlock (hDIBBuffer);
    }

    EndPaint(hWnd, &ps);
    break;

case WM_DESTROY:
    if (hDIBBuffer) {
        bRegreso = (BOOL) GlobalFree (hDIBBuffer);
        hDIBBuffer = NULL;
    }

    if (hPalette) {
        DeleteObject(hPalette);
        hPalette = NULL;
    }

    PostQuitMessage(0);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

BOOL MenuCommand (HWND hWnd, WPARAM id)
{
    OPENFILENAME ofn;
    RECT rect, rectClient;

    static char szFilter[] = "Mapas de bits (*.bmp)\0*.bmp\0"
                           "DIBs (*.dib)\0*.dib\0"
                           "Todos los archivos (*.*)\0*.*\0\0";

    static char pstrFileName[256];
    static char pstrTitleName[] = "Visualizador de BMPs";

    ofn.lStructSize      = sizeof (OPENFILENAME) ;
    ofn.hwndOwner         = hWnd;
    ofn.hInstance         = NULL;
    ofn.lpstrFilter       = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter    = 0 ;
    ofn.nFilterIndex      = 0 ;
    ofn.lpstrFile         = pstrFileName ;      // Set in Open and Close functions

```

```

ofn.nMaxFile      = 256 ;
ofn.lpstrFileName = pstrTitleName ;          // Set in Open and Close functions
ofn.nMaxFileTitle = _MAX_FNAME + _MAX_EXT ;
ofn.lpstrInitialDir = NULL ;
ofn.lpstrTitle     = NULL ;
ofn.Flags          = 0 ;                      // Set in Open and Close functions
ofn.nFileOffset    = 0 ;
ofn.nFileExtension = 0 ;
ofn.lpstrDefExt     = "txt";
ofn.lCustData       = 0L ;
ofn.lpfnHook        = NULL ;
ofn.lpTemplateName = NULL ;

switch (id)
{
case IDM_OPEN:

    if (GetOpenFileName(&ofn)) {
        hDIBBuffer = DIBLoad (pstrFileName);

        if (!hDIBBuffer) {
            MessageBox (hWnd,
                "Error al abrir el archivo", "Programa DIBEXE", MB_OK);
            return FALSE;
        }
        lpDIBBuffer = GlobalLock (hDIBBuffer);
        bmpWidth = DIBWidth ((LPSTR) lpDIBBuffer);
        bmpHeight = DIBHeight ((LPSTR) lpDIBBuffer);
        bRegreso = GlobalUnlock (hDIBBuffer);

        if (hPalette) {
            bRegreso = DeleteObject (hPalette);
        }
        if (bRegreso == 0)
            MessageBox (hWnd, "No se pudo borrar la paleta existente",
                "Programa DIBEXE", MB_OK);
    }

    // Crear una paleta nueva para este BMP
    hPalette = CreateDIBPalette (hDIBBuffer);

    bDraw = TRUE;

    rect.top = 0;
    rect.bottom = bmpHeight;
    rect.left = 0;
    rect.right = bmpWidth;

    GetClientRect (hWnd, &rectClient);

    AdjustWindowRect (&rect, WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
        WS_MAXIMIZEBOX | WS_MINIMIZEBOX | WS_THICKFRAME, TRUE);
    SetWindowPos (hWnd, (HWND)NULL, 0, 0,
        rect.right - rect.left + 1,
        rect.bottom - rect.top + 1, SWP_NOMOVE | SWP_NOZORDER);
    GetClientRect (hWnd, &rectClient);

    // Corrección para bmps pequeños que causan menús multilínea
    /* if (rectClient.bottom < rect.bottom) {
        rect.bottom += (rect.bottom - rectClient.bottom);
    }

```

```

        SetWindowPos (hWnd, (HWND)NULL, 0, 0,
                      rect.right - rect.left + 1,
                      rect.bottom - rect.top + 1,
                      SWP_NOMOVE | SWP_NOZORDER);
    }
    /*
    InvalidateRect(hWnd, NULL, TRUE);
}
// InvalidateRect(hWnd, NULL, FALSE);

break;

case IDM_ABOUT:
    DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
    break;

case IDM_EXIT:
    DestroyWindow(hWnd);
    break;

default:
    // return DefWindowProc(hWnd, message, wParam, lParam);
    break;
}

return TRUE;
}

// Mensaje para el diálogo Acerca de
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            return TRUE;

        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
    return FALSE;
}

```

APLICACIÓN CLIENTE EN VISUAL BASIC

El código para la aplicación cliente desarrollada en Visual Basic 3.0 para este trabajo, se lista a continuación.

Código fuente

PICBAS.BAS

```

Global Const Default = 0      ' 0 - Default
Global Const ARROW = 1       ' 1 - Arrow
Global Const HOURGLASS = 11  ' 11 - Hourglass

Declare Function FindDIBBits Lib "DIBUTIL" (ByVal lpbi&) As Long
Declare Function DibNumColors Lib "DIBUTIL" (ByVal lpbi&) As Integer
Declare Function PaletteSize Lib "DIBUTIL" (ByVal lpbi&) As Integer
Declare Function CreateDIBPalette Lib "DIBUTIL" (ByVal hDIB&) As Integer 'HPAL
Declare Function DIBHeight Lib "DIBUTIL" (ByVal lpDIB&) As Long
Declare Function DIBWidth Lib "DIBUTIL" (ByVal lpDIB&) As Long
Declare Function DIBLoad Lib "DIBUTIL" (ByVal lpszDIBName$) As Integer 'hDIBInfo

Global Const MF_BYCOMMAND = &H0
Global Const MF_BYPOSITION = &H400

Global Const SRCCOPY = &HCC0020

Global Const CBM_INIT = &H4

Type BitmapFileHeader
    bftype As Integer
    bftype As Integer
    bftype As Integer
    bfReserved1 As Integer
    bfReserved2 As Integer
    bfOffBits As Long
End Type

Type BitmapInfoHeader
    biSize As Long
    biWidth As Long
    biHeight As Long
    biPlanes As Integer
    biBitCount As Integer
    biCompression As Long
    biSizeImage As Long
    biXPelsPerMeter As Long
    biYPelsPerMeter As Long
    biClrUsed As Long
    biClrImportant As Long
End Type

Declare Function GlobalFree Lib "Kernel" (ByVal hMem&) As Integer
Declare Function GlobalLock Lib "Kernel" (ByVal hMem&) As Long
Declare Function GlobalUnlock Lib "Kernel" (ByVal hMem&) As Integer

Declare Function CreateCompatibleDC Lib "GDI" (ByVal hDC&) As Integer
Declare Function SelectObject Lib "GDI" (ByVal hDC&, ByVal hObject&) As Integer
Declare Function BitBlt Lib "GDI" (ByVal hDestDC&, ByVal X&, ByVal Y&, ByVal nWidth&,
ByVal nHeight&, ByVal hSrcDC&, ByVal XSrc&, ByVal YSrc&, ByVal dwRop&) As Integer
Declare Function DeleteDC Lib "gdi" (ByVal hDC&) As Integer
Declare Function SelectPalette Lib "USER" (ByVal hDC As Integer, ByVal hPalette As
Integer, ByVal bForceBackground As Integer) As Integer
Declare Function RealizePalette Lib "USER" (ByVal hDC As Integer) As Integer
Declare Function CreateDIBitmap Lib "GDI" (ByVal hDC As Integer,

```

CAPÍTULO 5. Desarrollo de la aplicación

```
ByVal lpInfoHeader As Long,
ByVal dwUsage As Long,
ByVal lpInitBits As Long,
ByVal lpInitInfo As Long,
ByVal wUsage As Integer) As Integer
Declare Function DeleteObject Lib "GDI" (ByVal hObject As Integer) As Integer
Declare Function GetDC Lib "USER" (ByVal hWnd As Integer) As Integer
Declare Function ReleaseDC Lib "USER" (ByVal hWnd As Integer,
ByVal hDC As Integer) As Integer
Declare Function GetProfileString Lib "Kernel" (ByVal lpAppName As String,
ByVal lpKeyName As String,
ByVal lpDefault As String,
ByVal lpReturnedString As String,
ByVal nSize As Integer) As Integer
Declare Function GetSystemMenu Lib "User" (ByVal Handle As Integer, ByVal bRevert As Integer) As Integer
Declare Function RemoveMenu Lib "User" (ByVal hMenu As Integer,
ByVal Position As Integer, ByVal Flags As Integer) As Integer
Declare Function LoadLibrary Lib "Kernel" (ByVal lpzFileName As String) As Integer
Declare Sub FreeLibrary Lib "Kernel" (ByVal hModule As Integer)
```

PICFRM.FRM

```
Option Explicit

' Apuntador al DLL
Dim hDIBUtil As Integer

' Apuntador a una paleta para ver un BMP
Dim hPalette As Integer

' Apuntador a memoria global para leer el DIB
Dim hDIBBuffer As Integer

' Apuntador a una memoria temporal que contiene el DIB
Dim lpDIBBuffer As Long

Sub cbDrives_Change ()
    lblDirList.Path = cbDrives.Drive
End Sub

Sub Form_Load ()
    Dim StartingDir As String

    'Establecer directorio inicial

    StartingDir = CurDir
    lblDirSelected.Caption = StartingDir
    cbDrives.Drive = StartingDir
    lblDirList.Path = StartingDir

    hDIBUtil = LoadLibrary("DIBUTIL.DLL")

    hPalette = 0

    frm256picbx.WindowState = 2

End Sub
```

```

Sub Form_Unload (Cancel As Integer)
    Dim ReturnVal As Integer

    If (hPalette <> Null) Then
        ReturnVal = DeleteObject(hPalette)
        If (ReturnVal = 0) Then
            MsgBox "(Form_Unload) Warning: Could not delete hPalette!"
        End If
    End If
    FreeLibrary hDIBUtil
End Sub

Function GetProfileInfo (Section As String, KeyName As String) As String
    Dim Result As String
    Dim ResultLength As Integer

    Result = String(255, 0)
    ResultLength = GetProfileString(Section, KeyName$, "", Result$, 255)
    GetProfileInfo = Left$(Result, ResultLength)
End Function

Sub lbDirList_Change ()
    labDirSelected.Caption = UCase$(lbDirList.Path)
    lbFileList.Path = lbDirList.Path
End Sub

Sub lbFileList_DblClick ()
    Call pbPreview_Click
End Sub

Sub lbFileList_PathChange ()
    lbFileList.ListIndex = -1
End Sub

Sub pbCancel_Click ()
    Unload frm256picbx
End
End Sub

Sub pbPreview_Click ()
    Dim SrcX, SrcY, DestX, DestY As Integer
    Dim bmpWidth, bmpHeight As Long
    Dim ShowFiller As Integer
    Dim OldMousePtr As Integer
    Dim FName As String
    Dim ReturnVal As Integer
    Dim DestWidth As Integer
    Dim DestHeight As Integer
    Dim hOldPalette As Integer
    Dim hBitmap As Integer
    Dim hCompatDC As Integer
    Dim hPrevBmp As Integer
    Dim Success As Integer

    ' Si no se seleccionó un archivo, tomar el primero
    If lbFileList.ListIndex < 0 Then
        lbFileList.ListIndex = -1
    End If

```

```

        Screen.MousePointer = OldMousePtr
    Exit Sub
End If

'Borrar BMPs existentes
PicPreview.Visible = True
PicPreview.Picture = LoadPicture()

FName = lbDirList.List(lbDirList.ListIndex)
If Right(FName, 1) <> "\" Then FName = FName + "\"
FName = FName + lbFileList.List(lbFileList.ListIndex)

' Cargar el BMP en memoria
hDIBBuffer = DIBLoad(FName)

If (hDIBBuffer = 0) Then
    MsgBox "Error al cargar " + FName + "!"
    Exit Sub
End If
lpDIBBuffer = GlobalLock(hDIBBuffer)
bmpWidth = DIBWidth(lpDIBBuffer)
bmpHeight = DIBHeight(lpDIBBuffer)
RetVal = GlobalUnlock(hDIBBuffer)

'Establecer los parámetros de bitblt
SrcX = 0: SrcY = 0
DestX = 0: DestY = 0
PicPreview.Width = bmpWidth
PicPreview.Height = bmpHeight

If (bmpWidth > PicPreview.ScaleWidth) Then
    DestWidth = PicPreview.ScaleWidth
Else
    DestWidth = bmpWidth
End If
If (bmpHeight > PicPreview.ScaleHeight) Then
    DestHeight = PicPreview.ScaleHeight
Else
    DestHeight = bmpHeight
End If

'Si ya existe una paleta, borrarla
If (hPalette <> Null) Then
    RetVal = DeleteObject(hPalette)
    If (RetVal = 0) Then
        MsgBox "(pbPreview_Click) Warning: Could not delete hPalette!"
    End If
End If

'Crea una paleta nueva para este BMP
hPalette = CreateDIBPalette(hDIBBuffer)

'Seleccionar la paleta que creamos
hOldPalette = SelectPalette(PicPreview.hDC, hPalette, False)
RetVal = RealizePalette(PicPreview.hDC)

'Asignar un bloque de memoria
lpDIBBuffer = GlobalLock(hDIBBuffer)

```

```

' Crear un DDB
' DIB_RGB_INFO = 0
hBitmap = CreateDIBitmap(PicPreview.hDC, lpDIBBuffer, CBM_INIT,
FindDIBBits(lpDIBBuffer), lpDIBBuffer, 0)

' Crear un DC de memoria del cual transferiremos el
' bloque de bits de la imagen
' Seleccionar la paleta y realizarla, luego seleccionar el DDB
hCompatDC = CreateCompatibleDC(PicPreview.hDC)
hOldPalette = SelectPalette(hCompatDC, hPalette, False)
ReturnVal = RealizePalette(hCompatDC)
hPrevBmp = SelectObject(hCompatDC, hBitmap)

' Transmitir la imagen en el cuadro de la imagen
Success = bitblt(PicPreview.hDC, 0, 0, bmpWidth, bmpHeight, hCompatDC, 0, 0,
SRCCOPY)

' Liberar memoria ocupada
ReturnVal = DeleteDC(hCompatDC)
If (ReturnVal = 0) Then
    MsgBox "cmdVer imagen, Advertencia: No se pudo borrar hComptDC%!"
End If
ReturnVal = DeleteObject(hBitmap)
If (ReturnVal = 0) Then
    MsgBox "cmdVer imagen, Advertencia: No se pudo borrar hBitmap%!"
End If

' Liberar memoria asinada anteriormente
Success = GlobalUnlock(hDIBBuffer)
Success = GlobalFree(hDIBBuffer)
hDIBBuffer = 0
End Sub

```

CONCLUSIONES

Visual C++ es una herramienta poderosa para el desarrollo de aplicaciones en la cual el desarrollo de proyectos puede tomar una forma muy ordenada. MFC simplifica el trabajo inicial de crear una aplicación. Gracias a la programación orientada a objetos, con el buen diseño de un desarrollo, se aprovecha su característica principal, la reutilización del código.

Las muchas ventajas que ofrece esta herramienta tienen un precio muy alto, que es la curva de aprendizaje. Debido a que Windows es un sistema operativo complejo (en comparación al MS-DOS) se deben vigilar más comportamientos del sistema. Así mismo, se tiene acceso a muchas características que permiten hacer casi cualquier cosa, pero el ir encontrando cada una, a veces es desalentador por la documentación tan superficial que se encuentra en la mayoría de los libros y mientras no se encuentra la instrucción exacta no se puede ver el resultado del programa.

A medida que el desarrollo de un proyecto aumenta, es cada vez más difícil examinar el código en busca de errores, seguir aumentando código y una vez finalizado, hacer modificaciones y agregar nuevas características.

Los DLLs ofrecen un método eficiente de hacer modular un proyecto y reutilizar código. Al agrupar funciones y recursos aislados de la aplicación principal, disminuye significativamente el exceso de información a examinar y aumenta la atención a elementos determinados que son parte del proyecto general.

Un equipo de desarrollo puede trabajar de manera separada en diversos módulos y al final integrarlos en una aplicación.

Durante el mantenimiento del sistema, se pueden hacer mejoras o cambios en archivos DLL específicos sin la posibilidad de corromper el sistema principal. De la misma manera, el tamaño de las actualizaciones son menores y disminuyen los costos de almacenamiento o distribución.

Al desarrollar un DLL se debe probar exhaustivamente para tener la certeza de que no fallará en un momento dado, sobre todo si las aplicaciones son críticas, pero una vez que se tiene un DLL se puede hacer uso de toda la funcionalidad de su contenido con unos cuantos llamados.

Apéndices

APÉNDICE A RETROSPECTIVA

El desarrollo de un DLL generalmente se realiza usando una herramienta de desarrollo en C/C++ como Visual C++. Aunque existen otros compiladores de compañías como Symantec o Inprise (antes Borland), las herramientas de Microsoft son las más conocidas y también las de mayor difusión.

El uso masivo permite una aceptación mayor por parte de los usuarios, y por parte de la compañía es de esperarse una amplia compatibilidad con las versiones nuevas tanto de lenguajes de desarrollo como de sistemas operativos, incluso cuando hay cambios significativos como fue el caso del salto de los sistemas operativos de 16 bits a 32 bits donde el código anterior pudo emplearse en gran parte haciendo algunos cambios.

Visual Basic tiene la opción de desarrollar DLLs pero con un enfoque es distinto. Visual Basic, a partir de la versión 5, puede crear DLLs ActiveX. Este concepto está enfocado hacia los componentes de software. Visual C++ también permite crear estos componentes, aunque el proceso de construcción de DLLs comunes y de componentes ActiveX está claramente separado.

Los objetos o funciones de una biblioteca tradicional dependen de su entorno; por ejemplo, la biblioteca MFC sólo puede ser utilizada con C++. En cambio, un componente tiene la ventaja de que es un elemento común prácticamente aplicable a cualquier plataforma y con independencia de las aplicaciones y el lenguaje con el que estén construidas. Además por el hecho de ser una norma no encontraremos problemas en las actualizaciones futuras, que en ocasiones pueden presentar las bibliotecas de objetos.

El modelo de componentes ha estado presente desde la introducción de OLE, que es un sistema de manipulación de objetos basado en el modelo de componentes COM, y todas las tecnologías derivadas, como son los controles OCX y en la actualidad, ActiveX.

Antes de este concepto, los desarrollos de código reutilizable estaban soportados exclusivamente por bibliotecas de objetos o de funciones como la MFC. Actualmente el desarrollo de software empieza a incorporar componentes de software. De cualquier manera, en ambos casos existe la idea de la reutilización de código, encapsulación y abstracción.

Sin embargo, el proyecto original de este trabajo, fue asignado específicamente a la construcción de DLLs en Visual C++.

La construcción de DLLs comunes en Visual C++ en versiones anteriores a la 4.0 permitía construir DLLs tanto de 16 bits como de 32 bits. A partir de la versión 4.0 de Visual C++, todos los proyectos construidos son orientados hacia la plataforma de 32 bits.

Desde la versión 4.0 de Visual C++, al paquete de desarrollo se le han agregado opciones como aplicaciones con soporte para Internet a través de clases extras de la MFC, soporte para ATL, y en la versión más reciente, soporte para la creación de extensiones ISAPI, entre otros.

Con respecto al proceso de construcción de DLLs, no ha habido cambios significativos en el proceso desde la versión 4 hasta la versión más reciente, la versión 6.

Los proyectos que crean el ambiente de desarrollo sufrieron ligeras modificaciones en los archivos que crea como proyecto. Anteriormente, la configuración del proyecto estaba contenida en un archivo con extensión MDP. Ahora, el ambiente de desarrollo crea dos archivos, uno que contiene la configuración del entorno del proyecto (con extensión .DSW) y otro con la configuración del proyecto mismo (con la extensión .DSP)

Para la construcción de DLLs en Visual C++ existen dos opciones:

- DLLs sin MFC
- DLLs MFC

La segunda opción permite construir DLLs MFC regulares (con MFC enlazada estática o dinámicamente) o un DLL extensión de MFC.

La versión más reciente de Visual Basic, la versión 6, permite construir, entre otras cosas, controles ActiveX, ActiveX ejecutables y DLLs ActiveX.

Los sistemas operativos más recientes de Microsoft, como son Windows 2000 y Windows NT 5.0 siguen dependiendo en gran medida de DLLs. Windows (en cualquiera de sus versiones) es un conjunto

de DLLs y para muestra, basta con ver el contenido del directorio de sistema Windows (normalmente c:\windows\system o c:\winnt\system32 para el caso de Windows NT).

Es de esperarse que sigan existiendo los DLLs por varios años más debido a que son una manera muy conveniente de modularizar el software además de su utilidad al hacer cambios no previstos.

Como ejemplo tenemos las actualizaciones continuas a sistemas operativos y a aplicaciones ante la presencia de fallas no detectadas o inconvenientes de cambiar hasta el momento de la liberación del producto. Dichas actualizaciones permiten hacer cambios en un producto determinado sin tener que instalar el producto completo otra vez o hacer una descarga completa e instalarlo de nuevo. Generalmente, el proceso de actualización consiste en ejecutar un archivo que automáticamente busca los elementos que deben reemplazarse y copia las versiones recientes.

Sin los DLLs, sería necesario obtener una versión actualizada del producto (lo cual representa grandes gastos al fabricante para distribuir versiones corregidas de su producto o un gran inconveniente a la mayoría de usuarios al descargar de Internet, si existiera la opción, un producto seguramente de tamaño considerable).

APÉNDICE B - REFERENCIA DE DUMPBIN

El descargador de Archivos Binarios COFF de Microsoft (DUMPBIN.EXE) muestra información sobre los archivos binarios del Formato de Archivos de Objetos Comunes de 32 bits (COFF, Common Object File Format). DUMPBIN se usa para examinar archivos de objetos COFF, bibliotecas comunes de objetos COFF, archivos ejecutables, y bibliotecas de enlace dinámico.

DUMPBIN sólo se puede ejecutar desde la línea de comandos.

LÍNEA DE COMANDOS DE DUMPBIN

A continuación se muestra la sintaxis para ejecutar DUMPBIN:

DUMPBIN [opciones] archivos...

Para controlar la información, se deben especificar uno o más archivos binarios, junto con cualquier opción requerida. DUMPBIN muestra la información en la salida común. También se puede redirigir la salida a un archivo o usar la opción /OUT para especificar el nombre del archivo para la salida.

Cuando se ejecuta DUMPBIN en un archivo sin especificar una opción, DUMPBIN despliega la salida de la opción /SUMMARY.

Cuando se ejecuta DUMPBIN sin opciones y sin especificar un archivo, DUMPBIN muestra enunciados de uso que resumen sus opciones.

OPCIONES DE DUMPBIN

/ALL

Esta opción muestra toda la información disponible excepto el código desensamblado.

/ARCHIVEMEMBERS

Esta OPCIÓN muestra información mínima sobre los objetos de miembros en la biblioteca.

Por ejemplo:

El tamaño del archivo de salida dependerá del número de módulos de objetos en la biblioteca.

/DISASM

Esta opción muestra secciones de código desensamblado, usando símbolos si están presentes en el archivo.

El siguiente ejemplo es parte del código desensamblado de LIB.EXE:

```
00401000: 8B 44 24 08      mov     eax,dword ptr [esp+8]
00401004: 68 00 20 40 00   push    402000h
00401009: 8B 4C 24 08      mov     ecx,dword ptr [esp+8]
0040100D: 50              push    eax
0040100E: 51              push    ecx
0040100F: E8 0C 00 00 00   call    00401020
00401014: 33 C0           xor     eax,eax
00401016: C3              ret
00401017: CC              int     3
```

/EXPORTS

Esta opción muestra todas las definiciones exportadas de un archivo ejecutable o un DLL. Por ejemplo, el siguiente listado muestra la información de exportación de MMFC40.DLL:

```
0 characteristics
30733286 time date stamp Wed Oct 04 18:19:02 1995
0.00 version
1 ordinal base
5782 number of functions
4 number of names

ordinal hint  name
2 0 DllCanUnloadNow (00096182)
1 1 DllGetClassObject (0009611E)
3 2 DllRegisterServer (000961DB)
4 3 DllUnregisterServer (00096242)
```

/HEADERS

Esta opción muestra el encabezado del archivo y el encabezado para cada sección. Cuando se usa con una biblioteca, muestra la información del encabezado para cada objeto miembro.

Ejemplo:

La siguiente información pertenece al compilador de recursos, RC.EXE:

```
FILE HEADER VALUES
14C machine (i386)
```

```

    6 number of sections
306F7A22 time date stamp Sun Oct 01 22:35:30 1995
    0 file pointer to symbol table
    1D1 number of symbols
    E0 size of optional header
    302 characteristics
        Executable
        32 bit word machine
        Debug information stripped

```

OPTIONAL HEADER VALUES

```

    10B magic #
    2.60 linker version
    1E00 size of code
    1E00 size of initialized data
    0 size of uninitialized data
    1144 address of entry point
    1000 base of code
    3000 base of data
    ----- new -----
2BB0000 image base
    1000 section alignment
    200 file alignment
    3 subsystem (Windows CUI)
    4.00 operating system version
    4.00 image version
    3.50 subsystem version
    8000 size of image
    400 size of headers
    62C8 checksum
100000 size of stack reserve
    1000 size of stack commit
100000 size of heap reserve
    1000 size of heap commit
    0 [ 0] address [size] of Export Directory
    5000 [ 3C] address [size] of Import Directory
    6000 [ 394] address [size] of Resource Directory
    0 [ 0] address [size] of Exception Directory
    0 [ 0] address [size] of Security Directory
    7000 [ 21C] address [size] of Base Relocation Directory
    3030 [ 38] address [size] of Debug Directory
    0 [ 0] address [size] of Description Directory
    0 [ 0] address [size] of Special Directory
    0 [ 0] address [size] of Thread Storage Directory
    0 [ 0] address [size] of Load Configuration Directory
    268 [ 44] address [size] of Bound Import Directory
    50A0 [ 64] address [size] of Import Address Table Directory
    0 [ 0] address [size] of Reserved Directory
    0 [ 0] address [size] of Reserved Directory
    0 [ 0] address [size] of Reserved Directory

```

SECTION HEADER #1

```

    .text name
    1D24 virtual size
    1000 virtual address
    1E00 size of raw data
    400 file pointer to raw data
    0 file pointer to relocation table
    3C20 file pointer to line numbers

```

```

0 number of relocations
37E number of line numbers
60000020 flags
    Code
    (no align specified)
    Execute Read

```

/IMPORTS

Esta opción muestra todas las definiciones importadas de un archivo ejecutable o un DLL.

La salida mostrada por esta opción es similar a la salida de /EXPORTS.

/OUT nombredelarchivo

Esta opción especifica el nombre del archivo para la salida. Por defecto, DUMPBIN muestra la información a la salida común.

RAWDATA

Esta opción muestra el contenido en bruto de cada sección en el archivo. Los argumentos controlan el formato de lo que se muestra, como se ve a continuación:

Tabla 14 - Argumentos de Rawdata

Argumento	Resultado
BYTES	Predeterminado. El contenido se muestra en bytes hexadecimales, y también como caracteres ASCII si tienen una representación imprimible.
SHORTS	El contenido se muestra en palabras largas hexadecimales.
NONE	Se omiten los datos en bruto. Este argumento es útil para controlar la salida de la opción /ALL.
Number	Las líneas mostradas se ajustan a un ancho que mantiene los valores de los números por línea.

Fuente: Ayuda de Visual C++

Ejemplo: La información generada con esta opción es larga. El siguiente listado es una muestra de la salida de una aplicación AppWizard usando el formato LONGS:

```

RAW DATA #1
00000000 53EC8B55 00A15756 | E9004058 00000000      Ui_SVWi.|X@....._

```

```

00000010 C95B5E5F EC8B55C3 | 5304EC83 4D895756    _^ [ ++Ui_ | a_ .SVWeM_
00000020 3058B8FC 00E90040 | 5F000000 C3C95B5E    n+X0@._. | ... _^ [ ++_

```

/SECTION

Esta opción restringe la información a la sección especificada.

/SUMMARY

Esta opción muestra información breve de las secciones, incluyendo su tamaño total. Esta opción es la predeterminada si no se especifica alguna.

El siguiente ejemplo muestra el contenido del archivo tracer.exe

Dump of file tracer.exe

File Type: EXECUTABLE IMAGE

Summary

```

1000 .data
1000 .idata
1000 .rdata
1000 .reloc
1000 .rsrc
1000 .text

```

/SYMBOLS

Esta opción muestra la tabla de símbolos COFF. Las tablas de símbolos existen en todos los archivos de objetos. Una tabla de símbolos COFF aparece en la imagen de un archivo sólo si se enlaza con las opciones Generar Información de Salida y el formato COFF bajo la categoría Debug del enlazador (o las opciones /DEBUG y /DEBUGTYPE:COFF en la línea de comandos).

El siguiente listado es de una aplicación generada por AppWizard:

COFF SYMBOL TABLE

```

000 000065D0 SECT5  notype      Static      | .rsrc$02
001 00000011 DEBUG notype      Filename   | .file
C:\code\test\testView.cpp
004 000041A0 SECT3  notype      Static      | $$G23289
005 00004018 SECT3  notype      Static      | _$$189$$S23301
006 00004168 SECT3  notype      Static      | _THIS_FILE$$S23282
007 00001E90 SECT1  notype      Static      | .text
Section length 2DE, #relocs 27, #linenums 22
009 00001EE2 SECT1  notype      Static      | _$E188
00A 00001EF7 SECT1  notype      Static      | _$E187
tag index 0000002E size 0000001F lines 000012DF next function 00000033

```

Apéndices

00C	000042AC	SECT3	notype	Static	__init_CTestView\$S23291
00D	00003998	SECT2	notype	Static	\$T23440
00E	00003978	SECT2	notype	Static	\$T23438
00F	000039C0	SECT2	notype	Static	\$T23447
010	000039A0	SECT2	notype	Static	\$T23446
011	00000021	DEBUG	notype	Filename	.file

APÉNDICE C NOMBRES DECORADOS

Las funciones en C y C++ se conocen internamente por sus nombres decorados. Un nombre decorado es una cadena creada por el compilador durante la compilación de la definición de la función o prototipo.

USO DE LOS NOMBRES DECORADOS

En la mayoría de las circunstancias, no se necesita saber el nombre decorado de una función. LINK u otras herramientas pueden manejar el nombre en su forma sin decorar.

Sin embargo, ciertas situaciones requieren que se especifique el nombre decorado. Se debe especificar el nombre decorado de las funciones en C++ que se redefinen (overloaded functions) y funciones miembro especiales, como constructores y destructores, para que LINK y otras herramientas puedan igualar el nombre. También se deben usar nombres decorados en los archivos fuente en ensamblador que hagan referencia a una función en C o C++.

Advertencia:

Si se cambia el nombre de la función, clase, convención de llamada, tipo de regreso, o cualquier otro parámetro, el nombre decorado ya no es válido. Se debe obtener la nueva versión del nombre de la función y usarlo en todos lados donde el nombre decorado se especifica.

FORMATO DE UN NOMBRE DECORADO C++

Un nombre decorado para una función C++ contiene la siguiente información:

- Nombre de la función
- Clase de la cual la función es miembro, si es miembro de alguna.
- Los tipos de datos de los parámetros de la función.
- La convención de llamado.
- El tipo de datos del valor que regresa la función.

La función y los nombres de la clase se codifican en el nombre decorado. El resto del nombre decorado es un código que tiene significado interno solamente para el compilador y el enlazador. Los siguientes son ejemplos de nombres C++ decorados y no decorados:

Tabla 15 - Nombres decorados y no decorados

Nombre sin decorar	Nombre decorado
<code>int a(char){int i=3;return i;};</code>	<code>?a@@YAHD@Z</code>
<code>void __stdcall b::c(float){};</code>	<code><u>?c@b@@AAGXM@Z</u></code>

Fuente: Ayuda de Visual C++

FORMATO DE UN NOMBRE DECORADO EN C

La forma de decoración para una función en C depende de la convención de llamada usada en la declaración, como se muestra abajo:

Tabla 16 - Formato de un nombre decorado

Convención de llamado	Decoración
<code>__cdecl</code> (the default)	Guión bajo (<code>_</code>)
<code>__stdcall</code>	Guión bajo (<code>_</code>) y un signo <code>@</code> seguido de un número de bytes en la lista de parámetros.
<code>__fastcall</code>	Lo mismo que <code>__stdcall</code> , pero precedida por el signo <code>@</code> en vez de un guión bajo.

Fuente: Ayuda de Visual C++

FORMA DE VER LOS NOMBRES DECORADOS

Se puede obtener la forma decorada del nombre de una función después de compilar el archivo fuente que contiene la definición de la función o prototipo. La forma más fácil para examinar los nombres decorados es usar la herramienta DUMPBIN. Ver el uso de DUMPBIN antes en esta sección.

APÉNDICE D ARCHIVOS DE DEFINICIÓN DE MÓDULOS (.DEF)

Un archivo de definición de módulos (.DEF) es un archivo de texto que contiene enunciados para definir a un archivo .EXE o .DLL. Las siguientes secciones describen los enunciados en un archivo .DEF.

Debido a que LINK proporciona opciones de línea de comandos equivalentes para la mayoría de los enunciados de definición de módulos, un programa típico Win32 no requiere un archivo .DEF. Las descripciones de los enunciados de definición de módulos son el equivalente a la línea de comandos para cada enunciado.

REGLAS PARA LOS ENUNCIADOS DE DEFINICIÓN DE MÓDULOS

Las siguientes reglas de sintaxis aplican a todos los enunciados en un archivo .DEF. Otras reglas que aplican a enunciados específicos se describen con cada enunciado:

- Las palabras reservadas de los enunciados y atributos no deben coincidir mayúsculas y minúsculas. Los identificadores especificados por el usuario sí deben coincidir en mayúsculas y minúsculas.
- Los nombres de archivos largos que contienen espacios o puntos y comas (;) deben encerrarse en comillas ("").
- Usar uno o más espacios, tabuladores, o caracteres de nueva línea para separar un enunciado de palabra reservada de sus argumentos y para separar los enunciados unos de otros. Los dos puntos (:) o el signo de igual (=) que designan un argumento se rodean de cero o más espacios, tabuladores o caracteres de nueva línea.
- Un enunciado NAME o LIBRARY, si se usa, debe preceder cualquier otro enunciado.
- La mayoría de los enunciados deben aparecer máximo una vez en el archivo .DEF y aceptar una especificación de argumentos. La especificación sigue al enunciado palabra reservada de la misma o siguiente línea de argumentos. Si el enunciado se repite con diferentes argumentos después en el archivo, el enunciado posterior anula el anterior.
- Los enunciados SECTIONS, EXPORTS, e IMPORTS puede aparecer más de una vez en el archivo .DEF. Cada enunciado puede tomar múltiples especificaciones, las cuales deben estar separadas por uno o más espacios, tabuladores, o caracteres de línea nueva. El enunciado palabra reservada debe

aparecer una vez antes de la primera especificación y puede repetirse antes de cada especificación adicional.

- Los comentarios en los archivos .DEF se designan con un punto y coma (;) al principio de cada línea comentario. Un comentario no puede compartir una línea con un enunciado, pero puede aparecer entre las especificaciones en un enunciado multilineas. (SECTIONS y EXPORTS son enunciados multilinea).
- Los argumentos numéricos se especifican en notación decimal o de lenguaje C.
- Si una cadena de argumentos es igual a una palabra reservada, debe encerrarse en comillas (").

NAME

Sintaxis: NAME [aplicación]

Este enunciado especifica un nombre para un archivo de salida principal. Una manera de especificar el nombre de archivo de salida es con la opción /OUT del enlazador la cual redefine el nombre y ubicación predeterminada del programa que crea el compilador. (En el ambiente de desarrollo de Visual C++, esta opción se encuentra en el menú Settings/Project/carpeta Link y categoría Customize).

LIBRARY

Sintaxis: LIBRARY [library]

Este enunciado le dice a LINK que cree un DLL. Al mismo tiempo, LINK crea una biblioteca de importación, a menos que un archivo .EXP se use en la construcción.

El argumento library especifica el nombre interno del DLL. (La opción /OUT permite especificar el nombre de salida del DLL.)

Una manera equivalente de especificar la construcción de un DLL es usar la opción /DLL.

DESCRIPTION

Sintaxis: DESCRIPTION "Text"

Este enunciado escribe una cadena en una sección .rdata. Se debe encerrar la descripción entre comillas ("). Para usar una comilla en la descripción (sencilla o doble) en la cadena, encerrar la cadena con otro tipo de signo.

STACKSIZE

Sintaxis: STACKSIZE reserve

Este enunciado asigna el tamaño de la pila en bytes.

SECTIONS

Sintaxis: SECTIONS definitions

Este enunciado establece los atributos de una o más secciones en la imagen del archivo. Puede usarse para redefinir los atributos predeterminados de cada tipo de sección.

SECTIONS marca el comienzo de una lista de definición de secciones. Cada definición debe estar en una línea separada. La palabra reservada SECTIONS puede estar en la misma línea como la primera definición o en una línea precedente. El archivo .DEF puede contener uno o más enunciados SECTIONS. La palabra reservada SEGMENTS se considera como sinónimo de SECTIONS.

La sintaxis para una definición de sección es:

sections [CLASS 'classname'] attributes

El nombre de la sección distingue entre mayúsculas y minúsculas. La palabra reservada CLASS se soporta por compatibilidad, pero se ignora. Los atributos son uno o más de los siguientes: EXECUTE, READ, SHARED y WRITE.

EXPORTS

Sintaxis: EXPORTS definitions

Este enunciado hace que una o más definiciones estén disponibles como exportaciones a otros programas.

EXPORTS marca el principio de una lista de definiciones de exportación. Cada definición debe estar en una línea separada. La palabra reservada EXPORTS puede estar en la misma línea como la primera definición o en una línea precedente. El archivo .DEF puede contener uno o más enunciados EXPORTS.

La sintaxis de una definición de exportación es:

Entryname[=internalname] [@ordinal[NONAME]] [DATA] [PRIVATE]

La palabra reservada **PRIVATE** le dice a **IMPLIB** que ignore la definición. **PRIVATE** previene al nombre de la entrada de ser puesto en la biblioteca de importación. La palabra reservada no tiene efecto en **LINK**.

Hay tres métodos de exportar una definición, listada en orden recomendado de uso:

- La palabra reservada `__declspec(dllexport)` en el código fuente
- Una especificación `/EXPORTS` en un comando **LINK**
- Un enunciado de exportación en un archivo `.DEF`.

Los tres métodos pueden usarse en el mismo programa. Cuando **LINK** construye un programa que contiene exportaciones, también crea una biblioteca de importación, a menos que se use un archivo `.EXP` en la construcción.

VERSION

Sintaxis: **VERSION** *major*[*.minor*]

Este enunciado le dice a **LINK** que ponga un número en el encabezado del archivo `.EXE` o `.DLL`. Los argumentos *major* y *minor* son números decimales en el rango 0 hasta 65535. La versión predeterminada es 0.0.

Glosario

Hay algunos términos básicos alrededor de la programación en Windows que son usados con mucha frecuencia.

TÉRMINOS NO ALFABÉTICOS

.BMP

Archivo que contiene una colección de estructuras que especifican o contienen los siguientes elementos:

- Un encabezado que describe la resolución del dispositivo en el cual el rectángulo de píxeles se creó, las dimensiones del rectángulo, el tamaño del arreglo de bits, etc.
- Una paleta lógica.
- Un arreglo de bits que define la relación entre los píxeles en la imagen de mapa de bits y las entradas en la paleta lógica.

.DEF

Archivo de texto que contiene uno o más enunciados que describen varios atributos de un módulo ejecutable. Los archivos de definición de módulos generalmente tienen una extensión .DEF.

.C

Un archivo de texto que contiene código en lenguaje C.

.CPP

O archivo .CXX. Archivo de texto que contiene código fuente de lenguaje C++.

.DIB

Archivo de texto que contiene un arreglo de bits combinado con varias estructuras que especifican el ancho y alto de la imagen de mapa de bits (en píxeles), el formato de color del dispositivo donde se creó la imagen, y la resolución del dispositivo usado para crear la imagen. El formato DIB asegura que la gráfica creada en una aplicación puede ser cargada y desplegada en cualquier otra aplicación exactamente de la misma manera en que aparece en la aplicación que la originó.

.DLL

Archivo que contiene una o más funciones que se compilan, se enlazan, y se guardan separadamente del proceso que las usa.

.EXE

Archivo de programa creado de uno o más archivos de código fuente que se convierten a código máquina y se enlazan juntos. En los sistemas operativos de Microsoft esta extensión indica que el archivo es un programa ejecutable.

.EXP

Archivo que contiene información sobre funciones exportadas y datos. La Herramienta de Administración de Bibliotecas (LIB.EXE) de 32 bits de Microsoft genera el archivo de exportación a partir del archivo de definición de módulo (.DEF). El enlazador usa el archivo de exportación para construir la biblioteca de enlace dinámico (.DLL).

.H

Header file, archivo de encabezados. Archivo de código fuente externo, identificado al principio de un programa escrito en lenguaje C, que contiene tipos de datos usados comúnmente y variables usadas por funciones en el programa. La directiva `#include` se usa para decir al compilador que inserte el contenido de un archivo de encabezados en el programa.

.HPP

O archivo .HXX. Archivo de código externo, identificado al principio de un programa escrito en lenguaje C++, que contiene tipos de datos usados comúnmente y variables usadas por funciones en un programa. La directiva `#include` se usa para decir al compilador que inserte el contenido de un archivo de encabezados en el programa.

.ICO

En Windows, un archivo que contiene un mapa de bits de un icono. Los archivos de iconos generalmente tienen esta extensión.

.LIB

Archivo en Formato de Archivo de Objetos Comunes (COOF, Common Object File Format) generado por la herramienta de 32 bits de administración de bibliotecas, LIB, para las bibliotecas comunes y de importación.

.MAK

(Visual C++ 4 y anteriores) (MAKE File) Archivo que contiene todos los comandos, definiciones de macros, opciones, y demás para especificar como construir el proyecto en un ambiente de trabajo. El archivo de construcción puede usarse para construir usando una aplicación diferente a Visual C++.

.OBJ

Archivo que contiene código objeto y/o datos generados por un compilador o un ensamblador del código fuente de un programa. Los archivos objeto generados por compiladores Visual C++ tienen una extensión

.OBJ.

.RC

O Resource Script File, Archivo Guión de recursos. Archivo de texto que contiene descripciones de recursos de los cuales el compilador de recursos crea un archivo de recursos binario. Para las aplicaciones Windows, los archivos de definición de recursos generalmente tienen una extensión .RC.

.RES

Archivo de recursos binario. Archivo binario que contiene los datos de recursos de una aplicación Windows y es creado por el compilador de recursos a partir del archivo de definición de recursos (.RC). Los archivos de recursos compilados generalmente tienen una extensión .RES.

16-BIT

Programa escrito para un sistema como MS-DOS o Windows 3.x, que usa arquitectura segmentada de 16 bits en la cual cada dirección de memoria apunta a una palabra de 16 bits.

32-bit

Se refiere al número de bits que pueden ser procesados o transmitidos en paralelo, o el número de bits usados para un solo elemento en un formato de datos. El término se aplica generalmente a lo siguiente:

Microprocesador: Un microprocesador de 32 bits puede procesar datos y direcciones de memoria que son representadas por 32 bits.

Bus: Indica el número de alambres en el bus. Un bus de 32 bits transmite 32 bits en paralelo.

Dispositivo de gráficos (digitalizador o cámara digital): Especifica el número de bits usados para representar cada pixel. Típicamente se usan 24 bits para el color y los 8 bits restantes se usan para la información de control.

Sistema operativo: Se refiere al número de bits usados para representar direcciones de memoria.

Las aplicaciones de 32 bits, escritas para los sistemas operativos Windows NT o Windows 9x, usan arquitectura de 32 bits en la cual cada dirección de memoria apunta a una palabra de 32 bits.

A

API

Application Programming Interface. El API (Interfaz de Programación de Aplicaciones) es un conjunto de rutinas que una aplicación usa para pedir y realizar servicios de bajo nivel ejecutados por el sistema operativo. En Windows, el API también apoya a las aplicaciones en el manejo de ventanas, menús, iconos y otros elementos de la interfaz gráfica de usuario. En una red de área local (LAN), el API proporciona rutinas a las aplicaciones para solicitar servicios de los niveles bajos de la red.

Aplicación cliente

Aplicación o proceso que solicita un servicio de otro proceso.

Apuntador

Tipo de datos que contiene la dirección de otro tipo de datos como una variable, o una función o una clase.

Apuntador/Identificador (Handle)

Ver Handle.

Apuntador/Identificador de/A archivo (File Handle)

Identificador de archivo. Identificador único que Windows asigna a un archivo cuando el archivo se abre o se crea. Un apuntador a archivo es válido hasta que el archivo se cierra.

Apuntador/Indicador de instancia (Instance handle)

Un identificador que Windows asigna a cada copia de una aplicación o un DLL cargado en un sistema multitarea. Cada clase ventana requiere un identificador de instancia para identificar a la aplicación o al DLL que registró la clase.

Apuntador/Identificador de ventana (window handle)

En el API Win32, un valor de 32 bits (asignado por Windows) que identifica únicamente a una ventana. Una aplicación usa este identificador para dirigir las acciones de funciones a la ventana. Un identificador de ventana tiene el tipo de datos HWND, una aplicación deberá usar este tipo cuando se declare una variable que guarde un identificador de ventana.

Archivo biblioteca

Archivo en Formato de Archivo de Objetos Comunes (COFF) generado por la herramienta de administración de bibliotecas de 32 bits de Microsoft, LIB, para las bibliotecas comunes y las de importación.

Archivo binario

Un archivo que contiene datos binarios o código ejecutable. Por ejemplo, un archivo ejecutable C++ es un archivo binario.

Archivo de definición de módulo (.def)

Archivo de texto que contiene uno o más enunciados que describen varios atributos de un módulo ejecutable.

Archivo de definición de recursos

O (archivo) guión de recursos. Archivo de texto que contiene las descripciones de las cuales el compilador de recursos crea un archivo de recursos binario. En las aplicaciones Windows, los archivos de definición de recursos generalmente tienen la extensión .RC.

Archivo de elaboración o de construcción (Makefile)

Archivo que contiene todos los comando, definiciones de macros, opciones y demás para especificar como construir los proyectos en un espacio de trabajo de un proyecto. Un archivo de elaboración tiene la extensión .MAK y generalmente usa el mismo nombre base que el archivo de configuración del espacio de trabajo (.DSP o .MDP).

Archivo de encabezados (header file)

Archivo fuente externo identificado al principio de un programa, que contiene declaraciones de tipos de datos usados, variables usadas, declaración de funciones y clases en un programa. La instrucción #include se usa para indicarle al compilador que inserte el contenido del archivo de encabezados.

Archivo de encabezados precompilado

Archivo que contiene código compilado para una porción de un proyecto. La construcción subsecuente combina este archivo con el código sin compilar, de esta forma reduce el tiempo de compilación total. La extensión para un archivo de encabezados precompilado es .PCH.

Archivo de exportación

Archivo que contiene información sobre las funciones exportadas y datos. La herramienta Administradora de bibliotecas de 32 bits de Microsoft (LIB.EXE) genera el archivo de exportación

a partir del archivo de definición del módulo (.DEF). El enlazador usa el archivo de exportación para construir la biblioteca de enlace dinámico (DLL). Los archivos de exportación tienen una extensión de nombre .EXP.

Archivo de importación

Archivo que describe las funciones y los datos a exportar cuando el enlazador hace referencia a un archivo de exportación (.EXP). El enlazador (LINK.EXE) usa el archivo de exportación para construir un programa que contiene exportaciones (generalmente un DLL), y usa la biblioteca de importación para resolver las referencias a esas exportaciones en otros programas. Las bibliotecas de importación generalmente tienen una extensión .LIB.

Archivo objeto

Archivo que contiene código objeto y/o datos generados por un compilador o ensamblador a partir del código fuente de un programa. Los archivos objeto generados por el compilador Visual C++ tienen una extensión .OBJ.

Área cliente

O rectángulo cliente. La porción de una ventana donde la aplicación despliega la salida tal como texto o gráficos.

Argumento

Valor o expresión usada con un operador o pasada a un subprograma (subrutina, procedimiento, o función). El programa luego ejecuta operaciones usando los argumentos. Sinónimo de parámetro.

Arquitectura Documento/Vista

Metodología de diseño que se enfoca en lo que el usuario ve y necesita en lugar de la aplicación o lo que la aplicación necesita. Este diseño está implementado mediante un conjunto de clases que administran, guardan y presentan los datos específicos de la aplicación. Estas clases pueden manipular archivos de datos basados en disco (objetos documento), desplegar los datos de un documento (objetos vista), y usan automáticamente un tipo particular de ventana (objetos ventana).

B

Biblioteca

Una colección de programas o archivos de datos. También, es una colección de funciones o subrutinas preescritas. Las funciones se unen al programa principal cuando éste se compila.

Biblioteca de Clases

Conjunto de clases y rutinas para usarse en programas C++ cuya razón de existir es permitir que los programadores usen el código de alguien más en sus desarrollos.

Biblioteca compartida

En general, cualquier módulo de código que se puede accesarse y usarse por muchos programas. Las bibliotecas compartidas se usan principalmente para compartir código común entre diferentes archivos ejecutables o para dividir una aplicación en componentes separados, de esta manera las actualizaciones son más fáciles. En Windows, a las bibliotecas compartidas generalmente se les conoce como bibliotecas de enlace dinámico (DLLs).

Biblioteca de Enlace Estático

Un archivo biblioteca que se enlaza a un programa cuando el archivo ejecutable se construye. Los archivos biblioteca de enlace estático generalmente tienen extensión LIB.

BIOS

Basic Input/Output System. Conjunto de rutinas que trabajan estrechamente con el hardware y el sistema operativo para dar soporte a la transferencia de información entre los elementos del sistema, como la memoria, los discos, y el monitor.

Brocha lógica

Descripción ideal de un mapa de bits brocha. Una brocha lógica describe todos los atributos (estilo, color, y demás) especificados por la aplicación que los creó, aunque algunos no sean representables en los dispositivos de salida disponibles. Windows proporciona siete brochas lógicas predefinidas.

Buffer

Ver Memoria Intermedia.

Bus

(1) Conjunto de alambres por el cual los datos las señales eléctricas pasan entre los componentes del sistema. un bus de datos es un trayecto interno a través del cual los datos son transferidos hacia y desde el procesador. Los datos se transmiten de una parte de la computadora a otra parte. El bus conecta todos los componentes internos de la computadora, el CPU, la memoria y los dispositivos de E/S.

En las computadoras personales, el término bus generalmente se refiere al bus interno.

Tipos de bus:

En la familia de procesadores 80x86, existen los principales buses son: el bus de datos, el bus de direccionamiento y el bus de control.

Bus de datos: Transfiere los datos reales.

Bus de direccionamiento: Transfiere información sobre donde deben ir los datos.

Bus de expansión: Hace posible que las tarjetas de expansión accedan al CPU y la memoria.

Un componente de un sistema 80x86 típico usa niveles lógicos TTL. Esto significa que cada alambre en un bus usa un voltaje común para representar cero y uno. Los niveles de voltaje varían dependiendo del procesador (especialmente en las computadoras tipo cuaderno).

Cuando se conecta una tarjeta de circuito impreso en una de las ranuras de expansión de una computadora personal, se le está conectando al bus.

El tamaño de un bus se conoce como su ancho, es importante porque determina que cantidad de datos que pueden ser transmitidos en un tiempo dado. Por ejemplo, un bus de 16 bits puede transmitir 16 bits de datos, mientras que un bus de 32 bits puede transmitir 32 bits de datos.

Cada bus tiene una velocidad de reloj medida en MegaHertz. Un bus rápido permite a los datos transmitirse más rápido, lo cual hace que las aplicaciones corran rápido. En las PCs, el antiguo bus ISA está siendo reemplazado por buses más rápidos como el PCI.

Casi todas las PCs que se fabrican hoy en día incluyen un bus local para los datos que requieren velocidades de transmisión rápidas como los datos de video. El bus local es un camino rápido que se conecta directamente al procesador.

(2) En redes, un bus es un cable central que conecta todos los dispositivos en una red de área local. También se le llama columna vertebral (backbone).

Byte

Unidad de información que consiste 8 bits. Un byte, o "binary term, término binario", es la colección de bits más pequeña que puede accederse directamente.

C**Código máquina**

Resultado de la compilación de lenguaje ensamblador o cualquier lenguaje de alto nivel que se carga y se ejecuta por un procesador. El código máquina es el único lenguaje que entienden las

computadoras, cualquier otro lenguaje representa formas en que los humanos hacen que las computadoras realicen tareas específicas.

Código objeto

Código ejecutable generado por un compilador o ensamblador a partir el código fuente de un programa. Ver también archivo código.

Código sin decorar

En c++, la forma en que un identificador tiene en el código fuente (como una cadena de caracteres entendibles por el humano) en contraste a su nombre decorado, el cual consiste de símbolos que son significantes solamente al compilador y al enlazador.

COFF

Common Object File Format. En programación de 32 bits, un formato para los archivos ejecutables y para los archivos objeto que es portable entre plataformas. La implementación COFF de Microsoft se deriva de una especificación de UNIX para COFF, pero incluye encabezados especiales por compatibilidad con MS-DOS y Windows de 16 bits. Esta versión de Microsoft a veces se le llama "Formato de archivos ejecutables portable (PE)"

Cola (Queue)

Estructura de datos en la cual los elementos se agregan al final de una lista y se quitan desde la cabeza de la lista. Una cola de prioridad generalmente quita los elementos de la lista de acuerdo a algún valor de prioridad asignado a cada elemento. La cola de mensajes del sistema, por ejemplo, retiene la entrada del ratón y del teclado en espera de ser procesados.

Cola (de mensajes) de la aplicación (Application Queue)

En Windows, un depósito para los mensajes que espera a ser procesados por una aplicación. Como Windows quita los mensajes de su cola del sistema, manda los mensajes que contienen la entrada (del usuario) de la aplicación a la cola de mensajes de la aplicación apropiada.

Color derivado

Color del área cliente de una ventana vacía o una pantalla.

Compilación

Traducción de un código fuente a código objeto.

Compilador

Programa que traduce código fuente, como C++ o Pascal, en código de máquina ejecutable directamente.

Compilador de recursos

Aplicación que crea un archivo de recursos binario basado en el archivo de definición de recursos (.RC). El compilador de recursos puede también anexar datos de recursos binarios a un archivo ejecutable y crear una tabla de recursos en el encabezado del archivo ejecutable.

Construir (Build)

Proceso de compilar y enlazar código fuente para generar un programa ejecutable, una biblioteca, un archivo de ayuda, u otro archivo ejecutable.

Controlador (Driver)

Controlador. Un programa que controla un dispositivo. Cada dispositivo, sea una impresora, una unidad de disco o un teclado, debe tener un programa controlador. Muchos controladores, como el controlador del teclado, están incluidos con el sistema operativo.

Un controlador actúa como un traductor entre el dispositivo y los programas que usan el dispositivo. Cada dispositivo tiene su propio conjunto especializado de comandos que solamente su controlador conoce. En contraste, la mayoría de los programas accesan dispositivos usando comandos genéricos. El controlador, sin embargo, acepta comandos genéricos de un programa y luego los traducen en comandos especializados para el dispositivo.

Controlador de dispositivo (Device driver)

Controlador de dispositivo. Ver driver.

Convención de llamada C

La norma C para llamar a una función, esto es, mover los argumentos a la pila de la derecha a la izquierda (en orden inverso a la manera en que aparecen en la lista de argumentos). Después de que la función regresa, la función quita los argumentos de la pila. La convención de llamada C permite pasar un número variable de argumentos.

Convención de llamada Pascal

Convención de llamada que manda los argumentos a la pila en el orden en que aparecen en la llamada de la función (de izquierda a derecha). La convención de llamada Pascal requiere que la rutina que llama la función quite los argumentos de la pila.

Convención de llamado

Convención que determina el orden en que los argumentos pasados a las funciones se mueven a la pila (la secuencia de llamado). ya sea la función que hace el llamado o la función llamada quita los argumentos de la pila, y la convención de decoración de nombres que usa el compilador para identificar las funciones individuales. Ver también **secuencia de llamado**.

D

Decoración de nombres

Proceso durante la compilación de la definición de una función o prototipo en C++ en el cual el compilador crea una cadena que identifica únicamente a una función. La decoración de nombres es necesaria para que el compilador y el enlazador distinga entre varias funciones que pudieran tener el mismo nombre.

Depurador

Programa diseñado para ayudar a encontrar errores en otro programa permitiendo al programador ir paso a paso a través del programa, examinar los datos, y verificar las condiciones. Existen dos tipos de depuradores: Depuradores a nivel máquina, que despliegan las instrucciones reales de la máquina (desensambladas en lenguaje ensamblador) y permiten al programador ver en los registros y en las posiciones de memoria. Y el segundo tipo son los depuradores a nivel de código fuente, que permiten al programador ver el código fuente original, examinar las variables y las estructuras de datos por nombre, y demás.

Dirección

El camino a un valor, objeto, documento, página, u otro destino. Una dirección puede ser un lugar en memoria, un URL (dirección a un sitio en Internet), etc.

Dispositivo (Device)

Cualquier máquina o componente que se conecte a una computadora. Ejemplos de dispositivos incluyen unidades de disco, impresoras, ratones y modems. Estos dispositivos particulares caen en la categoría de dispositivos periféricos porque están separados de la computadora principal. Los monitores y teclados son también dispositivos, pero debido a que son partes integrales de la computadora, no son considerados como periféricos.

Cada dispositivo, sea periférico o no, requiere de un programa llamado controlador de dispositivo (device driver) que actúa como un traductor, convirtiendo comandos generales de una aplicación en comandos que entiende el dispositivo.

Dispositivos físicos

Espacio de coordenadas final para la mayoría de las operaciones de dibujo GDI. También, los dispositivos físicos son todo el hardware de la computadora.

DUMPBIN

Descargador de archivos binarios COFF de Microsoft (DUMPBIN.EXE). DUMPBIN despliega información sobre los archivos binarios de Formato de Archivos de Objetos Comunes (COFF) de 32 bits. DUMPBIN puede usarse para examinar los archivos objeto COFF, bibliotecas comunes de objetos COFF, archivos ejecutables, y Biblioteca de Enlace Dinámico (DLLs). DUMPBIN es una herramienta de 32 bits que corre solamente desde la línea de comandos.

E

Editor de recursos

Programa especializado que se usa para agregar, editar y borrar recursos de un programa.

En tiempo de compilación (Compile Time)

Punto en el cual un programa se está compilando, o la cantidad de tiempo requerido para compilar un programa.

En tiempo de carga (Load time)

Periodo de tiempo requerido para colocar los archivos ejecutables de un programa en la memoria antes de su ejecución, o el punto en el tiempo cuando los archivos se cargan.

En tiempo de ejecución (Run time)

Punto en el tiempo donde un programa se está ejecutando, o el periodo de tiempo necesario para ejecutar un programa.

Enlace dinámico

Enlace de un programa, a una biblioteca o a un objeto, que se establece cuando el programa se carga en memoria (enlace dinámico durante la carga, load-time dynamic linking) o durante su ejecución (enlace dinámico en tiempo de ejecución, run time dynamic linking).

Enlace incremental

Proceso de reconstruir solamente los archivos de programas que tienen una fecha (tomando en cuenta también la hora) más reciente que el último enlace, o actualizar solamente las partes del programa que han cambiado. Un programa enlazado incrementalmente tiene la misma funcionalidad que un programa sin enlace incremental, pero difiere en que es preparado para enlaces incrementales subsecuentes y como resultado es más grande.

Enlazador (Linker)

Herramienta que combina archivos objeto y bibliotecas para crear un archivo ejecutable o una biblioteca de enlace dinámico. Durante el proceso de enlace, el enlazador resuelve las referencias externas, como las llamadas a una rutina en una biblioteca, de un programa.

Enlazar

Combinar archivos objeto y bibliotecas para formar un archivo ejecutable o una biblioteca de enlace dinámico.

Error en tiempo de ejecución

Error matemático o lógico en un programa que se vuelve aparente solamente durante la ejecución. Tal error puede producir resultados incorrectos o causar que el programa se comporte impredeciblemente o estalle.

Espacio de trabajo de la aplicación (APPLICATION FRAMEWORK)

O Estructura. Grupo de clases C++ en la biblioteca MFC que proporcionan los componentes esenciales de una aplicación Windows. El espacio de trabajo de la aplicación define el esqueleto, o estructura de una aplicación y proporciona las implementaciones de la interfaz del usuario común que puede ser colocada en el esqueleto.

Espacio de dirección

O espacio de memoria. La porción de memoria asignada a un proceso dado. Ver Heap.

Espacio de memoria. Más general, el rango de sitios de memoria a los cuales un procesador puede referirse. El espacio de direcciones de memoria de una computadora es la cantidad de memoria que un procesador puede usar si toda la memoria estuviera disponible.

Espacio de dirección segmentado (Segmented Address Space)

Espacio de direcciones dividido lógicamente en secciones llamadas segmentos.

Direccionamiento de memoria en el que cada byte se ve referenciado por un segmento, o base, número y un desplazamiento que se añade a él. Ver la diferencia con Espacio de dirección plano y modelo plano.

Espacio plano de direcciones (Flat Address Space)

Direccionamiento de memoria en donde cada byte tiene como referencia un número secuencial diferente comenzando por el 0. Ver la diferencia con Espacio de dirección segmentado y modelo plano.

Etiqueta

En el cuerpo de un programa en C o C++ , es un nombre único seguido de dos puntos (:). Las etiquetas denotan enunciados a los cuales el enunciado *goto* puede ramificarse. En un enunciado *switch*, las etiquetas precedidas por la palabra reservada *case* listan valores para ser comparados con la expresión al inicio del enunciado *switch*.

En el diseño de una interfaz de usuario, es un texto definido por la aplicación asociado con un elemento gráfico como un botón o una caja de selección.

F

Fase de traducción

Uno de los pasos que sigue el compilador al crear un archivo ejecutable. En C/C++, estos pasos incluyen mapeo de caracteres, contracción de líneas, preprocesamiento, mapeo basado en caracteres, concatenación de cadenas, traducción, y enlace.

Función de llamado recurrente (Callback Function)

Función de llamado recurrente. Son funciones en una aplicación que llaman al sistema operativo en respuesta a un requerimiento hecho por una aplicación.

Típicamente, las funciones que se definen en una aplicación no necesitan un trato especial. Existen dos excepciones:

- Funciones en una Biblioteca de Enlace Dinámico (DLL) que se llaman fuera de la biblioteca.
- Funciones de llamado recurrente (también llamados procedimientos de llamado recurrente).

Las funciones de llamado recurrente son funciones que son llamadas por Windows, no por la aplicación. Los siguientes son tipos comunes de estas funciones:

- WinMain. Este es el punto de entrada de la aplicación.

Procedimientos de Ventana de la aplicación. Estos procedimientos procesan mensajes enviados a la ventana.

- Procedimientos de cajas de diálogo de la aplicación. Estos procedimientos procesan mensajes enviados a la caja de diálogo.
- Funciones de llamado recurrente de enumeración. Estas funciones manejan los resultados de funciones de enumeración de Windows.

- Funciones de notificación de memoria. Estas funciones son llamadas por Windows para notificar a la aplicación que un objeto de memoria va a ser desechado.
- Funciones Window-hook (filtros). Estas funciones procesan mensajes enviados a la ventana de otras aplicaciones. La mayoría de las funciones window-hook deben estar en una biblioteca.

Función exportada

Función llamada por otras entidades de ejecución, tal como los DLLs o los archivos ejecutables. Generalmente, un DLL necesita exportar funciones para permitir que sus clientes lo controlen. Las funciones pueden ser exportadas usando un archivo de definición de módulo (.DEF) o el modificador de almacenamiento de clase `__declspec(dllexport)`, una extensión del lenguaje C específica de Microsoft. Este modificador asegura que otras aplicaciones y bibliotecas de enlace dinámico serán capaces de obtener, dinámicamente en tiempo de ejecución, la dirección de la variable o la función a la cual aplica.

H

Handle

En la programación de Windows es un apuntador, un identificador o un indicador. Es un símbolo, típicamente un apuntador que permite al programa acceder un recurso, como una función de una biblioteca.

Handles y objetos. Un objeto es una estructura interna que representa un recurso del sistema como un archivo, un hilo (thread) o una imagen gráfica. Una aplicación no puede acceder directamente la estructura interna de un objeto o el recurso del sistema que un objeto representa. En vez de eso, una aplicación debe obtener un apuntador (handle) al objeto y usar este apuntador (handle) para examinar o modificar el recurso del sistema (o ambos). En el API de Win32 de Microsoft, los apuntadores (handles) están implementados generalmente como apuntadores indirectos, pero este no es siempre el caso.

La mayoría de los apuntadores son índices a tablas internas. Windows usa índices de apuntadores para acceder información guardada en estas tablas. Típicamente, una aplicación tiene acceso solamente al apuntador y no a la información. Cuando la aplicación tenga que examinar o cambiar la información, proporciona el apuntador y Windows hace el resto. Esta es una manera en que Windows protege la información con sus capacidades de multitarea.

Dos de los parámetros de la función `WinMain` (`hInstPrevious` y `hInstCurrent`) son llamados apuntadores. Un apuntador es un entero único que usa Windows para identificar un objeto creado

o usado por una aplicación. Windows usa una amplia variedad de apuntadores para identificar objetos como instancias de una aplicación, ventanas, menús, controles, memoria asignada, dispositivos de salida, archivos, y plumas y brochas de la Interfaz de Dispositivos Gráficos (GDI).

Mango. En muchas aplicaciones, cuando se selecciona un objeto gráfico, un contorno del objeto aparece con cajas pequeñas. Cada caja es un mango. Arrastrando los mangos, se puede cambiar la forma y el tamaño del objeto. Los mangos proporcionan acceso a las operaciones de un objeto, pero pueden indicar la selección para algunos tipos de objetos.

Heap (Montón)

O almacén libre. Porción de memoria reservada por un programa para usarse como almacenamiento temporal de estructuras de datos cuya existencia o tamaño no puede ser determinado hasta que el programa está corriendo. El programa puede solicitar memoria libre del montón para mantener tales elementos, usarlos como sea necesario, y luego liberar la memoria.

Hilo

Entidad básica en la cual el sistema operativo asigna tiempo del CPU. Un hilo puede ejecutar cualquier parte del código de la aplicación, incluyendo una parte que está siendo ejecutada en el momento por otro hilo. Todos los hilos de un proceso comparten el mismo espacio de direcciones virtual, variables globales, y recursos (del sistema operativo) del proceso.

I

Identificador de recurso (Resource ID)

En Windows y en MacOS, un número que identifica un recurso en particular dentro de un tipo de recursos —por ejemplo, un menú entre otros muchos recursos de tipo MENU que usa un programa. El tipo de recurso, junto con el identificador (ID) del recurso, define únicamente cada recurso en una aplicación. Es posible también, que recursos de diferentes tipos tengan el mismo identificador de recurso.

Independiente del dispositivo

Característica del software y archivos que generan la misma salida sin importar el hardware involucrado. Un programa independiente de un dispositivo podría, por ejemplo, emitir el mismo comando para dibujar un rectángulo sin importar si el dispositivo de salida es una impresora, un graficador o un monitor.

Instancia

En Windows se puede correr más de una aplicación al mismo tiempo. Una instancia es ejecutar más de una copia, o instancia, de la misma aplicación al mismo tiempo. Windows puede distinguir una instancia de otra usando un apuntador único a una instancia (un entero que identifica la instancia) cada vez que la aplicación llama a la función WinMain para iniciar la aplicación.

En algunos sistemas multitarea, para ejecutar múltiples instancias de una misma aplicación, el sistema carga una copia del código y los datos de la aplicación a memoria y ejecuta la instancia. En Windows, cuando se inicia una instancia nueva de una aplicación, sólo se inician los datos para la aplicación que se cargó. Windows usa el mismo código para todas las instancias de esa misma aplicación. La ventaja es que se ahorra espacio para otras aplicaciones al no cargarse el mismo código para cada instancia iniciada. Sin embargo, este método requiere que los segmentos de código de la aplicación permanezcan sin cambios, es decir, que no se guarden datos, y que el código de la aplicación no cambie mientras se ejecuta la aplicación.

Generalmente, en Windows la primera instancia tiene un papel especial. Muchos de los recursos que crea una aplicación, como clases de ventanas, están generalmente disponibles para todas las aplicaciones. Por consiguiente, solamente la primera instancia de una aplicación crea estos recursos. Todas las subsecuentes pueden usarlos sin tener que crearlos.

Instanciación

Acto de crear un objeto o tipo de datos, generalmente una clase.

L

Línea de comandos

Cadena de texto escrita en el indicador de comandos, o ejecutada de un archivo de comandos, que especifica una tarea o tareas a ejecutarse por el sistema operativo o una aplicación.

Link

Enlazador incremental de 32 bits de Microsoft (LINK.EXE).

Llamada de una función

Expresión sufija seguida de un paréntesis posiblemente vacía, lista de expresiones separada por comas la cual constituye los argumentos de la función.

M

Macro

En C y C++, un nombre definido por la directiva *#define* que especifica el texto a reemplazar cuando se invoque la macro en el futuro en la unidad de traducción.

Mapa de bits

También imagen de píxeles o mapa de píxeles. Arreglo de bits que contiene datos que describen los colores encontrados en una región rectangular en la pantalla. Una colección secuencial de bits que representa, en memoria, una imagen desplegada en la pantalla o impresa.

MDI (Multiple document interface)

Interfaz de documentos múltiples. Arquitectura de interfaz del usuario común para las aplicaciones basadas en Windows. Una aplicación MDI permite al usuario trabajar con más de un documento dentro de la misma instancia de la aplicación. Cada documento se muestra dentro del área cliente de la ventana principal de la aplicación.

Memoria intermedia

Área de almacenamiento temporal, generalmente en la memoria RAM. Sirve como área de retención, permitiendo que el CPU manipule los datos antes de transferirlos al dispositivo. Debido a que el proceso de lectura y escritura de datos al disco es relativamente lento, muchos programas vigilan los cambios de datos en una memoria intermedia y luego copian la memoria intermedia al disco. Por ejemplo, los procesadores de palabras emplean una memoria intermedia para vigilar los cambios en el archivo actual. Luego cuando se guarda el archivo, el procesador de palabras actualiza el archivo en disco con el contenido de la memoria intermedia. Esto es más eficiente que acceder el archivo en el disco cada vez que se hacen cambios.

Memoria virtual

Esquema de administración de memoria que permite que una aplicación "vea" una parte de memoria (RAM) continua y grande que, en realidad, es una parte de memoria más pequeña suministrada por memoria secundaria (como un disco duro), con fragmentos de datos siendo movidos hacia y desde la memoria principal.

MFC Library – La biblioteca MFC

La biblioteca MFC es la biblioteca de clases para el desarrollo de MS Windows. Ver Biblioteca de Clases.

Módulo[ROP38]

Componente autónomo de hardware o software que interactúa con un sistema mayor. Los módulos de hardware se construyen a menudo para agregarse al sistema principal. Los de

software se diseñan para manejar una tarea específica dentro de un programa mayor. Pueden contener código y recursos de un archivo ejecutable. Estos elementos pueden ser compartidos porque son de sólo-lectura. Para cada módulo que sea único en el sistema, Windows crea una base de datos del módulo para guardar un registro de la información que está asociada con un módulo, incluyendo funciones exportadas, y ubicación de segmentos de código y recursos. Éste término ya no se usa para describir a los DLLs en Win32, sólo se usa para hacer referencia a un componente de un sistema mayor.

Modo real

El modo real es un estado operacional en el cual la computadora se limita a 1 MB de memoria.

Modo virtual 386

Es un modo operacional en procesadores 386 y superiores el cual permite al sistema operativo desempeñarse como múltiples CPUs 8086. Cada máquina virtual ejecuta una aplicación bajo su propia copia de DOS u otro sistema operativo, y todas las máquinas virtuales trabajan juntas en multitarea.

Modo protegido

Es un estado operacional que permite direccionar toda la memoria disponible de la computadora en máquinas con procesadores 386 o superiores. Este modo de ejecución evita que un programa se introduzca en los límites de la memoria de otro, permitiendo de este modo ejecutarse en un entorno "protegido".

Los modos virtual 8086 y protegido se refieren al modo de operación del procesador. Para la familia 8086 de Intel, existen dos modos de operación: el modo real y el modo protegido. El tercer modo, el virtual 8086, puede emular el modo real más de una vez en el sistema mediante rutinas del sistema operativo.

Modelo plano (Flat model)

Modelo plano. Este modelo coloca el espacio de direcciones completo del procesador dentro de un segmento de 32 bits. En teoría un programa puede acceder hasta 4 GB de memoria virtual (con un procesador Intel de 32 bits). Ver Modelo segmentado.

Modelo segmentado (Arquitectura segmentada)

La familia de procesadores de Intel anteriores al 80386 emplean una arquitectura en la que cada dirección se representa como un segmento y un desplazamiento (offset). La arquitectura

segmentada se diseñó originalmente para permitir a los procesadores de 16 bits acceder un espacio de direcciones mayor a 64K.

Moldeado (Casting)

Conversión implícita o explícita de un tipo de dato a otro.

Moldeo de tipos

Conversión explícita de una variable, estructura, objeto o expresión de un tipo de dato a otro.

Montón (heap)

Ver Heap.

N

Nombre decorado

O nombre de tipos seguro. En C++, una cadena generada por el compilador que contiene un nombre sin decorar (literal) seguido de una cadena de caracteres que usan el compilador y el enlazador para conservar información de los tipos. La decoración de nombres es necesaria para resolver ambigüedades que surgen en C++ de tener más de una función, por ejemplo, con el mismo nombre.

NULL

Enunciado que no ejecuta alguna acción, no declara algo, y puede usarse donde se espera un enunciado.

O

Objeto

Generalmente, es una instancia de una entidad que incorpora datos y funciones específicas que lo manipulan. En la programación orientada a objetos, un objeto es una entidad que tiene un estado, comportamiento e identidad. El estado de un objeto consiste de sus atributos y sus valores actuales. El comportamiento de un objeto consiste en las operaciones que pueden ejecutarse en él y los cambios de estado que lo acompañan. La identidad de un objeto es lo que se usa para distinguirlo de otros objetos.

Objeto gráfico

Una de las herramientas de dibujo, tal como una pluma, brocha, mapa de bits, paleta y demás, que Windows proporciona para usarse en un contexto del dispositivo.

P**Palabra clave (Keyword)**

O palabra reservada. Palabra que tiene un significado especial en a un programa o en un lenguaje de programación. Las palabras reservadas generalmente incluyen aquellas usadas para los enunciados de control, declaraciones de datos, y similares. Una palabra reservada puede ser usada sólo en ciertas circunstancias predefinidas, no puede usarse en nombres de funciones, variables, estructuras, etiquetas, clases, o herramientas definidas por el usuario como las macros.

Paleta

Arreglo de colores disponibles actualmente en un dispositivo.

Paleta de colores

Arreglo que contiene los valores RGB (Red, Green, Blue) que identifican los colores que pueden ser desplegados o dibujados en el momento en el dispositivo de salida. Las paletas de colores se usan en los dispositivos que son capaces de generar muchos colores pero que sólo pueden desplegar o dibujar un subconjunto de éstos en un momento determinado. Ver también Paleta de colores lógica.

Paleta de colores lógica

Arreglo de colores creados por una aplicación que puede seleccionarse en un contexto del dispositivo y usado para la salida de gráficos. Un color lógico es un color individual en el arreglo PALLETE. Una aplicación que usa una paleta de colores lógica puede pasar un valor COLOREF, en vez de un valor rojo, verde, azul (RGB) explícito a las funciones GDI que esperan un color.

Paleta del sistema

Representación de la paleta física del sistema. La paleta del sistema contiene los valores RGB para todos los colores que pueden ser desplegados o dibujados por el dispositivo.

Parámetro

Valor o expresión usada con un operador o pasada a un subprograma (subrutina, procedimiento, o función). El programa luego ejecuta operaciones usando los argumentos. Sinónimo de argumento.

Pila

Estructura de datos implementada como una lista UEPS (LIFO: last in, first out, ó el último en entrar es el primero en salir) para que el último elemento agregado a la estructura sea el primero en quitarse.

Región reservada en la memoria, organizada como una pila, en la cual los programas guardan temporalmente el estado de datos como las direcciones de regreso de llamadas a procedimientos y funciones, parámetros pasados y variables locales. Ver también Montón (heap).

Pila de llamados (call stack)

Lista ordenada de funciones que ha sido llamadas pero que no han regresado, con la función actual en ejecución listada primero. Cada llamada se muestra opcionalmente con los argumentos y tipos que se le pasaron. Durante una sesión de depuración, se pueden ver las funciones que se han llamado pero que no han regresado.

Pixel

Pix (picture) ELeмент. Elemento de imagen. El elemento más pequeño en una pantalla. Una pantalla se divide en miles de puntos pequeños, y un pixel es uno o más puntos que se tratan como una unidad. Un pixel puede ser un punto en una pantalla monocromática, tres puntos (rojo, verde y azul) en una pantalla a color, o una agrupación de esos puntos.

Pluma

En aplicaciones gráficas, un objeto de dibujo usado para dibujar líneas y bordes.

Preprocesador

Es un procesador de texto que manipula el texto de un archivo fuente como parte de la primera fase de traducción. El preprocesador no analiza el texto fuente, pero lo divide en símbolos con el propósito de localizar las llamadas a macros. Aunque el compilador invoca al preprocesador en su primera pasada, el preprocesador puede ser invocado separadamente para procesar texto sin compilar.

Proceso

Una vez que un programa está corriendo, es un proceso. Un proceso posee su propia memoria, apuntadores a archivos (file handle), y otros recursos del sistema. Si se ejecuta un mismo programa dos veces seguidas, se tienen dos procesos. En Windows 95, un proceso tiene su propio espacio privado de direcciones de memoria virtual de 4 GB, código, datos y otros recursos del sistema operativo, como archivos, y otros. Un programa puede acceder cada byte de este

espacio con una sola dirección lineal de 32 bits. El espacio de memoria de cada proceso contiene una variedad de elementos, incluyendo:

- La imagen del programa EXE
- DLLs no pertenecientes al sistema que cargue el programa, incluyendo DLLs MFC
- Datos globales del programa (de sólo lectura y de lectura/escritura)
- La pila del programa
- Memoria asignada dinámicamente, incluyendo la memoria (heap) de Windows y CRT
- Archivos mapeados a memoria
- Bloques de memoria compartidos entre procesos
- Hilos de ejecución de memoria local a específica
- Todas las clases de bloques de memoria del sistema especiales, incluyendo tablas de memoria virtual
- El núcleo y el ejecutivo (executive) de Windows, además de los DLLs que son parte de Windows

Punto de entrada

Dirección de inicio de una función, archivo ejecutable o biblioteca de enlace dinámico.

Punto de rompimiento (Breakpoint)

Sitio en un programa donde la ejecución se detiene para permitir al desarrollador examinar el código del programa, las variables, los valores de los registros, y si es necesario, hacer cambios, continuar o terminar la ejecución.

R

Región

En las aplicaciones Windows, una región es un rectángulo, polígono, elipse, o una combinación de dos o más de estas formas usadas para definir una parte de la ventana cliente a ser pintada, invertida, o llenada con salida. También, se refiere a un área dedicada a, o reservada para un propósito en particular.

Registro

En Windows de 32 bits, la base de datos en la cual la información de configuración se registra. Esta base de datos se encarga de la mayoría de los archivos de configuración e inicialización de Windows y de las aplicaciones basadas en Windows.

RGB

Red Green Blue. Modelo mixto, o método de describir colores, en medios basados en luz tal como los monitores. RGB mezcla porcentajes de los colores primarios basados en luz (rojo, verde y azul, en inglés red, green, blue) para crear otros colores. Windows define estos porcentajes como tres valores de 8 bits llamados valores RGB. Un valor de cero para todos los colores primarios, o un valor RGB de (0,0,0), produce negro y un valor máximo, 255, o un valor RGB de (255, 255, 255), produce el color blanco.

S

SDK

Software Development Kit. Conjunto de bibliotecas, archivos de encabezados (header files), libros, ayuda en línea y ejemplos de programas diseñados para ayudar a un desarrollador a crear software.

Secuencia de llamado

Determina el orden en el cual los argumentos se mueven a la pila durante la llamada de una función y que bloque de código es responsable para el apuntador a la pila. Generalmente, el compilador de C genera el código que mueve los parámetros de la pila de derecha a izquierda, empezando con el último parámetro.

Servicio

Funcionalidad obtenida de un programa determinado. En Win32, un objeto ejecutable que se instala en una base de datos del registro mantenida por el Administrador del Control de Servicios. El archivo ejecutable está asociado con un servicio puede iniciarse mientras se carga el sistema operativo o por el sistema, o puede iniciarse en demanda del Administrador del Control de Servicios. Hay dos tipos de servicios: servicio Win32 y servicio de dispositivo.

SDI

Single document interface. Interfaz de un solo documento. Arquitectura de la interfaz del usuario que permite al usuario trabajar con sólo un documento al mismo tiempo. El cuaderno de notas de Windows es un ejemplo de una aplicación SDI.

Símbolo

Cualquier otro carácter que los caracteres alfanuméricos comunes. Generalmente se refiere a caracteres algebraicos, científicos o lingüísticos no encontrados en el teclado.

En programación, un nombre que representa un registro, un valor absoluto, o una dirección de memoria (relativa o absoluta).

En un compilador, una variable, nombre de función u otro identificador.

En Visual C++, un identificador de recurso que consiste de una cadena de texto (nombre) mapeado a un valor entero. Un símbolo proporciona una manera de referirse a recursos y objetos de la interfaz del usuario, ambos en código fuente y en los editores de recursos.

T

Tecla aceleradora

O acelerador de teclado, atajo o atajo de teclado. Una tecla o combinación de teclas que invoca a un comando en particular.

Tabla aceleradora

O recurso acelerador. Estructura de datos que contiene una lista de teclas aceleradoras y los identificadores de comandos asociados con ellas.

Temporizador

Reloj interno. En Windows, es una rutina interna que mide repetidamente un intervalo especificado, en milisegundos. Cada vez que el intervalo (o valor de descanso) pasa, el sistema notifica a la ventana asociada con el reloj. Debido a que la precisión de un reloj depende de la velocidad del reloj del sistema y la frecuencia con la cual la aplicación recupera mensajes de la cola de mensajes, el valor de descanso es solamente aproximado.

Timer

Ver Temporizador.

Traducción

En programación, convertir un programa de un lenguaje a otro —por ejemplo, convertir código fuente en C a código objeto.

U

Unidad de traducción

Archivo fuente junto con todos sus archivos de inclusión, suministrados como entrada al compilador, el cual combina y traduce los archivos para producir un archivo objeto.

En C/C++, Una secuencia de signos que genera el compilador durante la fase del preprocesador. La unidad de traducción incluye código de las directivas del preprocesador, tal como las directivas `#include` y `#define`, en el código fuente de un solo archivo y excluye cualquier código removido por las directivas de compilación condicional.

V

Ventana hijo

Ventana que tiene el estilo `WS_CHILD` o `WS_CHILDWINDOWS` y está confinada al área cliente de su ventana padre, la cual inicia y define la ventana hijo. Generalmente, una aplicación usa ventanas hijo para dividir el área cliente de una ventana padre en áreas funcionales.

Ventana marco

En MFC, la ventana que coordina las interacciones de la aplicación con un documento y su vista. La ventana marco proporciona un marco visible alrededor de una vista, con una barra de estado y controles de ventana comunes opcionales tales como un menú de control, botones para minimizar y maximizar la ventana, y controles para cambiar el tamaño de la ventana. La ventana marco es responsable de manejar la salida de sus ventanas hijo y otros elementos del área cliente como las barras de control y las vistas. El marco de la ventana también reenvía comandos a sus vistas y puede responder a los mensajes de notificación de las ventanas de control.

Ventana marco principal

Ventana principal responsable de coordinar el marco con su vista. En una aplicación SDI, la ventana marco del documento es también la ventana marco principal. En una aplicación MDI, las ventanas del documento son ventanas hijo mostradas en la ventana marco principal. Los estilos y otras características de las ventanas marco se heredan de la clase de la ventana marco principal.

Vista

Objeto de una ventana a través del cual un usuario interactúa con un documento.

W

Win32s

Win32s es un subconjunto del API Win32. Windows 3.1 y Windows para Grupos de Trabajo usan Win32s para ejecutar aplicaciones hechas para el API Win32. Una aplicación Win32s es idéntica

a una aplicación de Windows NT o Windows 9x excepto que puede usar sólo un subconjunto específico del API Win32.

Windows NT y Windows 9x soportan el API Win32 mientras que Win32s soporta llamadas que pueden ser implementadas en MS-DOS y Windows 3.x. Por ejemplo, Win32s no soporta funciones Win32 de hilos, seguridad y E/S asíncrona de archivos.

BIBLIOGRAFIA

Advanced Windows: The developer's guide to the Win32 API for Windows NT and Windows 95
Microsoft Press
Jeffrey Richter
ISBN 1-55615-677-4

Inside Visual C++
David J. Kruglinski
Microsoft Press
ISBN: 1-55615-891-2

Microsoft Developer Network
Development Library
July 95 (MSDNL12)

Programming Windows 95
Charles Petzold
Microsoft Press
ISBN 1-55615-676-6

Visual C++
Ayuda en línea
Microsoft Corporation