Universidad Nacional Autónoma de Nicaragua
UNAN–León

Tesis de Maestría en Computación

Maestría impartida por la UAH-España en la UNAN-León

PLAN DOCENTE PARA LA ASIGNATURA

PROGRAMACIÓN VISUAL

TITULACION INGENIERIA EN SISTEMAS DE INFORMACION

Autor: Álvaro Rafael Altamirano Osorio

Director: Francisco Javier Ceballos Sierra

León, Julio de 2004

ÍNDICE GENERAL

PLAN DOCENTE DE LA ASIGNATURA: PROGRAMACIÓN VISUAL

Capítulo 1: ASPECTOS INTRODUCTORIOS

1.1 INTRODUCCIÓN

En el año 1994 se crea bajo el Proyecto de Colaboración con Nicaragua, un convenio entre el Departamento de Automática de la Universidad de Alcalá (UAH) y el Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua en León (UNAN-León) con el objetivo de apoyar la carrera de Licenciatura en Computación, ahora Ingeniería en Sistemas de Información, ofertada por la UNAN-León. El convenio está dirigido a impulsar la excelencia en la enseñanza que permitirá formar mejores profesionales en el área de la informática. El convenio ha permitido la especialización y actualización del personal docente y de estudiantes que se han incorporado al mismo. Por lo tanto, este proyecto pretende ser una contribución para alcanzar los objetivos que se pretenden alcanzar con el convenio.

En el presente documento se desarrolla un plan docente para la asignatura **"Programación Visual"** que será impartida en la carrera de Ingeniería en Sistemas de Información que es ofertada por el Departamento de Computación en la Facultad de Ciencias de la UNAN-León. Esta planificación incluye los aspectos didácticos relativos a su impartición.

Este documento presenta la situación actual de esta asignatura, su relación con otras asignaturas, la metodología y material didáctico para impartirla, la metodología de evaluación, así como la presentación del contenido teórico y práctico, el desarrollo y solución de las prácticas, la bibliografía y referencia necesaria para la misma.

El desarrollo del proyecto se plantea en dos partes. En la primera se hace un análisis de los aspectos relacionados con las características docentes y el método de impartición de la asignatura Programación Visual. En la segunda parte se incluyen los elementos, tanto teóricos como prácticos, que constituyen el material didáctico necesario para dicha asignatura.

1.2 OBJETIVOS

Los objetivos del presente proyecto docente son:

- Desarrollar un plan docente que abarque los conocimientos básicos en el área de la Programación Visual que un estudiante de Ingeniería en sistemas debería tener.

- Elaborar el contenido teórico y práctico de la asignatura que le servirá tanto al profesor encargado de impartir la asignatura como al estudiante que la recibe.

- Permitir una mejor planificación y organización de la asignatura.

- Contar con un documento guía para impartir la asignatura de Programación Visual.

- Introducir al alumno en el aprendizaje de la creación de aplicaciones gráficas para Windows.

1.3 DESARROLLO DEL PROYECTO

Para elaborar este proyecto docente, se han seguido las etapas siguientes:

- Como primer paso se establecen las asignaturas relacionadas con la Programación Visual.

- Como segundo paso se establecen: material y método didáctico, planificación temporal, metodología de evaluación y contenido del temario.

- Como tercer paso se detallan cada una de las clases de la asignatura Programación Visual como desarrollo del temario el cual presenta: clase de presentación con el documento para el Alumno, y donde cada tema contiene: introducción, objetivos, tabla de contenidos, duración (temporización) y, bibliografía.

- Como última etapa se desarrollan las Prácticas de Laboratorio y sus soluciones.

Para la realización del documento se ha contado como recurso primario con una computadora Pentium 2 conectada en red con sistema operativo Windows 2000 Profesional, en la cual se han llevado a cabo pruebas de las prácticas, utilizando el software necesario como es el Microsoft Visual C++ 6.0 que forma parte de Microsoft Visual Studio el cual podrá ser instalado en los laboratorios para que los estudiantes desarrollen sus prácticas.

Capítulo 2: PLAN DE ESTUDIOS Y METODOLOGÍA DOCENTE

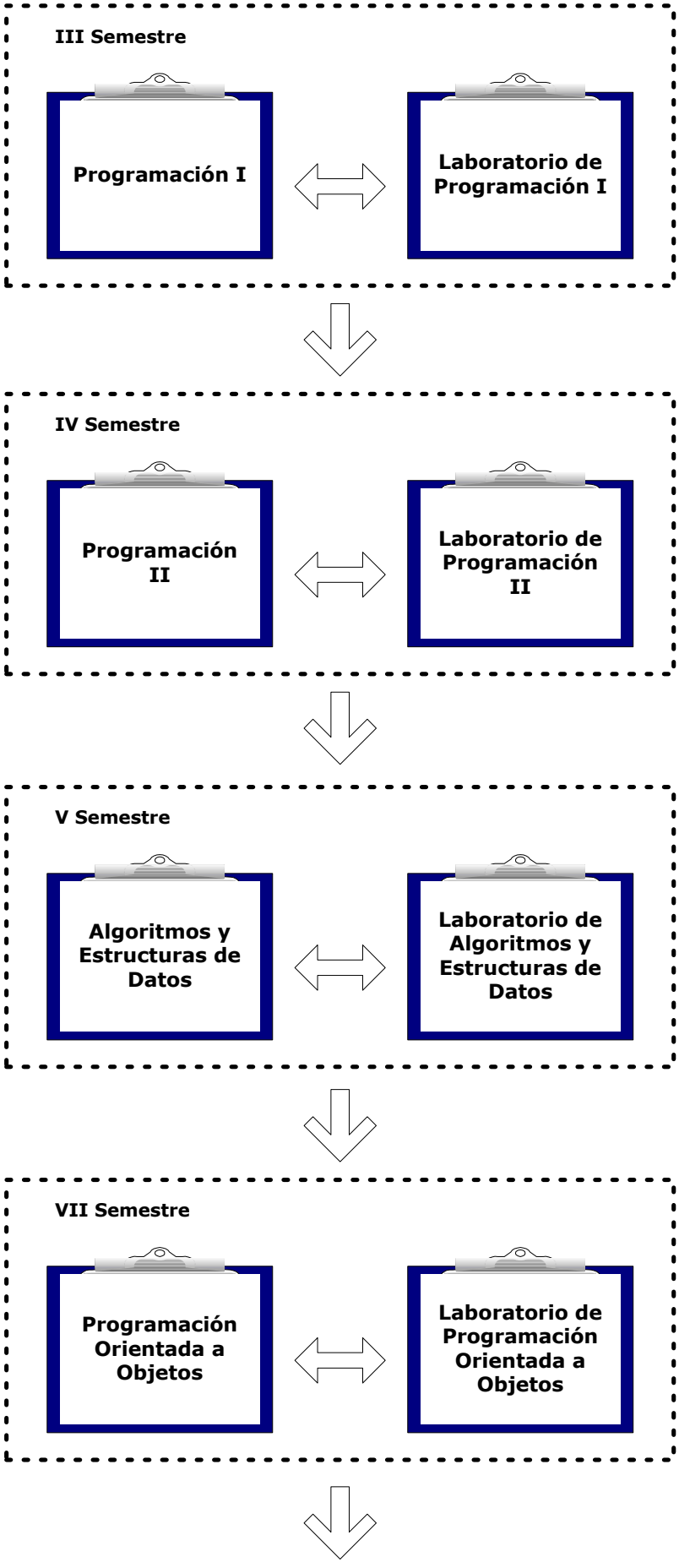
2.1 PROGRAMACIÓN VISUAL EN EL PLAN DE ESTUDIOS DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

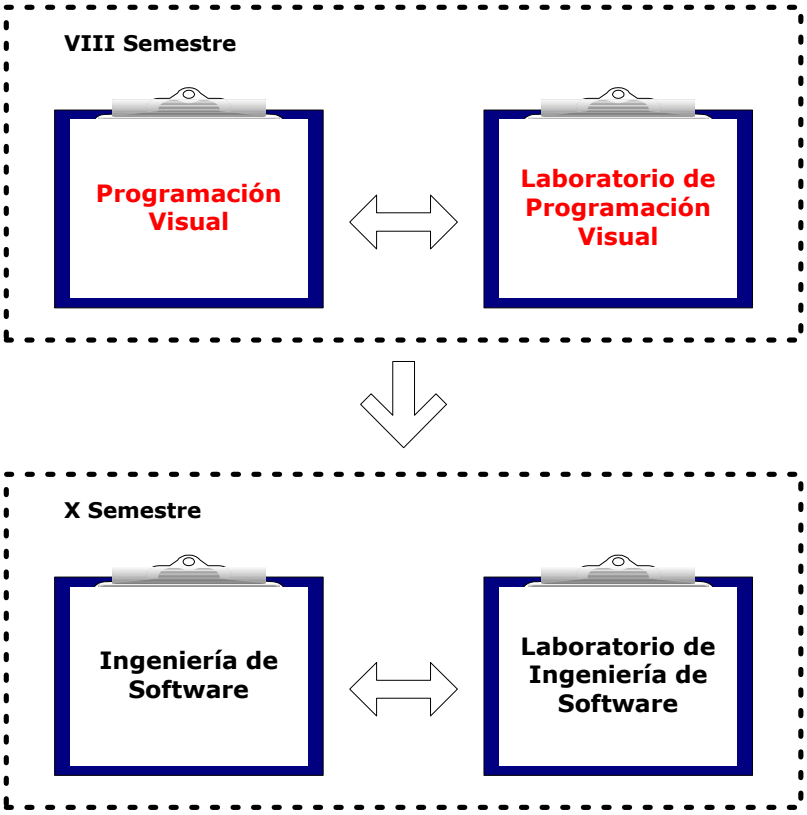
Antes de desarrollar el contenido teórico de la asignatura “Programación Visual” debemos hacer un análisis de la situación de dicha asignatura en el plan de estudios de la carrera de Ingeniería en Sistemas de Información (ISI) ofertada por el Departamento de Computación de la UNAN – LEÓN.

Dicho análisis nos permitirá saber los conocimientos previos que debe tener el alumno que lleva esta materia, así como también la relación que tiene con otras asignaturas; esto también nos ayudará a lograr los objetivos propuestos en la elaboración del plan docente ya que debe existir una planificación adecuada en el orden en que se imparte el contenido de una asignatura en cada uno de los semestres que conforman un año académico.

Para poder aprender los conceptos y la metodología en que se basa la programación visual es necesario que el estudiante esté familiarizado con algún lenguaje orientado a objetos; el lenguaje C++ es el más recomendable ya que Visual C++ (que es lenguaje que se abordará en este plan docente) está basado en las características y funcionalidad del lenguaje C++. C++ ya ha sido abordado en el plan docente de Programación Orientada a Objetos.

Por todo lo expuesto anteriormente, se mostrará primero de forma gráfica la relación de “Programación Visual” con el resto de asignaturas y luego se describirá de forma breve cada una de ellas.


III Semestre
Programación I
Laboratorio de Programación I
IV Semestre
Programación II
Laboratorio de Programación II
V Semestre
Algoritmos y Estructuras de Datos
Laboratorio de Algoritmos y Estructuras de Datos
VII Semestre
Programación Orientada a Objetos
Laboratorio de Programación Orientada a Objetos



2.1.1 Programación I

En esta asignatura se introduce al estudiante en el aprendizaje de las técnicas de programación estructurada y el lenguaje de programación C.

En el primer tema se hace una breve introducción a los fundamentos de los ordenadores, haciendo cierto énfasis en los sistemas operativos, programas y lenguajes.

En el segundo tema se abordan las fases en el desarrollo de un programa y los elementos básicos del lenguaje C: tipos fundamentales de datos, tipos derivados, nombres de tipos, constantes, identificadores, expresiones, etc.

En el tercer tema se aborda la estructura de un programa en C: ficheros de cabecera, declaraciones, expresiones, declaración y definición de funciones sentencias, salida con formato, etc.

El cuarto tema se enfoca en las sentencias de control del lenguaje C: sentencia if, switch, while, for, etc.

El quinto tema aborda los tipos estructurados de datos: arrays, cadenas de caracteres, funciones para manipular cadenas de caracteres, funciones para conversión de datos, estructuras, uniones, etc.

El sexto y último tema se aborda los punteros en C: creación de punteros operadores de indirección, punteros y arrays, asignación dinámica de memoria, etc.

Todos los conocimientos teóricos abordados en la teoría son llevados a la práctica en los laboratorios.

Esta asignatura es una base importante no sólo para Programación II que es la continuación de ésta, sino también es de vital importancia para otras como son Algoritmos y Estructuras de Datos, Circuitos Lógicos, Sistemas Operativos, Programación Orientada a Objetos, Diseño de Compiladores, Programación Visual, Inteligencia Artificial y Sistemas Expertos.

2.1.2 Programación II

Esta asignatura es la continuación de Programación I, y aborda elementos más avanzados de la programación en C, para lo cual se necesita tener los conocimientos bases adquiridos en la primera parte.

En el primer tema se desarrolla la capacidad en el estudiante para que sea capaz de utilizar funciones en C, así como de crear sus propias funciones para su implementación en forma estructurada.

El segundo tema aborda las funciones estándar de E/S que proporciona los conocimientos necesarios para que el estudiante sea capaz de utilizar y crear archivos en lenguaje C, y diferenciar entre los diferentes tipos de archivos que se pueden implementar en este lenguaje.

El tercer tema se enfoca en el preprocesador de C: el uso de macros predefinidas y directrices en C en el proceso de precompilación.

Por último el cuarto tema se centra en el diseño y utilización de estructuras dinámicas de datos y los algoritmos básicos de ordenación.

Al igual que Programación I, todos los elementos teóricos son llevados a la práctica en los laboratorios.

Esta asignatura refuerza los conocimientos adquiridos en la primera parte, y sirve de base para comprender las asignaturas de Algoritmos y Estructuras de Datos y Circuitos Lógicos, y es imprescindible para desarrollar las prácticas de la misma. Al igual que la primera parte, también aporta elementos necesarios para la comprensión de la teoría y desarrollo de algunas prácticas de las asignaturas de Programación Orientada a Objetos, Diseño de Compiladores, Sistemas Operativos, Programación Visual, Inteligencia Artificial y Sistemas Expertos, y para Arquitectura de Computadores I.

2.1.3 Algoritmos y Estructuras de Datos

Para esta asignatura es necesario que el alumno tenga una base firme en programación para que pueda comprender más fácilmente la teoría en su totalidad, y pueda desarrollar las prácticas de laboratorio.

En esta asignatura el estudiante aprende a aplicar las técnicas de análisis de los algoritmos, realizar seudocódigos e interpretarlos, evaluar la eficiencia de los algoritmos y a desarrollar algoritmos de búsqueda usando diferentes técnicas.

El primer tema hace una introducción a la algorítmica. Se aborda la representación de los algoritmos, la eficiencia de los algoritmos, el análisis de casos peor y medio entre otras cosas.

El segundo tema aborda las estructuras dinámicas de datos: listas lineales, las operaciones básicas de inserción, borrado de elementos, recorrido de la lista, búsqueda dentro de la lista y el uso de las pilas.

El tercer tema cubre los algoritmos clásicos de recursividad, listas circulares, listas doblemente enlazadas, árboles, recorrido de árboles, árboles de búsqueda, borrado de árboles y árboles perfectamente equilibrados.

El cuarto tema ahonda en la clasificación de datos abordando el método de la burbuja, inserción, búsqueda de datos, algoritmo de Hash, entre otras cosas.

En esta asignatura también se aplican los conocimientos teóricos en la práctica a través de los laboratorios para los cuales es estrictamente necesario tener conocimientos de programación.

Ésta y las asignaturas de programación anteriormente descritas forman una base muy importante e imprescindible para la asignatura de Programación Orientada a Objetos en la cual se necesita tener conocimientos de algorítmica y programación para su mayor comprensión y para el desarrollo de las prácticas correspondientes a la programación en el lenguaje C++. De igual forma, Programación Orientada a Objetos es necesaria para el aprendizaje de Programación Visual.

2.1.4 Programación Orientada a Objetos

Esta asignatura sirve para introducir al estudiante en los conceptos de la programación orientada a objetos ya que la mayoría de los lenguajes de programación actuales se basan en esta metodología de programación y su conocimiento es de vital importancia para el estudio de las otras asignaturas que tienen relación y que se verán en los próximos semestres, entre estas tenemos Programación Visual, Diseño de Compiladores e Ingeniería de Software.

En el primer tema se ve una introducción a la POO, se explican los conceptos en los que se fundamenta esta metodología de programación.

En el segundo tema se aborda el lenguaje C++ viendo algunos elementos propios del lenguaje y las mejoras con respecto a su antecesor: el lenguaje C.

El tercer tema es uno de los más importantes ya que se encarga de explicar uno de los principales temas: el concepto de CLASE.

El cuarto tema es una continuación del tercero, donde se ahonda aún más en el estudio de las clases, y se ven varios temas de carácter un poco más avanzado tales como miembros de una clase que son punteros, punteros a miembros de una clase, arrays de objetos, etc.

El quinto tema abarca el tema de OPERADORES SOBRECARGADOS. Se explica cómo sobrecargar los operadores más importantes dentro de un contexto específico ya sean unitarios o binarios.

El sexto tema aborda otra de las características más importantes de la POO: el concepto de HERENCIA. También se estudia el concepto de POLIMORFISMO.

En el séptimo tema se estudian los tipos genéricos para comprender la forma de cómo generar plantillas de clase que puedan ser aplicadas a cualquier tipo de datos.

En el octavo y último tema se explica el mecanismo de manejo de excepciones que utiliza C++ para controlar situaciones anómalas durante la ejecución de un programa.

2.1.5 Programación Visual

Está incluida en el perfil del presente proyecto docente. El estudiante de Programación Visual ya cuenta en este nivel con los conocimientos necesarios para estudiar esta asignatura, tanto la parte teórica como la realización de las prácticas de laboratorio.

La presente asignatura tiene como objetivo fundamental introducir al estudiante en la creación de aplicaciones gráficas para Windows utilizando Visual C++ y la metodología de programación orientada a objetos.

El contenido tanto teórico como práctico de esta asignatura será abordado ampliamente más adelante en este documento.

2.1.6 Ingeniería de Software

En esta asignatura el alumno aprende toda la teoría y práctica relacionada con la creación de un software siguiendo la metodología orientada a objetos; para la etapa de análisis del software se utiliza el Lenguaje Unificado de Modelado (UML) que se ha convertido en el estándar actual para la creación de software orientado a objetos. La relación estrecha que existe entre esta asignatura y Programación Visual es que al utilizar UML como lenguaje de modelado del sistema, tenemos que usar un lenguaje de programación orientado a objetos a la hora de hacer la codificación pudiendo usar para este fin el lenguaje Visual C++.

En el primer tema se hace una introducción a UML explicando algunos conceptos y definiciones.

El segundo tema se centra en la etapa de planeación, análisis y diseño del sistema. Específicamente se ven temas tales como la creación del documento de “Especificación de requisitos”, casos de uso del sistema, el modelo conceptual, contratos, etc.

En el tercer tema se comienza a estudiar el lenguaje de modelado UML propiamente dicho; se estudia la notación que se debe usar para representar clases y relaciones así como también la representación en diagramas de clases.

El cuarto tema es una continuación del tema anterior, profundizando en características avanzadas de las clases y relaciones.

En el quinto tema se comienzan a ver los aspectos dinámicos del sistema; para ello se ve en este tema el diagrama de actividades donde se explica cómo deben reflejarse en un diagrama las distintas tareas que debe llevar a cabo el sistema que se está modelando.

El sexto tema se centra en la creación de máquinas de estados y diagramas de estados como una alternativa más para reflejar los distintas tareas que puede ejecutar un sistema en un momento determinado.

En el séptimo y último tema se estudia el modelado arquitectónico de un sistema para ver cómo representar en un diagrama los distintos componentes lógicos y físicos del sistema que se esté desarrollando.

2.2 METODOLOGÍA DOCENTE

2.2.1 Material y Método Didáctico

La metodología a utilizar para impartir esta asignatura consiste en lecciones magistrales con una duración de dos horas dos veces a la semana. Se desarrollará a lo largo del semestre todo el contenido teórico de la asignatura que se tenga planificado según la temporización que se detallará más adelante.

Para lograr un adecuado y completo aprendizaje de esta asignatura, el profesor propondrá una serie de prácticas de laboratorio que deben ser resueltas por los estudiantes en tiempo y forma. Dichas prácticas serán realizadas haciendo uso del lenguaje de programación Visual C++ y como herramienta de desarrollo se usará Microsoft Visual C++ 6.0 que forma parte de Microsoft Visual Studio 6.0.

La asignatura contará con una página web donde el profesor pondrá a disposición del estudiante toda la información, clases teóricas, prácticas de laboratorio, apuntes y material de apoyo sobre la misma a fin de que el estudiante consulte esta página cuando lo requiera.

El estudiante también podrá complementar las clases y el autoestudio con tutorías en las que podrá hacer consultas al profesor fuera del aula en un horario determinado brindado por el profesor al inicio del curso. También tendrá acceso en forma impresa a todo el material disponible en la Web de la asignatura así como a la bibliografía básica y complementaria de la misma, que estarán disponibles en el Departamento para su debido uso.

En cuanto al material didáctico para impartir las clases se hará uso de pizarra acrílica, marcadores y retroproyector. También es posible el uso de un DataShow cuando el profesor lo estime conveniente.

2.2.2 Metodología de Evaluación

La asignatura de Programación Visual está compuesta por las clases teóricas y por la parte práctica, que corresponde al laboratorio. A lo largo del curso se realizan dos evaluaciones parciales y por último un examen semestral.

Las notas obtenidas en las dos evaluaciones parciales corresponden al 60% de la nota final de la asignatura, que sumado al 40% de la nota semestral, se obtiene el 100% de la nota final de todo el curso. En cuanto a la obtención de cada una de las notas parciales, el desglose es el siguiente:

- ✓ Realización de un examen teórico..... 80 puntos
- ✓ Evaluación del laboratorio..... 20 puntos

El examen semestral tiene un valor de 100 puntos.

El cuadro que se presenta a continuación aclara lo antes expuesto:

Evaluación	Examen Teórico	Evaluación Práctica	Porcentaje Nota Final
I Parcial	80 puntos	20 puntos	30%
II Parcial	80 puntos	20 puntos	30%
Examen semestral	100 puntos	--	40%
TOTAL			100 %

La escala de evaluación es de 0 a 100 puntos siendo 60 la nota mínima para aprobar. Los exámenes escritos consistirán en preguntas teóricas directas que permitan evaluar el grado de comprensión de los conceptos básicos que tiene el estudiante. También contendrá preguntas de carácter analítico, basadas en algún problema planteado en las cuales el alumno deberá realizar un análisis a partir de los conocimientos teóricos adquiridos. Por último, también es posible que se hagan preguntas prácticas para evaluar el nivel alcanzado en la resolución de las prácticas del laboratorio.

Los estudiantes deberán realizar todas las prácticas establecidas en tiempo y forma para poder tener derecho a los 20 puntos correspondientes al laboratorio. El valor de cada práctica será proporcional entre el número de prácticas realizadas en cada evaluación parcial. El estudiante deberá mostrar cada práctica funcionando, responder a las preguntas hechas por el profesor y entregar una memoria que contendrá lo siguiente:

- ✓ Explicación de la solución
- ✓ Código comentado
- ✓ Posibles mejoras añadidas
- ✓ Conclusiones y comentarios

Otra forma de evaluar el laboratorio que puede adoptar el profesor es la siguiente: proponer un problema práctico cuya resolución por parte de los estudiantes puede tomar un tiempo considerable, por ejemplo: una hora. Al finalizar dicho lapso de tiempo, el profesor pasará evaluando uno a uno el trabajo realizado por cada estudiante. Por supuesto obtendrá la máxima calificación aquel alumno que resuelva de manera exacta y con todos los requerimientos el ejercicio propuesto.

Capítulo 3: PROGRAMACIÓN VISUAL (PV)

3.1 INTRODUCCIÓN

La asignatura de Programación Visual, que es de carácter obligatorio, será impartida en el VIII semestre de la carrera de Ingeniería en Sistemas de Información (ISI). Cuenta con 2 horas a la semana para la clase teórica y 2 horas semanales para el laboratorio.

Para la mejor asimilación y comprensión de esta asignatura se recomienda que el alumno tenga conocimientos sobre programación orientada a objetos; así como también conocimientos en el área de algoritmos y estructuras de datos; todos ellos fueron abordados en semestres anteriores en las asignaturas de Programación I, Programación II, Algoritmos y estructuras de datos y Programación Orientada a Objetos.

Con esta asignatura se pretende introducir al alumno en la creación de aplicaciones gráficas para Windows usando Visual C++ y la metodología de programación orientada a objetos. Para lograr esto en la parte teórica se ven los conceptos y numerosos ejemplos para explicarlos; y en el laboratorio se realizan una serie de prácticas para afianzar los conocimientos teóricos programando en el lenguaje Visual C++.

En el primer tema se hace una introducción viendo algunos conceptos principales, como por ejemplo cómo se hace la comunicación de una aplicación con Windows, bucle de mensajes, arquitectura de una aplicación, biblioteca de clases MFC, etc. Además se presenta el entorno de desarrollo que se utilizará para generar las aplicaciones.

3.1.1 Objetivos

Los objetivos de esta asignatura son:

- ✓ Introducir al estudiante en la generación de aplicaciones gráficas para Windows usando la metodología de programación orientada a objetos.
- ✓ Enseñar el lenguaje de programación orientado a objetos Visual C++ para lograr cumplir el objetivo anterior.
- ✓ Motivar a los estudiantes a que resuelvan problemas de la vida cotidiana aplicando este estilo de programación.
- ✓ Sentar las bases necesarias para continuar con el estudio de otras asignaturas que tienen estrecha relación con Programación Visual.

3.1.2 Planificación Temporal

Para realizar una adecuada planificación del contenido de esta asignatura a lo largo del semestre debemos analizar la duración del período lectivo y obtener el número de horas de que se dispone tanto para las clases teóricas como las prácticas de laboratorio.

Tomando en cuenta el calendario académico propuesto por la UNAN – LEON y que esta asignatura será impartida durante el segundo semestre del año lectivo, obtenemos que se cuenta con un total de 19 semanas, pero considerando los días festivos y otras incidencias que no permitan el normal desenvolvimiento de las actividades académicas, podemos deducir que contamos con 16 semanas netas para dar fiel cumplimiento con el contenido de la asignatura.

Para el desarrollo de la parte teórica se cuenta con dos períodos semanales de dos horas, por lo tanto se tienen un total de 64 horas en todo el semestre. Para el laboratorio se cuenta con una semana menos, la cual es dedicada para las siguientes actividades:

- ✓ Presentación del laboratorio.
- ✓ Organización de los estudiantes en grupos.
- ✓ Introducción al uso de la herramienta a utilizar (Microsoft Visual C++).

Esto significa que para la parte práctica se cuenta realmente con 15 laboratorios durante el semestre de dos horas cada uno para un total de 30 horas. El cuadro siguiente muestra la temporización para impartir la parte teórica:

UNIDAD	SEMANA	CONTENIDO	HORAS
-	1	Presentación de la asignatura	2
I	1-2	Introducción a la PV	6
II	3-4	Mi Primera aplicación	8
III	5-6-7	Cajas de diálogo, gráficos, tiempo y uso del ratón	12
IV	8-9-10	Variables de categoría, fuentes, color, cajas de diálogo estándar y menús	12
V	11-12-13	El teclado y la subclasificación	12
VI	14-15-16	Trabajando con menús	12
		TOTAL_HORAS	**64**

En cuanto al laboratorio, se tiene la siguiente planificación:

PRÁCTICA	SEMANA	CONTENIDO	HORAS
1, 2 y 3	1-5	Ventana principal, gráficos y diálogos	10
4 y 5	6-8	Fuentes y color, cajas de diálogo estándar	6
6	9-10	Subclasificación y diálogos	4
7	11-12	Menús dinámicos y flotantes	4
8 y 9	13-15	Otros controles. Seriación	6
		TOTAL_HORAS	**30**

3.1.3 Contenido del Temario Teórico

UNIDAD I: INTRODUCCIÓN A LA PV

1. ¿Qué se entiende por Programación Visual?
2. Entorno de desarrollo
3. Aplicación con ventanas
4. Comunicación con Windows
5. Bucle de mensajes
6. Biblioteca de clases MFC
7. Arquitectura de una aplicación
8. Análisis de una aplicación

UNIDAD II: MI PRIMERA APLICACIÓN: "HOLA MUNDO"

1. Diseño de una ventana principal
2. Iniciar la vista
3. Fecha y hora

UNIDAD III: CAJAS DE DIÁLOGO, GRÁFICOS, TIEMPO Y USO DEL RATÓN

1. Cajas de diálogo
2. Gráficos
3. Temporizador
4. Uso del ratón

UNIDAD IV: VARIABLES DE CATEGORÍA, FUENTES, COLOR, CAJAS DE DIÁLOGO ESTÁNDAR Y MENÚS

1. Variables de categoría "Control"
2. Fuentes
3. Color
4. Cajas de diálogo estándar
5. Menús

UNIDAD V: EL TECLADO Y LA SUBCLASIFICACIÓN

1. Subclasificación
2. Interceptar la tecla pulsada
3. Controles
4. Más sobre diálogos modales

UNIDAD VI: TRABAJANDO CON MENÚS

1. Menús
2. Clase CMenu
3. Operaciones con menús
4. Menús dinámicos
5. Aceleradores
6. Menús flotantes

3.2 PRESENTACIÓN DE LA ASIGNATURA

UNIDAD N° 0

OBJETIVOS:

- Dar a conocer al estudiante los aspectos relacionados a la docencia, horario de tutorías.
- Conocer los objetivos generales de la asignatura, el temario y su distribución temporal, así como las relaciones de la misma con la carrera profesional de la titulación.
- Conocer y comprender el método de enseñanza que se va a seguir durante el curso, los criterios de evaluación y la bibliografía necesaria.
- Conocer y comprender el modo de acceso a la documentación preparada por el profesor para la impartición de la asignatura.

CONTENIDO

- Documento de presentación de la asignatura al alumno. Ver documento en la siguiente página.
- Información sobre la organización de los grupos del laboratorio.

Todos los alumnos deben formar parte de un grupo para trabajar en el laboratorio, en total son 2 grupos, los días y horas de cada grupo se establecerán según el Departamento de Computación lo decida y en ese momento se usará el documento de presentación del laboratorio en donde se le brindará todos los datos necesarios para este, el cual estará disponible en la página web del profesor y en la secretaría del Departamento de Computación, se aclara que este documento estará presente en el proyecto en la sección de "Desarrollo de las Prácticas".

DURACIÓN: 1 HORA.

ASIGNATURA:	PROGRAMACIÓN VISUAL	CURSO 2004/2005
DEPARTAMENTO:	COMPUTACIÓN – DC	
TITULACIÓN:	INGENIERÍA EN SISTEMAS DE INFORMACIÓN	
AÑO:	IV	
PROFESOR:	ÁLVARO ALTAMIRANO OSORIO	

TEMARIO DE TEORÍA:

UNIDAD I: INTRODUCCIÓN A LA PV

1. ¿Qué se entiende por Programación Visual?
2. Entorno de desarrollo
3. Aplicación con ventanas
4. Comunicación con Windows
5. Bucle de mensajes
6. Biblioteca de clases MFC
7. Arquitectura de una aplicación
8. Análisis de una aplicación

UNIDAD II: MI PRIMERA APLICACIÓN: "HOLA MUNDO"

1. Diseño de una ventana principal
2. Iniciar la vista
3. Fecha y hora

UNIDAD III: CAJAS DE DIÁLOGO, GRÁFICOS, TIEMPO Y USO DEL RATÓN

1. Cajas de diálogo
2. Gráficos
3. Temporizador
4. Uso del ratón

UNIDAD IV: VARIABLES DE CATEGORÍA, FUENTES, COLOR, CAJAS DE DIÁLOGO ESTÁNDAR Y MENÚS

1. Variables de categoría "Control"
2. Fuentes
3. Color
4. Cajas de diálogo estándar
5. Menús

UNIDAD V: EL TECLADO Y LA SUBCLASIFICACIÓN

1. Subclasificación
2. Interceptar la tecla pulsada

3. Controles
4. Más sobre diálogos modales

UNIDAD VI: TRABAJANDO CON MENÚS

1. Menús
2. Clase CMenu
3. Operaciones con menús
4. Menús dinámicos
5. Aceleradores
6. Menús flotantes

MATERIAL DE ESTUDIO:

TRANSPARENCIAS Y MATERIAL DE APOYO:

- http://isi.unanleon.edu.ni/~alvaro/

BIBLIOGRAFÍA (TEORÍA):

- Ceballos Sierra, Francisco Javier. Visual C++, Aplicaciones para Win32, 2da. Edición. Editorial RAMA

TUTORÍAS:

- Despacho: Oficina Docentes Departamento de Computación
- Horario: Lunes 10:00 – 12:00
  Viernes 14:00 – 16:00
- Correo: alvaro@unanleon.edu.ni

EVALUACIÓN DE LA ASIGNATURA:

La escala de puntuación es de 0 a 100 puntos siendo 60 la nota mínima para aprobar la asignatura. El cálculo de la nota se hará de la siguiente manera:

80 puntos → Examen teórico-práctico
20 puntos → Evaluación del laboratorio

3.3 UNIDAD I: INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL

Título del tema:

Introducción a la Programación Visual

Objetivos:

- Conocer el entorno de desarrollo para generar las aplicaciones gráficas.
- Entender la forma de cómo se ejecuta una aplicación basada en el uso de mensajes.
- Conocer las clases que forman la arquitectura de una aplicación.

Contenido:

- Entorno de desarrollo
- Aplicación con ventanas
- Comunicación con Windows
- Bucle de mensajes
- Biblioteca de clases MFC
- Arquitectura de una aplicación

Duración:

- 2 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

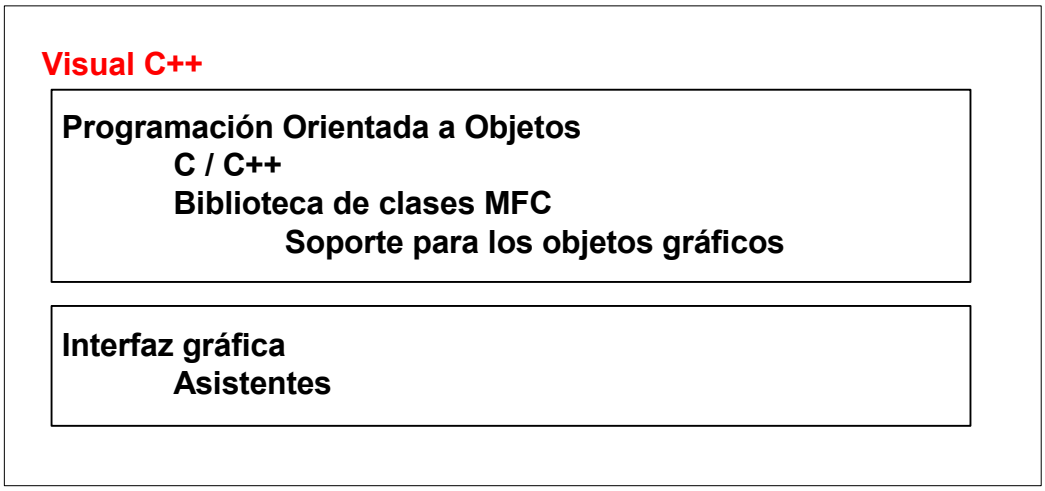
3.3 UNIDAD I: INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL

3.3.1 ¿Qué se entiende por Programación Visual?

La Programación Visual se encarga del estudio de la generación de aplicaciones gráficas para Windows basadas en el lenguaje C++ (Programación Orientada a Objetos) y la biblioteca de clases base de Microsoft MFC. Como herramienta de desarrollo se cuenta con Microsoft Visual C++ que forma parte de Microsoft Visual Studio el cual proporciona un entorno totalmente integrado (incluyendo asistentes) que permite crear ventanas con controles sin escribir nada de código, así como ejecutar todas las tareas de edición, ejecución, depuración y mantenimiento de programas de una forma fácil y cómoda.

3.3.2 Entorno de desarrollo

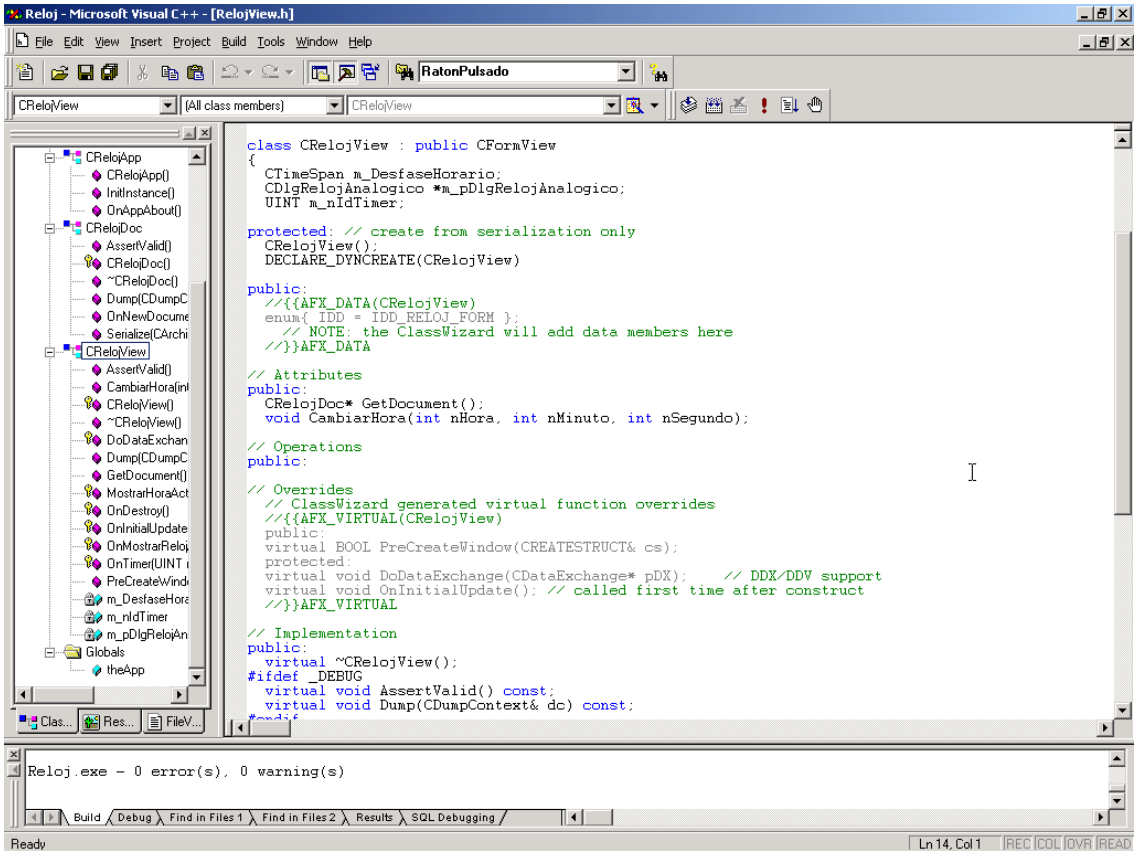
En la siguiente figura se muestra lo que se requiere para generar una aplicación gráfica para Windows haciendo uso de Visual C++:



Esta figura indica que en una aplicación gráfica intervienen por un lado, los conceptos de la programación orientada a objetos (el código se escribe en el lenguaje C++) y por otro lado tenemos la ayuda de los asistentes para facilitarnos la generación de la interfaz gráfica que se presentará al usuario. Ambas partes se conjugan y será lo que estudiaremos de aquí en adelante.

A continuación podremos ver nuestro entorno de trabajo en Visual C++. La ventana de trabajo está compuesta por:

- ✓ **Ventana del proyecto ó Workspace:** tiene tres pestañas; la vista de clases, la vista de recursos y la vista de ficheros que forman el proyecto.
- ✓ **Ventana de edición:** para visualizar y/o modificar el código.
- ✓ **Ventana de salida:** para visualizar los mensajes de salida a la hora de compilar, depurar y ejecutar el programa.



3.3.3 Aplicación con ventanas

Para crear una aplicación Windows debemos seguir los pasos siguientes:

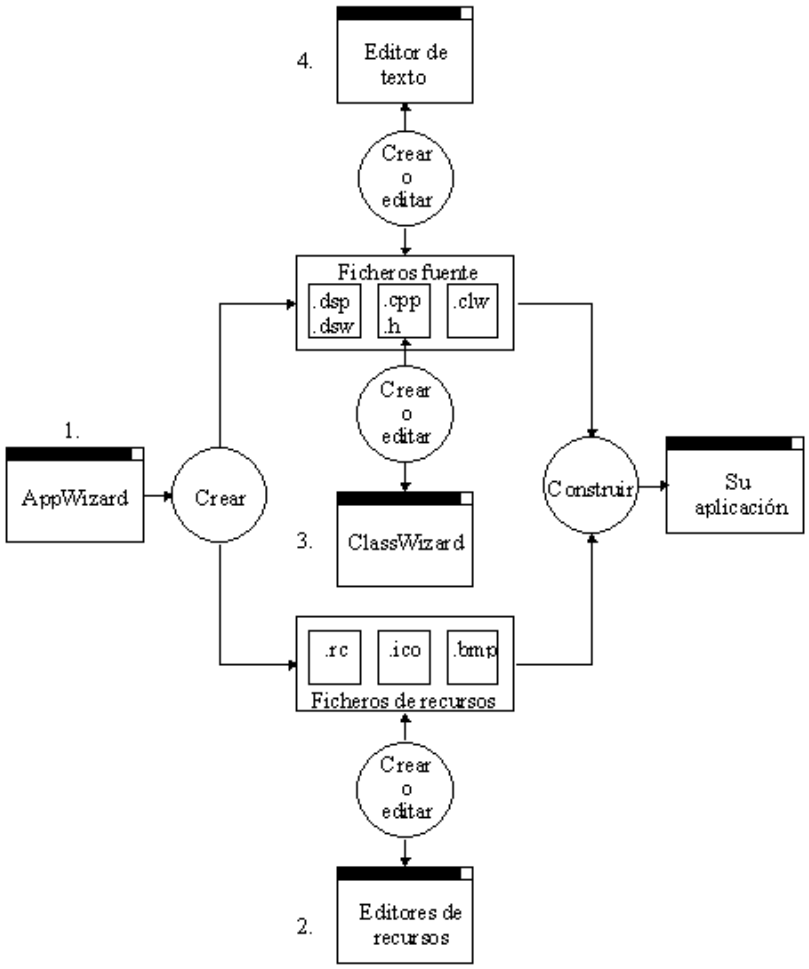
- ✓ Generar el esqueleto de la aplicación
- ✓ Desarrollar la aplicación

- Diseño de la interfaz de usuario
    - Edición de los ficheros de recursos
- Edición de los ficheros fuente
- Construir el fichero ejecutable

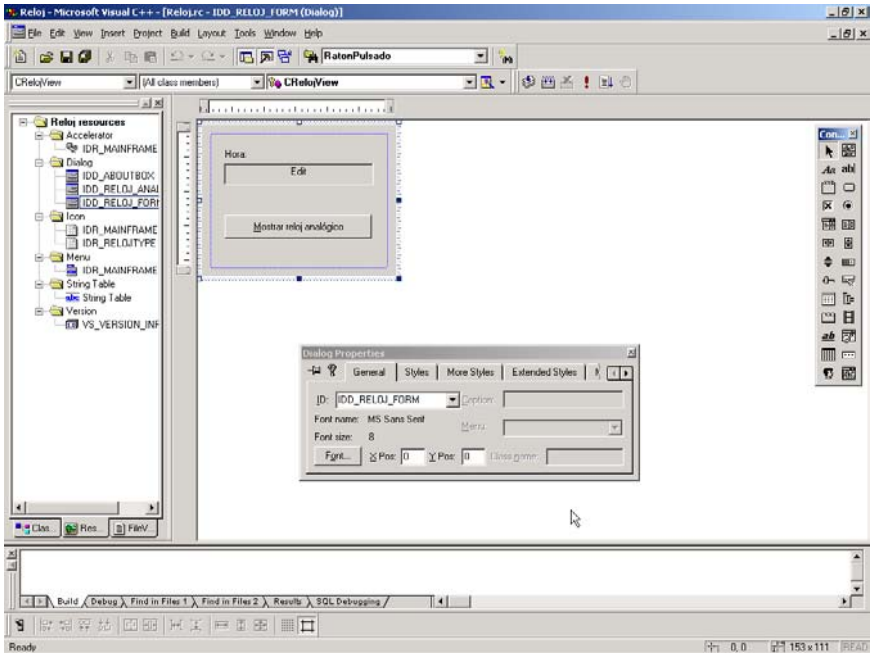
En “Generar el esqueleto de la aplicación” se generan un conjunto de ficheros de partida, como por ejemplo los ficheros .dsp, .dsw, .cpp, .h, .rc, .ico, .bmp, ... Para esto se usa el asistente *AppWizard* al iniciar un proyecto nuevo.

Luego se desarrolla la aplicación. Lo primero que debemos hacer es diseñar la interfaz gráfica de usuario (edición de los ficheros de recursos) para luego escribir el código correspondiente (edición de los ficheros fuente) y finalmente compilar y construir el fichero ejecutable de la aplicación.

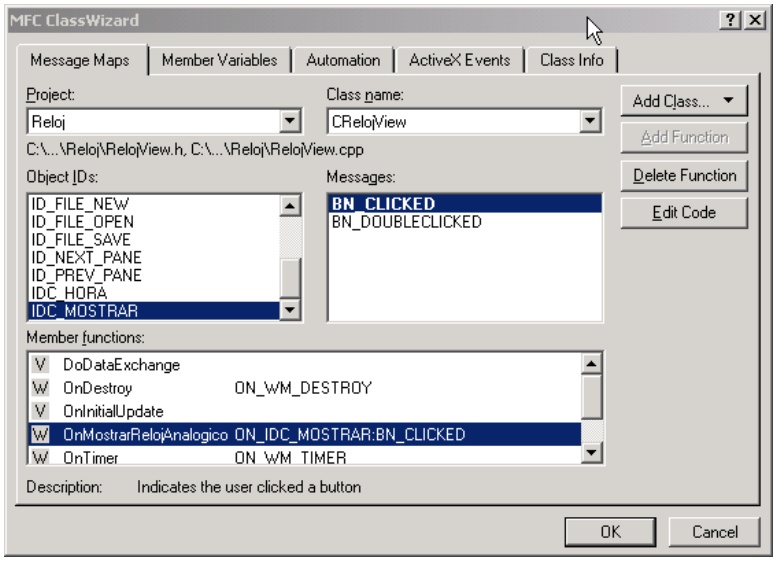
Estos pasos quedan resumidos en la siguiente figura:



A continuación podemos ver cómo editar el fichero de recursos, es decir, generar la interfaz gráfica:



En la siguiente figura podemos ver cómo añadir a través del asistente una nueva función para que responda a un evento (en este caso un clic del ratón):



3.3.4 Comunicación con Windows

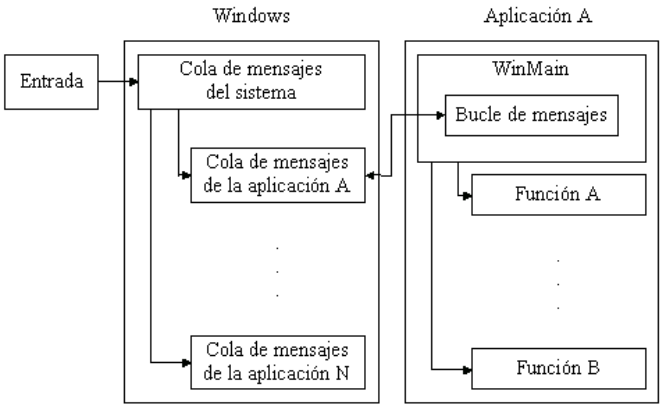
Cuando se ejecute el código de nuestra aplicación suceden los pasos siguientes:

- ✓ Se construye un objeto aplicación (theApp).
- ✓ Windows llama a WinMain de las MFC.
- ✓ WinMain llama a InitInstance de theApp.

- ✓ theApp crea el objeto ventana principal y los controles, y los visualiza.

Luego se debe responder a los mensajes de Windows y entonces sucede lo siguiente:

- ✓ theApp llama a su función miembro Run y ejecuta el bucle de mensajes.
- ✓ La aplicación procesa los mensajes dirigidos a ella.



3.3.5 Bucle de mensajes

En un mapa de mensajes, distinguimos tres tipos de mensajes:

- ✓ **Mensajes de órdenes:** son mensajes **WM_COMMAND**. Se generan cuando el usuario selecciona una orden de un menú, un acelerador (combinación de teclas que activan una orden de un menú) o un botón de la barra de herramientas.

  Para este tipo de mensajes, el mapa de mensajes utiliza la macro **ON_COMMAND**.

  Los controladores no tienen argumentos y no retornan un valor. Ejemplo:

```
/////////////////////////////////////////////////////////////////////////////
// CRelojApp

BEGIN_MESSAGE_MAP(CRelojApp, CWinApp)
  //{{AFX_MSG_MAP(CRelojApp)
  ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // 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
  // Standard file based document commands
  ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
  ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()
```

```
class CRelojApp : public CWinApp
{
public:
  CRelojApp();

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CRelojApp)
  public:
  virtual BOOL InitInstance();
  //}}AFX_VIRTUAL

// Implementation
  //{{AFX_MSG(CRelojApp)
  afx_msg void OnAppAbout();
    // NOTE - the ClassWizard will add and remove member functions here.
    //    DO NOT EDIT what you see in these blocks of generated code !
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
```

- ✓ **Mensajes de notificación:** es un mensaje que Windows envía a una ventana en respuesta a una tecla pulsada, al pulsar un botón, etc. Esto es, es un mensaje **WM_COMMAND** enviado por una ventana hija a su ventana padre, indicando el evento ocurrido. Por ejemplo, un botón envía un mensaje **BN_CLICKED** a su ventana padre para notificarle que ha sido pulsado.

  Para este tipo de mensajes, el mapa de mensajes utiliza la macro **ON_mensaje**.

  Los controladores no tienen argumentos y no retornan un valor.

- ✓ **Mensajes de ventana:** son mensajes no relacionados con controles. Estos mensajes son generados por el sistema o por una acción del usuario. Por ejemplo, cuando el usuario pulsa el botón izquierdo del ratón, se genera el mensaje **WM_LBUTTONDOWN**. Ejemplo:

  Para este tipo de mensajes, el mapa de mensajes utiliza la macro **ON_WM_xxx**.

  Los controladores tienen argumentos y pueden retornar un valor. Ejemplo:

```
IMPLEMENT_DYNCREATE(CRelojView, CFormView)

BEGIN_MESSAGE_MAP(CRelojView, CFormView)
  //{{AFX_MSG_MAP(CRelojView)
  ON_WM_DESTROY()
  ON_WM_TIMER()
  ON_BN_CLICKED(IDC_MOSTRAR, OnMostrarRelojAnalogico)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
class CRelojView : public CFormView
{

// ...

// Generated message map functions
protected:
  //{{AFX_MSG(CRelojView)
  afx_msg void OnDestroy();
  afx_msg void OnTimer(UINT nIDEvent);
  afx_msg void OnMostrarRelojAnalogico();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
```

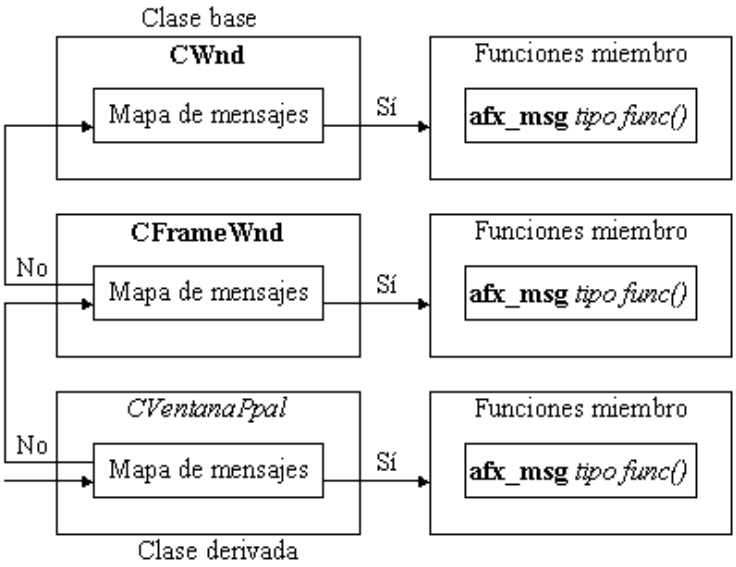
Cuando una ventana recibe un mensaje de ventana o de notificación (no mensajes de órdenes) la búsqueda del mensaje para su proceso se produce en su mapa de mensajes. Si en el mapa de mensajes de la clase de la ventana en cuestión no existe una entrada para el mensaje en proceso, la búsqueda por ese mensaje continúa en el mapa de mensajes de su clase base y así sucesivamente. En el caso más desfavorable, se ejecuta un manipulador por omisión.

Observe que la macro **BEGIN_MESSAGE_MAP** especifica el nombre de la clase derivada y de su clase base. Por ejemplo:

```
BEGIN_MESSAGE_MAP( CVentanaPpal, CFrameWnd )
```

El primer argumento especifica por dónde comienza la búsqueda. El segundo argumento indica por dónde debe continuar la búsqueda.

La ruta seguida para procesar un mensaje puede verse a continuación:



En cuanto a los mensajes de órdenes, un mensaje de este tipo es enviado a través de una ruta estándar de objetos destino de órdenes, hasta encontrar el objeto que tiene definido el controlador para dicho mensaje. Para seguir esta ruta, el objeto destino de la orden utiliza la función **OnCmdMsg** para determinar si puede procesar el mensaje y si no puede, se lo envía al siguiente objeto en la ruta. De forma genérica la ruta seguida es:

1. A su objeto hijo actualmente activo.
2. A sí mismo.
3. A otros objetos que pueden ser destino de órdenes.

3.3.6 Biblioteca de clases MFC

La biblioteca MFC (Microsoft Foundation Class Library – Biblioteca de clases base de Microsoft), constituye una interfaz orientada a objetos para Windows, que permite desarrollar aplicaciones para Windows de una forma más intuitiva (basado en el uso de ventanas).

La clase **CObject** es la clase raíz para la mayoría de las clases de la biblioteca MFC. La funcionalidad proporcionada por esta clase incluye:

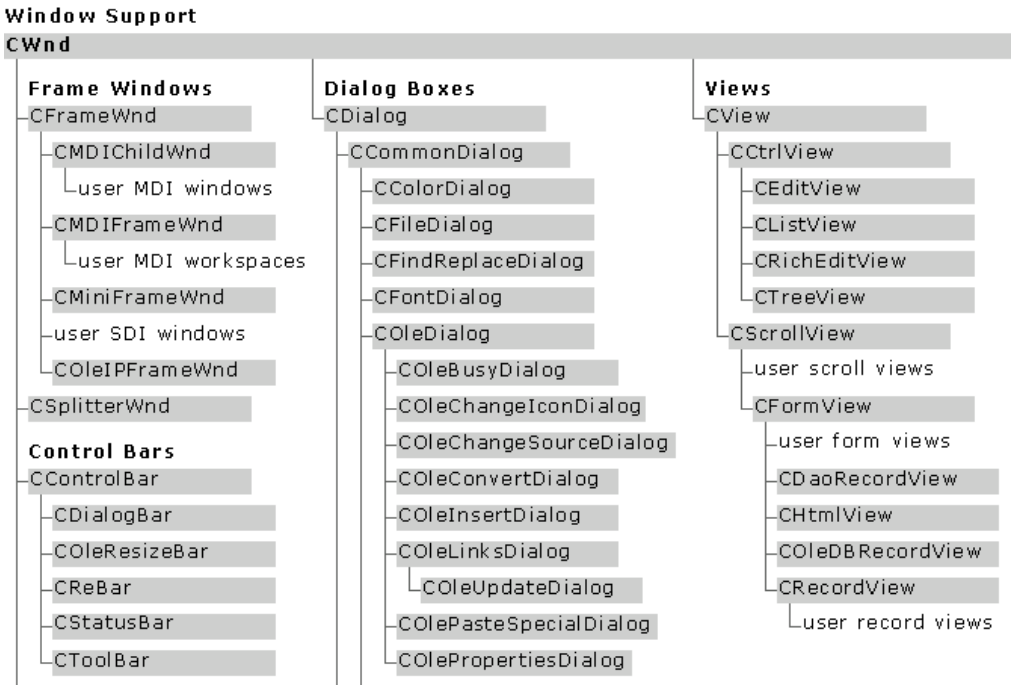
- ✓ Arquitectura de la aplicación
- ✓ Manejo de excepciones
- ✓ Soporte de ficheros
- ✓ Soporte gráfico
- ✓ Manejo de menús
- ✓ Bases de datos

- ✓ Objetos de sincronización (hilos)
- ✓ Sockets
- ✓ Colecciones (arrays, listas y mapas)
- ✓ Servicios de Internet
- ✓ Soporte de ventanas

Existen además otras clases no derivadas de CObject.

CWnd proporciona el soporte para trabajar con ventanas, tales como:

- ✓ Ventanas marco
- ✓ Barras de control
- ✓ Hojas de propiedades
- ✓ Cajas de diálogo
- ✓ Vistas
- ✓ Controles



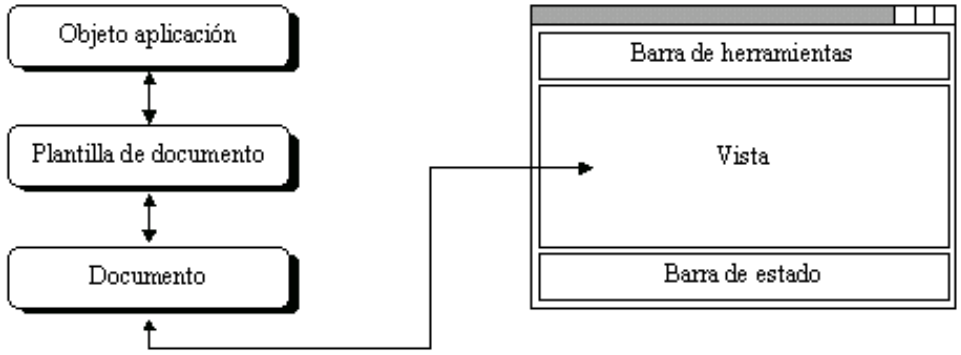
3.3.7 Arquitectura de una aplicación

Los objetos fundamentales que intervienen en una aplicación son:

- ✓ Aplicación
- ✓ Documento
- ✓ Ventana marco
- ✓ Vista

Existen dos tipos de arquitecturas:

- ✓ **SDI (*Single Document Interface*):** solo permite tener abierta una ventana marco con su vista, para presentar el documento que tiene abierto, que también es único por cada ejemplar activo de la aplicación.
- ✓ **MDI (Multiple Document Interface):** permite tener abiertas varias ventanas marco dentro de la ventana principal de la aplicación, es decir, dentro de la ventana marco principal pueden abrirse varias ventanas marco hijas, de las cuales sólo una estará activa, la que tiene la barra de título resaltada y cada ventana hija contiene una vista de un documento.



El objeto Aplicación, tiene las características siguientes:

- ✓ Es un objeto de una clase derivada de **CWinApp**.
- ✓ Inicia, ejecuta y finaliza la aplicación.
- ✓ Gestiona las plantillas de documento

El objeto Plantilla de documento, tiene las características siguientes:

- ✓ Es un objeto de una clase derivada de **CDocTemplate**.
- ✓ Una plantilla de documento crea y gestiona todos los documentos abiertos de un mismo tipo.
- ✓ Las aplicaciones que soportan más de un tipo de documento tienen varias plantillas de documento.

El objeto Documento, tiene las características siguientes:

- ✓ Es un objeto de una clase derivada de **CDocument**.

- ✓ Un documento representa una unidad de datos que el usuario abre con la orden *Abrir* del menú *Archivo* y guarda con la orden *Guardar*.

- ✓ Proporciona el soporte para la apertura y administración de ficheros en el disco.

El objeto Ventana marco, tiene las características siguientes:

- ✓ Es un objeto de una clase derivada de **CFrameWnd**.

- ✓ Proporciona los controles estándar de una ventana a las vistas utilizadas para visualizar los datos tales como el marco, la barra de título, el menú de control, los botones de maximizar y minimizar, la barra de herramientas y la barra de estado.

El objeto Vista, tiene las características siguientes:

- ✓ Es un objeto de una clase derivada de **CView**.

- ✓ Este objeto representa una ventana que hace de interfaz entre el usuario y los datos del documento.

- ✓ Un documento puede tener varias vista de los datos.

3.3.8 Análisis de una aplicación

El esqueleto de una aplicación basada en la arquitectura documento-vista tiene básicamente cuatro clases:

- ✓ Clase Aplicación

- ✓ Clase Ventana marco

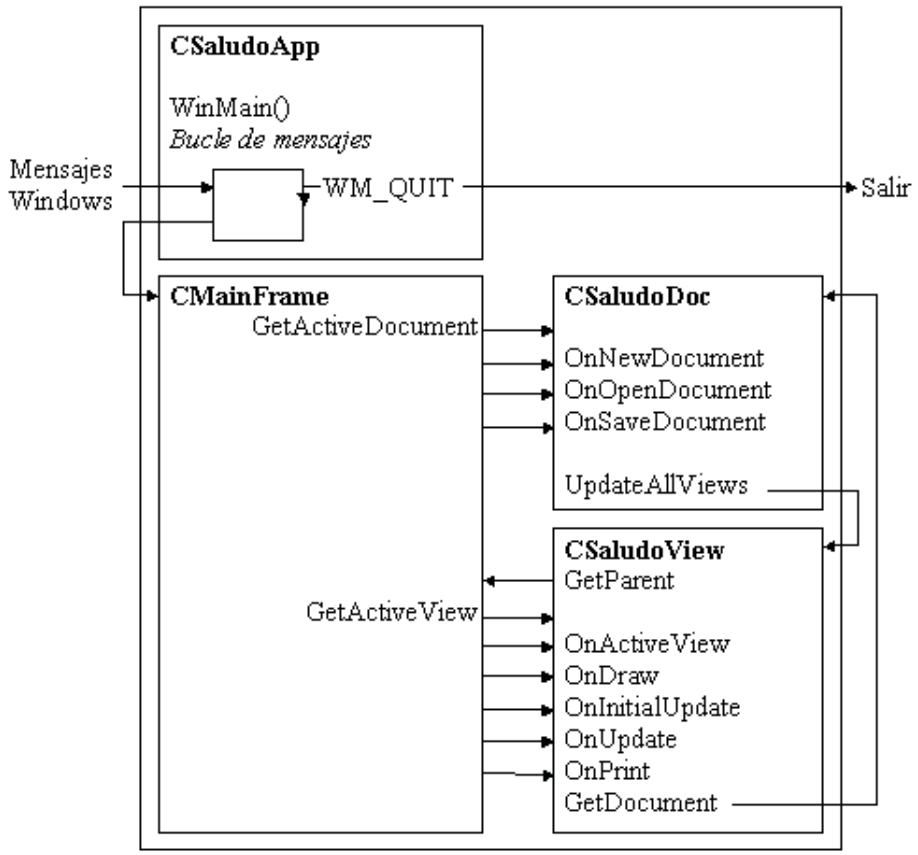
- ✓ Clase Documento

- ✓ Clase Vista

Cuando la aplicación está en ejecución y ocurre un evento, por ejemplo el usuario redimensiona una ventana o pulsa una tecla, el propio Windows envía un mensaje a la aplicación indicándole el nuevo tamaño de la ventana o la tecla que se ha pulsado, para que la aplicación proceda en consecuencia.

El objeto de la clase derivada de **CWinApp** representa la aplicación. El objeto **CMainFrame** representa la ventana principal de la aplicación, el objeto documento almacena los datos y el objeto vista visualiza los datos.

La figura siguiente muestra esquemáticamente cómo estos objetos se comunican entre sí (sólo se han representado algunas de las llamadas):


CSaludoApp
WinMain()
Bucle de mensajes
Mensajes
Windows
WM_QUIT
Salir
CMainFrame
GetActiveDocument
CSaludoDoc
OnNewDocument
OnOpenDocument
OnSaveDocument
UpdateAllViews
CSaludoView
GetParent
GetActiveView
OnActiveView
OnDraw
OnInitialUpdate
OnUpdate
OnPrint
GetDocument

3.4 UNIDAD II: MI PRIMERA APLICACIÓN

Título del tema:

Mi primera aplicación

Objetivos:

- Entender los pasos que hay que seguir para generar una aplicación Windows.
- Aplicar las funciones correspondientes al uso de la fecha y la hora en la aplicación

Contenido:

- Diseño de una ventana principal
- Iniciar la vista
- Fecha y Hora

Duración:

- 4 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

3.4 UNIDAD II: MI PRIMERA APLICACIÓN: "HOLA MUNDO"

3.4.1 Diseño de una ventana principal

Los pasos que debemos seguir para crear una ventana principal son:

1. Generar el esqueleto de la aplicación.
2. Dibujar los controles de la aplicación.
3. Establecer las propiedades de los objetos.
4. Unir el código a los objetos.

Este último paso se puede hacer de dos formas:

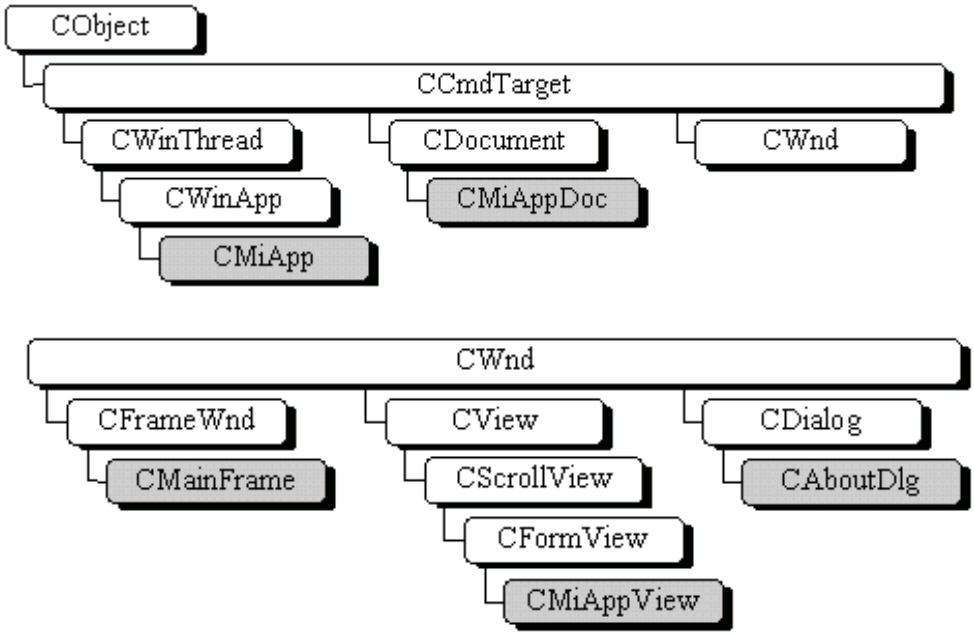
- ✓ Utilizando funciones miembro.
- ✓ Utilizando variables miembro.

Veamos ahora cada uno de los pasos.

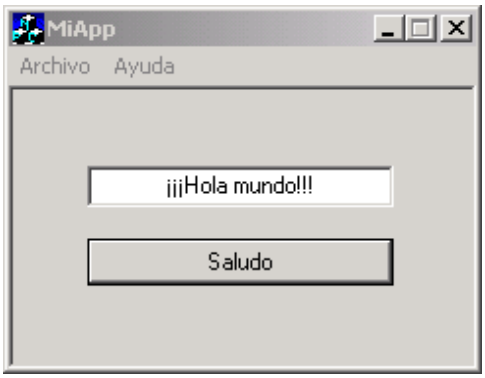
En lo que se refiere al esqueleto de la aplicación, esta se genera haciendo uso del asistente AppWizard cuando iniciamos una nueva aplicación. El asistente genera lo que se conoce como arquitectura documento-vista la cual incluye una serie de clases que conforman la aplicación. Por ejemplo, si creamos una nueva aplicación llamada MiApp, AppWizard generará las clases siguientes:

- ✓ CMiApp
- ✓ CMiAppDoc
- ✓ CMainFrame
- ✓ CMiAppView
- ✓ CAboutDlg

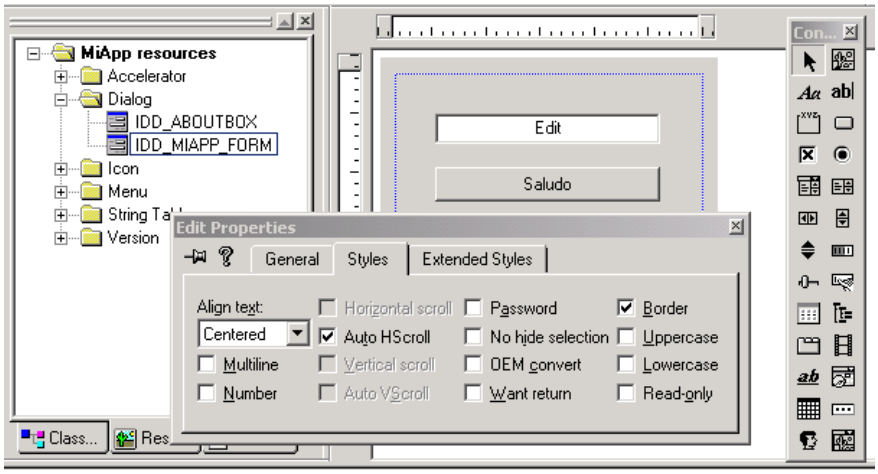
En las siguientes figuras se pueden ver sombreadas las clases que forman la arquitectura documento-vista dentro de la jerarquía de clases de la MFC para nuestro ejemplo anterior de MiApp:



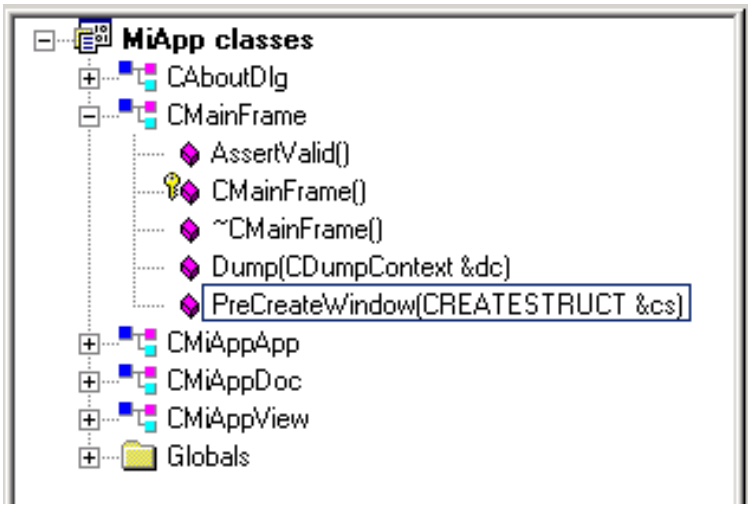
El paso 2 consiste en añadir a la ventana los diferentes controles que van a formar parte de nuestra aplicación. En el siguiente ejemplo, se ha incluido una caja de texto (Edit Box) y un botón (Button):



El paso 3 consiste en establecer las propiedades a cada uno de los objetos de la aplicación según la funcionalidad que se quiera lograr. Par ver las propiedades de un control hay que presionar clic derecho del ratón cuando esté ubicado sobre el control. En el ejemplo siguiente se pueden ver las propiedades de la caja de texto anterior:



En el paso 4 debemos unir el código para que la aplicación responda a nuestras necesidades. Para llevar a cabo esta tarea nos podemos ayudar con el asistente. Para ubicarnos, recordemos que tenemos la vista de clases en la ventana del WorkSpace:

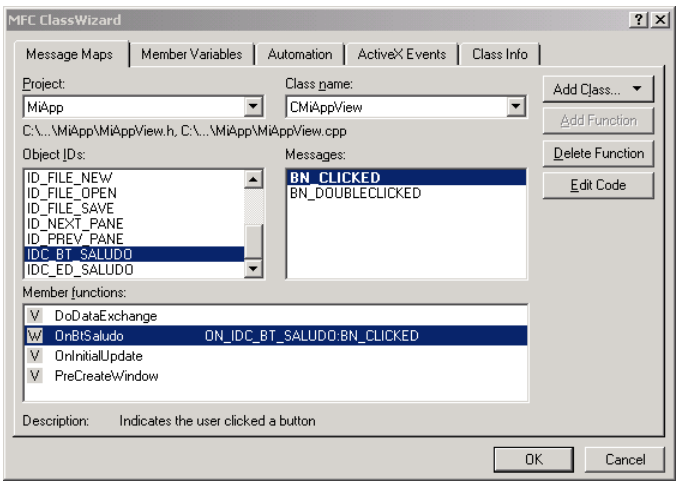


La figura siguiente muestra el código de la función **PreCreateWindow** de la clase **CMainFrame** de nuestra aplicación de ejemplo MiApp (esta función es llamada cuando el marco de la ventana es creado y sirve para establecer sus características):

```
}
  return TRUE;

  cs.style &= ~(LONG_PTR)(WS_THICKFRAME | WS_MAXIMIZEBOX);

  if( !CFrameWnd::PreCreateWindow(cs) ) return FALSE;
{
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
```

El código anterior hace que el marco de la ventana de la aplicación se visualice sin poner el título del documento en la barra de título; que dicha ventana no permita variar su tamaño y que el botón de maximizar permanezca inactivo. Para

añadir una función haciendo uso del asistente debemos presionar **CTRL W** cuando nos encontremos encima de un determinado objeto. Por ejemplo, la figura siguiente muestra como añadir la función **OnBtSaludo** para que dé respuesta al mensaje **BN_CLICKED** cuando presionemos el botón cuyo identificador es **IDC_BT_SALUDO**; en este caso tuvimos que posicionarnos sobre el botón y luego invocar al asistente:



Para escribir el código lo podemos hacer de dos maneras: utilizando funciones miembro predefinidas ó por medio del uso de variables miembro asociadas a los controles. Ambas formas serán vistas a continuación.

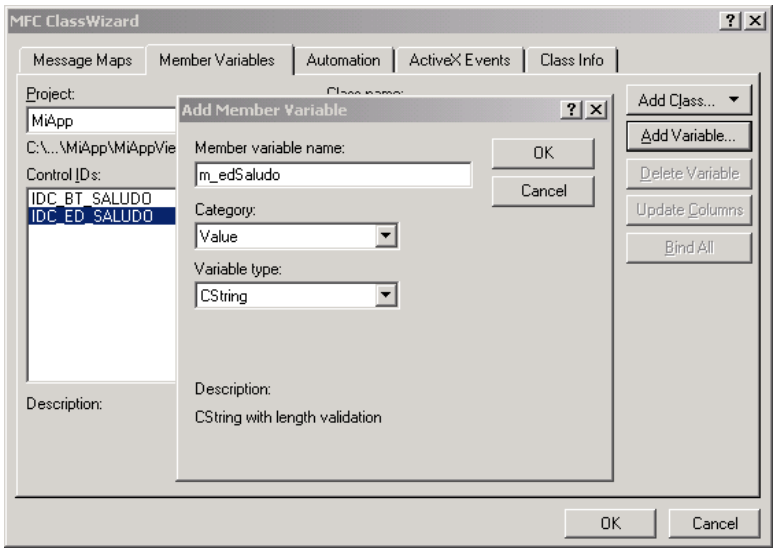
El objetivo de la aplicación de ejemplo **MiApp** es que al dar clic sobre el botón se visualice en la caja de texto el mensaje **¡¡¡Hola Mundo!!!**.

Si lo hacemos utilizando funciones miembro debemos escribir el código siguiente:

```
void CMiAppView::OnBtSaludo()
{
  SetDlgItemText(IDC_ED_SALUDO, "¡¡¡Hola mundo!!!");
}
```

La función **SetDlgItemText** se encarga de poner el texto especificado en el segundo argumento en el control cuyo id es el primer argumento; en nuestro caso en la caja de texto. Esta acción se produce cuando demos clic en el botón.

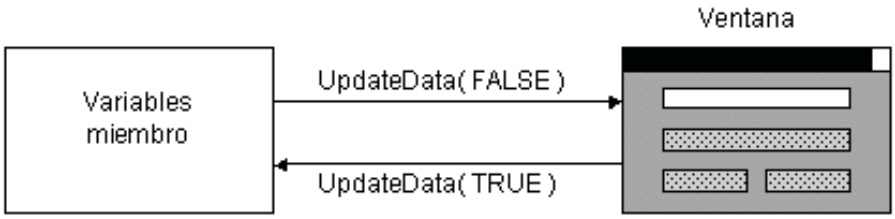
Si lo hacemos utilizando variables miembro debemos invocar a **ClassWizard** para asociar una variable de tipo **CString** con la caja de texto tal y como se muestra a continuación:



En nuestro ejemplo, esta variable se llama **m_edSaludo** y ella contendrá el texto que queremos mostrar. Ahora el código debe ser el siguiente:

```
void CMiAppView::OnBtSaludo()
{
  m_edSaludo = "¡¡¡Hola mundo!!!";
  UpdateData(FALSE);
}
```

La función **UpdateData** con el argumento **FALSE** se encarga de enviar el contenido de la variable a la correspondiente caja de texto asociada. Si el argumento fuera **TRUE** el proceso es el inverso: el contenido del control se transfiere a la variable miembro. Esto último es muy útil cuando se quieren capturar datos introducidos por el usuario. La figura siguiente aclara lo dicho:



Cuando una variable miembro es añadida por medio del asistente, éste genera automáticamente un código como el siguiente:

```
class CMiAppView : public CFormView
{
protected: // create from serialization only
  CMiAppView();
  DECLARE_DYNCREATE(CMiAppView)

public:
  //{{AFX_DATA(CMiAppView)
  enum { IDD = IDD_MIAPP_FORM };
  CString m_edSaludo;
  //}}AFX_DATA

// Attributes
public:
```

```
void CMiAppView::DoDataExchange(CDataExchange* pDX)
{
  CFormView::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CMiAppView)
  DDX_Text(pDX, IDC_ED_SALUDO, m_edSaludo);
  //}}AFX_DATA_MAP
}
```

Las macros AFX_MSG delimitan las funciones miembro ligadas a los mensajes de un mapa de mensajes de las MFC. Por eso, debemos tener cuidado de ***no borrar estas macros***. En la primera figura podemos ver la declaración de la variable y en la segunda la conexión entre la variable y la caja de texto.

3.4.2 Iniciar la vista

Antes de que la vista sea mostrada por primera vez se invoca a la función **OnInitialUpdate** de la clase **CMiAppView**. Esta función fue heredada de **CFormView**. Tiene como objetivo establecer ciertas características asociadas a la vista:

```
void CMiAppView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();
  GetParentFrame()->RecalcLayout();
  ResizeParentToFit();

}
```

La función **CWnd::GetParentFrame** se encarga de obtener un puntero a la ventana marco. La función **CFrameWnd::RecalcLayout** es invocada cuando las barras de control estándar aparecen o desaparecen, o bien cuando se redimensiona la ventana. La función **CScrollView::ResizeParentToFit** ajusta la ventana marco al tamaño de la vista.

3.4.3 Fecha y Hora

La clase **CTime** permite obtener información relativa a la fecha y la hora actuales. Para visualizar la fecha y la hora de acuerdo con diversos patrones, utilice la función **Format** (consultar la ayuda en línea de **MSN Visual Studio**).

Dentro de las funciones de la clase CTime tenemos:

- ✓ **GetCurrentTime**: proporciona la fecha y la hora del sistema
- ✓ **Format**: para visualizar la fecha y la hora de diversas maneras.
- ✓ **GetDay, GetMonth, GetYear**: obtener el día, mes y año.

- ✓ **GetHour, GetMinute, GetSecond**: obtener la hora, minutos y segundos.

Un objeto de la clase **CTimeSpan** representa la diferencia que hay entre dos valores CTime. Ejemplo:

```
CTime t1( 2001, 3, 19, 22, 15, 0 ); // 10:15PM Marzo 19, 2001
CTime t2( 2001, 3, 20, 22, 15, 0 ); // 10:15PM Marzo 20, 2001
CTimeSpan ts = t2 - t1;  // Restar 2 CTimes
```

Otro ejemplo:

```
// ...
  CTimeSpan m_DesfaseHorario;
// ...

void CRelojView::CambiarHora(int nHora, int nMinuto, int nSegundo)
{
  CTime FechaHora = CTime::GetCurrentTime();

  CTime FechaHoraNueva(FechaHora.GetYear(),
                       FechaHora.GetMonth(),
                       FechaHora.GetDay(),
                       nHora, nMinuto, nSegundo);

  m_DesfaseHorario = FechaHoraNueva - FechaHora;

  MostrarHoraActual();
}
```

3.5 UNIDAD III: CAJAS DE DIÁLOGO, GRÁFICOS, TIEMPO Y USO DEL RATÓN

Título del tema:

Cajas de diálogo, gráficos, tiempo y uso del ratón

Objetivos:

- Entender y aplicar conceptos varios en una aplicación Windows tales como cajas de diálogo, uso de gráficos, temporizadores y manejo del ratón

Contenido:

- Cajas de diálogo
- Gráficos
- Temporizador
- Uso del ratón

Duración:

- 6 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

3.5 UNIDAD III: CAJAS DE DIÁLOGO, GRÁFICOS, TIEMPO Y USO DEL RATÓN

3.5.1 Cajas de diálogo

Una caja de diálogo es una ventana con controles para aceptar nuevos datos o para visualizarlos. La funcionalidad para manipular cajas de diálogo es proporcionada por la clase **CDialog**. Existen 3 tipos:

- ✓ *Cajas de diálogo personalizadas*: son cajas de diálogo hechas a medida, bien añadiendo controles a un formulario, o bien modificando cajas de diálogo existentes.

- ✓ *Cajas de diálogo predefinidas:* son cajas de diálogo creadas por medio de las funciones de la biblioteca MFC; por ejemplo, **AfxMessageBox**.

- ✓ *Cajas de diálogo estándar:* son cajas de diálogo muy comunes en Windows; por ejemplo, la caja de diálogo *Abrir* o *Imprimir*.

3.5.2 Cajas de diálogo modales y no modales

- ✓ *Cajas de diálogo modales:* tiene que ser cerrada para que la aplicación pueda continuar.

- ✓ *Cajas de diálogo no modales:* no tiene que ser cerrada para que la aplicación pueda continuar.

3.5.3 Crear una caja de diálogo modal

Los pasos para crear una caja de diálogo modal son los siguientes:

- ✓ *Diseñar la plantilla de diálogo:* es decir, diseñar la ventana con sus controles cuyo identificador será pasado como parámetro en el constructor de la clase;

la plantilla se crea con el editor de diálogos del entorno de desarrollo de Visual C++.

- ✓ *Crear una clase derivada de **CDialog** que utilice esa plantilla:* esto se hace con ayuda de ClassWizard una vez que se ha creado la plantilla.

- ✓ *Definir un objeto de la nueva clase:* esto se hace en el lugar donde se quiera utilizar el diálogo modal.

- ✓ *Invocar a la función **DoModal** del objeto:* esto se hace en el lugar donde se quiera visualizar el diálogo modal. Ejemplo:

```
void CMiAppApp::OnAppAbout()
{
  CAboutDlg aboutDlg;
  aboutDlg.DoModal();
}
```

A continuación podemos ver los botones que tiene una caja de diálogo modal y las funciones miembros controladoras que dan respuesta a los eventos producidos por éstos:

Botón	Identificador	Función miembro
Aceptar	IDOK	CDialog::OnOk
Cancelar	CANCEL	CDialog::OnCancel

Cuando se cierra una caja de diálogo modal o no modal (o sea pulsar los botones Aceptar/Cancelar) se invoca a la función **CDialog::EndDialog**. Esta función se encarga solamente de ocultar el diálogo (no lo destruye).

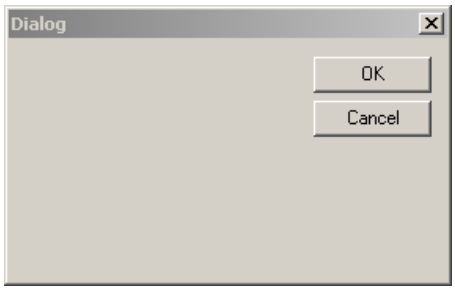
Si se desea destruir la ventana hay que invocar a la función:

CWnd::DestroyWindow

3.5.4 Crear una caja de diálogo no modal

Los pasos para crear una caja de diálogo no modal son los siguientes:

- ✓ *Diseñar la plantilla de diálogo:*



- ✓ *Crear una clase derivada de **CDialog** que utilice esa plantilla:*

```
class CMiDialogo : public CDialog
{
// Construction
public:
  CMiDialogo(CWnd* pParent = NULL);

  // ...

  DECLARE_MESSAGE_MAP()
};
```

✓ *Invocar a la función **CDialog::Create** desde el constructor de esa clase:*

```
CMiDialogo::CMiDialogo(CWnd* pParent /*=NULL*/)
  : CDialog(CMiDialogo::IDD, pParent)
{
  //{{AFX_DATA_INIT(CMiDialogo)
    // NOTE: the ClassWizard will add ...
  //}}AFX_DATA_INIT
  Create(IDD_DIALOG1);
}
```

✓ *Crear un objeto dinámicamente de la clase derivada:*

```
void CMiAppView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();
  GetParentFrame()->RecalcLayout();
  ResizeParentToFit();

  m_pDlg = new CMiDialogo(this);
}
```

```
void CMiAppView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();
  GetParentFrame()->RecalcLayout();
  ResizeParentToFit();

  m_pDlg = new CMiDialogo(this);
}
```

✓ *Eliminar el objeto:*

```
CMiAppView::~CMiAppView()
{
  delete m_pDlg;
}
```

3.5.5 Avisos y mensajes

La macro **TRACE** visualiza los argumentos especificados en la ventana de salida *Debug*; es muy útil en procesos de depuración. **TRACE** se utiliza igual que **printf** y está operativa siempre que se ejecute la versión *Debug* de su aplicación en modo depuración; esto es, pulsando *F5* y no *Ctrl+F5*. Ejemplo:

```
int i = 1;
char sz[] = "Hola";
TRACE( "Entero = %d, Cadena = %s\n", i, sz );
// Output: 'Entero = 1, Cadena = Hola'
```

La función **AfxMessageBox** permite visualizar un mensaje en una ventana. La sintaxis es la siguiente:

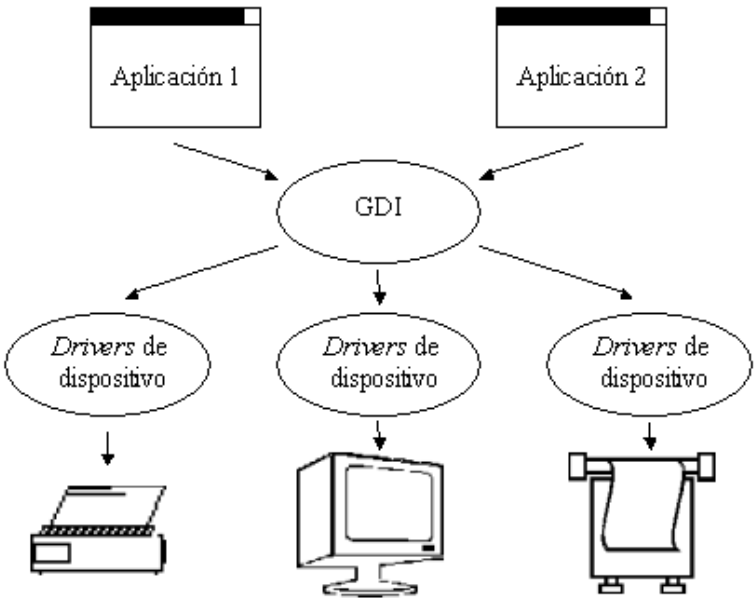
```
int AfxMessageBox( LPCTSTR lpszText, UINT nType = MB_OK,
UINT nIDHelp = 0 );

int AFXAPI AfxMessageBox( UINT nIDPrompt, UINT nType =
MB_OK, UINT nIDHelp = (UINT) -1 );
```

El parámetro *lpszText* es un objeto *CString* o la dirección de una cadena de caracteres, que contiene el mensaje que se desea visualizar.

El entero *nType* es una suma lógica (OR) de valores que describe el número y el tipo de botones a visualizar, el estilo del icono y cuál es el botón por omisión. (Para ver estos valores, consulte la ayuda MSDN de AfxMessageBox).

3.5.6 Gráficos

Para pintar los gráficos (incluyendo texto) así como para mostrar los elementos que componen una interfaz de usuario, Windows utiliza la interfaz de dispositivos gráficos (**GDI: *Graphics Device Interface***), la cual se encarga de llamar a las rutinas de los distintos gestores de dispositivo (*drivers* de vídeo, de impresora y de trazadores gráficos) que son los que directamente actúan sobre el dispositivo.



Puesto que existen muchos dispositivos gráficos de salida diferentes soportados por Windows, uno de los objetivos principales de la GDI es permitir mostrar los gráficos independientemente del dispositivo utilizado. De esta forma podemos

escribir aplicaciones que se ejecuten independientemente de las características particulares del dispositivo gráfico de salida que utilice una determinada máquina.

3.5.7 Pintar

En una ventana, Windows sólo puede pintar en el área de cliente. Para informar sobre la necesidad de pintar, Windows envía un mensaje **WM_PAINT** a su procedimiento de ventana.

La función controladora, es decir, que da respuesta a este tipo de mensajes es **OnPaint**.

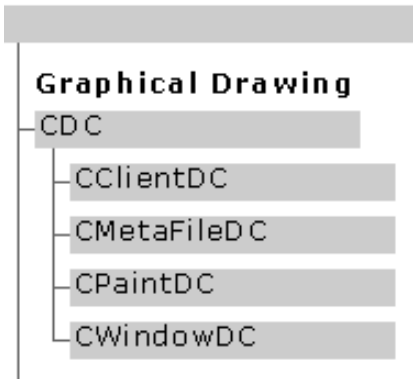
3.5.8 Contexto de dispositivo

Como Windows no permite un acceso directo al hardware de visualización, comunica con él utilizando un nivel de abstracción por encima del hardware denominado contexto de dispositivo (***device context* - DC**). Esto quiere decir que para poder dibujar en un dispositivo de salida gráfico primero hay que obtener un contexto de dispositivo relativo a ese dispositivo físico.

Un contexto de dispositivo es una estructura de datos, mantenida internamente por la GDI que contiene los atributos necesarios para pintar el gráfico o el texto. Estos atributos son utilizados por las funciones de la GDI.

3.5.9 CDC

La clase base **CDC** proporciona todas las funciones miembro necesarias para dibujar:



Para obtener información del contexto de dispositivo se pueden usar las funciones:

```
CDC::GetDeviceCaps(indice)
CDC::GetSystemMetrics(indice)
```

Cuando un contexto de dispositivo está vinculado a un dispositivo gráfico de salida, tal como la pantalla o la impresora, y necesitemos obtener información sobre él (por ejemplo, tamaño de la superficie de proyección), posibilidades de color, puntos por pulgada, etc.) invocaremos a la función miembro **GetDeviceCaps** de la clase **CDC**:

```
int GetDeviceCaps( int nIndex ) const;
```

donde *nIndex* es un identificador que indica la información que se desea obtener (por ejemplo):

- ✓ HORZRES: permite obtener el ancho en píxeles de la pantalla.
- ✓ VERTRES: permite obtener el alto en píxeles de la pantalla.
- ✓ LOGPIXELSX: permite obtener el número de píxeles por pulgada lógica en el ancho de la pantalla.
- ✓ LOGPIXELSY: permite obtener el número de píxeles por pulgada lógica en el alto de la pantalla.

Para recuperar las magnitudes de varios elementos de visualización de Windows (pantalla, icono, cursor, borde, barra de título, etc.) usaremos la función **GetSystemMetrics**:

```
int GetSystemMetrics( int nIndex );
```

donde nIndice es un identificador que indica la información que se desea obtener (por ejemplo):

- ✓ SM_CXMIN: permite obtener el ancho mínimo en píxeles de la ventana.
- ✓ SM_CYMIN: permite obtener el alto mínimo en píxeles de la ventana.
- ✓ SM_CXSCREEN: permite obtener el ancho en píxeles de la pantalla.
- ✓ SM_CYSCREEN: permite obtener el alto en píxeles de la pantalla.
- ✓ SM_CXVSCROLL: permite obtener el ancho en píxeles de la barra de desplazamiento vertical.
- ✓ SM_CYHSCROLL: permite obtener el alto en píxeles de la barra de desplazamiento horizontal.

El siguiente código muestra un ejemplo de cómo utilizar la función **GetDeviceCaps** para obtener información sobre la impresora:

```
CPrintDialog dlg(FALSE);

if (!dlg.GetDefaults())
 AfxMessageBox(_T("Usted no tiene impresora instalada"));
else
{
CDC dc;
dc.Attach(dlg.m_pd.hDC);

int nHorz = dc.GetDeviceCaps(LOGPIXELSX);
int nVert = dc.GetDeviceCaps(LOGPIXELSY);

CString str;

if (nHorz == nVert)
```

```
str.Format(_T("Su impresora soporta %d pixeles por
pulgada"), nHorz);

else

str.Format(_T("Su impresora soporta %d pixeles por pulgada
") _T("resolución horizontal, y %d pixeles por pulgada
resolución ") _T("vertical"), nHorz, nVert);

AfxMessageBox(str);
}
```

Este otro ejemplo muestra el uso de la función **GetSystemMetrics** para obtener información relativa al ratón:

```
#include <windows.h>
#include <stdio.h>

void main()
{
   BOOL fResult;

   fResult = GetSystemMetrics(SM_MOUSEPRESENT);

   if (fResult == 0)
      printf("No hay ratón instalado.\n");
   else
   {
      printf("Ratón instalado.\n");

      fResult = GetSystemMetrics(SM_SWAPBUTTON);

      if (fResult == 0)
         printf("Botones sin capacidad de intercambio\n");
      else printf("Botones con capacidad de intercambio\n");

   }
}
```

En el cuadro siguiente podemos ver algunas funciones miembro de la clase CDC que se pueden usar para dibujar:

Tipo	Función miembro
Puntos	SetPixel
Posición	MoveTo, GetCurrentPosition
Líneas	LineTo, Arc, Polyline, etc.
Figuras	Rectangle, Ellipse, Polygon, etc

Por ejemplo:

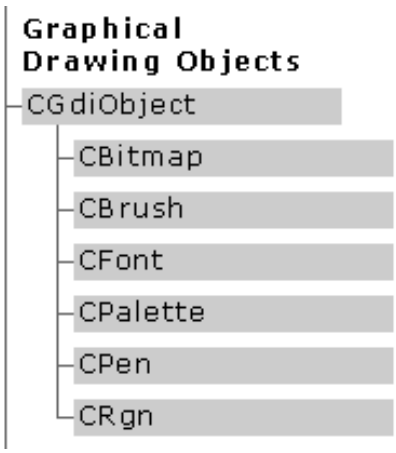
```
void CDibujoView::OnPaint()
{

CPaintDC dc(this); // device context for painting

dc.SetPixel(10,33,RGB(255, 255, 0));
dc.LineTo(50,50);
dc.MoveTo(100,60);
dc.LineTo(200,90);
dc.Rectangle(0,0,200,200);
dc.Ellipse(0,0,200,200);

}
```

Para mayores detalles de cada una de estas funciones y otras más hay que consultar la ayuda.

3.5.10 Objetos GDI

La GDI de Windows emplea varias herramientas de dibujo, incluyendo plumas, pinceles, tipos de letra, mapas de bits, paletas de color y regiones. Cada una de estas herramientas está contemplada en las MFC por una clase derivada de **CGdiObject**. Estas clases derivadas son las siguientes: **CPen**, **CBrush**, **CFont**, **CBitmap**, **CPalette** y **CRgn**.

La figura siguiente muestra la jerarquía de clases :



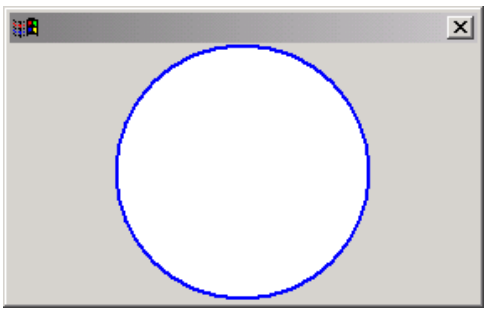
Para utilizar objetos GDI, debemos seguir los pasos siguientes:

- ✓ Crear el objeto GDI.
- ✓ Seleccionarlo para el DC (contexto de dispositivo a utilizar). Esto se logra con la función **CDC::SelectObject**.
- ✓ Eliminar el objeto cuando no se vaya a utilizar más. Esto se logra con la función **CGdiObject::DeleteObject**.

Por ejemplo:

```
void CMiDialogo::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  // TODO: Add your message handler code here
  CRect rect;
  GetClientRect(rect);

  int cx = rect.right - rect.left;
  int cy = rect.bottom - rect.top;
  int r = cy < cx ? cy/2 : cx/2;

  CPen PlumaAzul(PS_SOLID, 2, RGB(0, 0, 255));
  CPen *pPlumaAnterior = dc.SelectObject(&PlumaAzul);

  dc.Ellipse(cx/2-r, cy/2-r, cx/2+r, cy/2+r);

  dc.SelectObject(pPlumaAnterior);

  PlumaAzul.DeleteObject(); // no necesaria en este caso
}
```

En el código mostrado anteriormente, primero se declara un contexto de dispositivo *dc* para poder pintar. A continuación se obtiene por medio del objeto *rect* el área de cliente donde se dibujará, almacenando en las variables *cx* y *cy* el ancho y el alto respectivamente de dicha área. Luego se crea el objeto *PlumaAzul* que se usará para dibujar. Después se selecciona esta pluma, almacenando en *pPlumaAnterior* la pluma que estaba seleccionada por defecto; esto se hace para seleccionarla de nuevo cuando dejemos de usar la pluma azul. Por último, se dibuja una elipse, se selecciona la pluma anterior y se elimina la pluma azul utilizada para dibujar la elipse. El resultado es:



Este otro ejemplo muestra la forma de cómo pintar con un color de fondo una elipse, para ello hacemos uso de un objeto de tipo **CBrush**:

```
CBrush BrochaGris(::GetSysColor(COLOR_3DFACE));
CBrush *pBrochaAnterior;

pBrochaAnterior = dc.SelectObject(&BrochaGris);

dc.Ellipse(cx/2-r, cy/2-r, cx/2+r, cy/2+r);

dc.SelectObject(pBrochaAnterior);
```

3.5.11 Redimensionar / repintar

La función miembro **Invalidate** de la clase **CWnd** invalida el área de trabajo del objeto **CWnd** para el que es invocada. Esta función tiene un parámetro que especifica si el fondo debe ser borrado; por omisión vale true (en este caso, Windows envía un mensaje **WM_ERASEBKGND**).

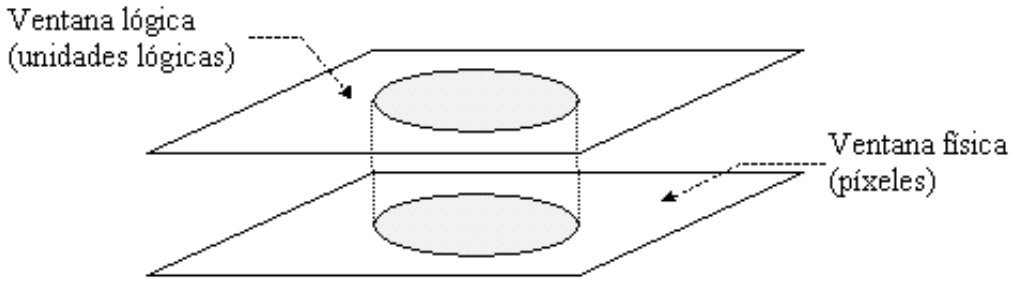
Esta área está marcada para ser pintada cuando ocurre el siguiente mensaje **WM_PAINT**. Por esta razón, es necesario llamarla cuando redimensionamos una ventana (es decir, cuando se envía el mensaje **WM_SIZE**), esto se muestra a continuación:

```
void CMiDialogo::OnSize(UINT nType, int cx, int cy)
{
  CDialog::OnSize(nType, cx, cy);

  // TODO: Add your message handler code here
  Invalidate();

}
```

3.5.12 Dibujar

Cuando dibujamos algo, las unidades por defecto son los píxeles o “unidades de dispositivo”. Sin embargo, Windows proporciona varios modos de proyección que se pueden asociar con el dispositivo sobre el que se está dibujando. También tiene que ver en esto la resolución de la pantalla.

Una aplicación que tiene que realizar gráficos se escribe para que dibuje sobre la ventana lógica (una ventana con un tamaño en unas unidades elegidas por nosotros). Windows, utilizando la razón entre el tamaño de la ventana lógica y el tamaño de la ventana física, proyectará el dibujo sobre la ventana física. La siguiente figura muestra lo dicho anteriormente:



Las unidades de un sistema de coordenadas *x-y* dependen del modo de proyección empleado. Los modos son los siguientes:

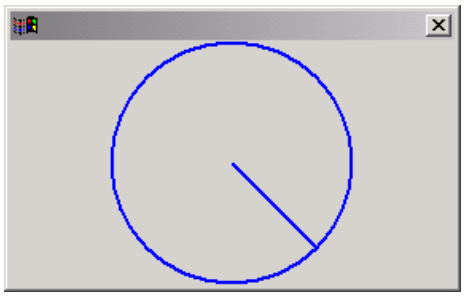
Modo de proyección	Unidad lógica =	Eje X	Eje Y
		Los valores aumentan hacia	
MM_TEXT	1 píxel	la derecha	abajo
MM_TWIPS	1 twip	la derecha	arriba

MM_LOMETRIC	0,1 mm	la derecha	arriba
MM_HIMETRIC	0,01 mm	la derecha	arriba
MM_LOENGLISH	0,01 pulgadas	la derecha	arriba
MM_HIENGLISH	0,001 pulgadas	la derecha	arriba
MM_ISOTROPIC	arbitrario (x == y)	Seleccionable	Seleccionable
MM_ANISOTROPIC	arbitrario (x != y)	Seleccionable	Seleccionable

Por ejemplo, si suponemos que el modo de proyección es MM_TEXT, en el cual la coordenada X aumenta hacia la derecha y el eje Y aumenta hacia abajo y que tenemos el código:

```
dc.Ellipse(cx/2-r, cy/2-r, cx/2+r, cy/2+r);
dc.MoveTo(cx/2, cy/2);
dc.LineTo(cx/2+r*cos(PI/4), cy/2+r*sin(PI/4));
```

el resultado es el siguiente:



Para personalizar el modo de proyección a utilizar, podemos usar las funciones siguientes:

- ✓ **CDC::SetMapMode**: ajusta el sistema de coordenadas al modo de proyección que se le pasa como parámetro. Ejemplo:

```
dc.SetMapMode(MM_ISOTROPIC);
```

- ✓ **CDC::SetWindowExt**: establece el tamaño lógico deseado para la ventana lógica. Ejemplo:

```
dc.SetWindowExt(200, 200);
```

- ✓ **CDC::SetViewportExt**: establece el tamaño de la superficie de proyección. En el ejemplo siguiente el signo menos en el segundo parámetro hace que los valores de y aumenten hacia arriba si tomamos en cuenta que hemos seleccionado el modo de proyección MM_ISOTROPIC:

```
dc.SetViewportExt(cx, -cy);
```

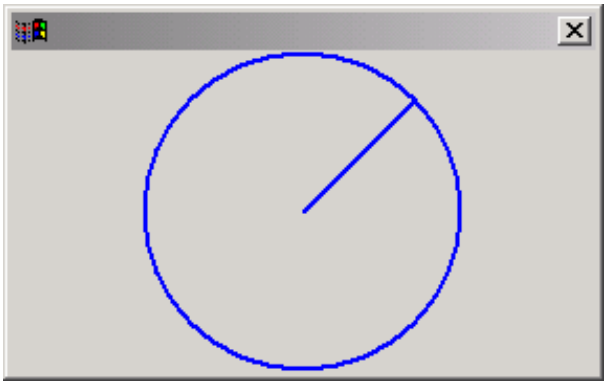
- ✓ **CDC::SetViewportOrg**: se usa para situar el origen lógico de coordenadas en el lugar deseado del área de cliente. Ejemplo:

```
dc.SetViewportOrg( cx/2, cy/2);
```

A continuación tenemos el ejemplo completo:

```
// Modo de proyección
dc.SetMapMode(MM_ANISOTROPIC);
// Tamaño de la ventana lógica
dc.SetWindowExt(cx, cy);
// Tamaño de la superficie de proyección
dc.SetViewportExt(cx, -cy);
// Origen lógico de coordenadas
dc.SetViewportOrg(cx/2, cy/2);

dc.Ellipse(-r, r, r, -r);
dc.MoveTo(0, 0);
dc.LineTo(r*cos(PI/4), r*sin(PI/4));
```

El resultado del código anterior es el siguiente:



3.5.13 Temporizadores

Un temporizador es una rutina interna de Visual C++ que notifica periódicamente a una aplicación cuando ha transcurrido un período determinado de tiempo.

Cada vez que transcurre el período de tiempo especificado, Windows coloca un mensaje **WM_TIMER** en la cola de mensajes de la aplicación. El procesamiento de este mensaje hace que se ejecute la función **OnTimer**, lugar donde se especificarán las acciones a ejecutar periódicamente.

Para instalar un temporizador hay que llamar a la función miembro **SetTimer** de la clase **CWnd**:

```
CWnd::SetTimer

UINT SetTimer( UINT nIDEvent, UINT nElapse, void (CALLBACK
EXPORT* lpfnTimer)(HWND, UINT, UINT, DWORD) );
```

El parámetro *nIDEvent* especifica el identificador del temporizador y el parámetro *nElapse* especifica el valor de tiempo en milisegundos. El tercer parámetro

lpfnTimer indica la dirección donde internamente va a ir alojado el temporizador, es decir, a qué objeto pertenece el temporizador, normalmente vale NULL porque pertenece a la ventana actual. Ejemplo:

```
BOOL CMiDialogo::OnInitDialog()
{
  CDialog::OnInitDialog();

  m_nIdTimer = SetTimer(ID_1SEGUNDO, 1000, NULL);

  return TRUE;
}

void CMiDialogo::OnTimer(UINT nIDEvent)
{
  m_n = ++m_n % 60;
  Invalidate();

  CDialog::OnTimer(nIDEvent);
}
```

Cuando ya no usemos el temporizador, debemos llamar a la función **KillTimer** miembro de **CWnd** para detener los mensajes **WM_TIMER**.

```
CWnd::KillTimer

BOOL KillTimer( int nIDEvent );
```

El parámetro *nIDEvent* es el identificador del temporizador usado en SetTimer.

3.5.14 El ratón

Las ventanas y varios tipos de controles reconocen tres eventos producidos por el ratón:

- ✓ Clic
- ✓ Doble clic
- ✓ Arrastrar

La tabla siguiente muestra los mensajes que pueden ser producidos por el ratón:

Acción	Mensaje	Función
Mover el ratón	WM_MOUSEMOVE	OnMouseMove
Pulsar el botón izquierdo	WM_LBUTTONDOWN	OnLButtonDown
Soltar el botón izquierdo	WM_LBUTTONUP	OnLButtonUp
Hacer doble clic con el botón izquierdo	WM_LBUTTONDBLCLK	OnLButtonDblClk

Pulsar el botón derecho	WM_RBUTTONDOWN	OnRButtonDown
Soltar el botón derecho	WM_RBUTTONUP	OnRButtonUp
Hacer doble clic con el botón derecho	WM_RBUTTONDBLCLK	OnRButtonDblClk
Pulsar el botón central	WM_MBUTTONDOWN	OnMButtonDown
Soltar el botón central	WM_MBUTTONUP	OnMButtonUp
Hacer doble clic con el botón central	WM_MBUTTONDBLCLK	OnMButtonDblClk

Todas las funciones anteriores tienen dos argumentos: *nFlags* y *point*. Por ejemplo:

```
afx_msg void OnLButtonDown( UINT nFlags, CPoint point );
```

El parámetro *nFlags* indica qué botón produjo el evento y qué teclas, *Ctrl* y/o *Mayús* (*Shift*), se pulsaron. Este parámetro es una combinación de los siguientes valores:

- ✓ MK_CONTROL La tecla *Ctrl* se está pulsando.
- ✓ MK_LBUTTON El botón izquierdo del ratón se está pulsando.
- ✓ MK_MBUTTON El botón central del ratón (si lo hay) se está pulsando.
- ✓ MK_RBUTTON El botón derecho del ratón se está pulsando.
- ✓ MK_SHIFT La tecla *Mayús* (*Shift*) se está pulsando.

El parámetro *point* es un objeto **CPoint** que almacena los valores *x* e *y* correspondientes a las coordenadas del cursor del ratón relativas a la esquina superior izquierda del objeto sobre el que se produce el evento.

```
xPos = point.x;        // posición horizontal del cursor
yPos = point.y;        // posición vertical del cursor
```

Ejemplo:

```
void CMiDialogo::OnLButtonDown(UINT nFlags, CPoint point)
{
  if ( nFlags == (MK_LBUTTON | MK_CONTROL))
    AfxMessageBox("Botón izquierdo del ratón + Ctrl");

  CDialog::OnLButtonDown(nFlags, point);
}
```

3.5.15 Cambiar el cursor del ratón

La función **OnSetCursor** controla la configuración del cursor y da respuesta al mensaje **WM_SETCURSOR**. Este mensaje es enviado cuando el ratón se mueve y es respondido por una ventana antes de ejecutar cualquier otro proceso. Ejemplo:

```
BOOL CMiDialogo::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
  SetCursor(AfxGetApp()->LoadStandardCursor(
    nHitTest == HTCLIENT ? IDC_SIZENS : IDC_ARROW));

  return true;
}
```

El código anterior hace que el cursor tome forma de cruz cuando se desplace sobre el área de cliente y forma normal (una flecha) cuando se arrastre por cualquier otro lugar.

El parámetro *pWnd* de la función **OnSetCursor** especifica un puntero a la ventana que contiene el cursor; *nHitTest* determina el código de área (por ejemplo, si es igual a *HTCLIENT* se refiere al área de cliente) y *message* especifica el número de mensaje del ratón.

La función **AfxGetApp** retorna un puntero de tipo **CWinApp** que representa a la aplicación y con él se accede a la función **LoadStandardCursor** que carga un cursor predefinido de Windows por medio del parámetro *lpszCursorName* que se le pasa:

```
HCURSOR LoadStandardCursor( LPCTSTR lpszCursorName ) const;
```

Para mayor información, consulte la ayuda para ver los valores que puede tomar el parámetro *lpszCursorName*.

3.6 UNIDAD IV: VARIABLES DE CATEGORÍA, FUENTES, COLOR, CAJAS DE DIÁLOGO ESTÁNDAR Y MENÚS

Título del tema:

Variables de categoría, fuentes, color, cajas de diálogo estándar y menús

Objetivos:

- Entender el concepto de variable asociada a un control, cómo trabajar con distintos tipos de fuentes, establecer color, saber utilizar la cajas de diálogo estándar y crear aplicaciones con menús

Contenido:

- Variables de categoría “Control”
- Fuentes
- Color
- Cajas de diálogo estándar
- Menús

Duración:

- 6 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

3.6 UNIDAD IV: VARIABLES DE CATEGORÍA, FUENTES, COLOR, CAJAS DE DIÁLOGO ESTÁNDAR Y MENÚS

3.6.1 Variables de categoría "Control"

El asistente de Visual C++ "***ClassWizard***" nos permite vincular un control de un formulario o de una caja de diálogo con una variable de categoría "***Control***". Una variable de esta categoría da acceso al propio control. La clase del control será la elegida de la lista de tipos de variables; por ejemplo, **CEdit** o **CButton**.

Cuando se define una variable de la clase del control se reemplaza el procedimiento de ventana del control por el de la clase de la variable (en este caso la clase es la misma que la del control). De esta forma podemos utilizar la variable como si fuera el control (accediendo a las funciones de la clase). Por ejemplo:

```
class CRelojView : public CFormView
{
  // ...
  CEdit m_ctrlHora;
  // ...
  CFont m_FuenteHora;
  // ...
};

void CRelojView::OnInitialUpdate()
{
  // ...
  m_ctrlHora.SetFont(&m_FuenteHora);
  // ...
}
```

En el ejemplo anterior podemos ver la declaración de la variable ***m_ctrlHora*** de categoría "*Control*" y de tipo *CEdit* asociada a una caja de texto (esta caja pertenece al formulario) la cual fue añadida usando *ClassWizard*. Luego, al

iniciar la vista, establecemos la fuente que tendrá esa caja de texto desde la variable a través del método *SetFont* de la clase *CEdit* y utilizando la variable ***m_FuenteHora*** de tipo *CFont* (esta clase la veremos a continuación).

3.6.2 Establecer una fuente

La GDI (*Graphics Device Interface*) de Windows emplea la funcionalidad de la clase *CFont* (derivada de *CGdiObject*) para manipular las fuentes de caracteres. Para establecer una fuente, debemos seguir los pasos siguientes:

- ✓ Crear un objeto de tipo *CFont.*
- ✓ Establecer las características de la fuente. Esto se logra usando las funciones *CreateFont* ó *CreateFontIndirect* (para esta última función hay que emplear una estructura de tipo ***LOGFONT*** que define las características de un tipo de letra).
- ✓ Aplicar la fuente. Esto se logra llamando a la función ***CWnd::SetFont***. Por ejemplo, si queremos aplicar esa fuente a un control, hay que utilizar la función miembro ***SetFont*** heredada por ***CEdit*** de la clase ***CWnd***. Ejemplo:

```
void CRelojView::OnInitialUpdate()
{
  // ...
  LOGFONT lf;

  memset(&lf, 0, sizeof(LOGFONT));
  strcpy(lf.lfFaceName, "Arial");
  lf.lfHeight = -UnidadesLogicas(16);
  m_FuenteHora.CreateFontIndirect(&lf);
  m_ctrlHora.SetFont(&m_FuenteHora);
  // ...
}
```

La función ***CreateFont*** tiene 14 parámetros que se corresponden directamente con los miembros de una estructura de tipo ***LOGFONT*** (esta estructura tiene una serie de campos que definen las características de un tipo de letra). Por esta razón, es más cómodo crear un tipo de letra utilizando la estructura ***LOGFONT*** y la función ***CreateFontIndirect***, tal como se muestra en el ejemplo anterior.

Establecer las características del tipo de letra significa asignar los valores deseados a los miembros de la estructura ***LOGFONT*** que son:

```
typedef struct tagLOGFONT { // lf
  LONG lfHeight;          // altura en unidades virtuales
  LONG lfWidth;           // anchura en unidades virtuales
  LONG lfEscapement;      // ángulo de cada línea de texto
  LONG lfOrientation;     // ángulo de la línea base de cada caracter
  LONG lfWeight;          // grado de negrita (400 normal, 700 negrita)
  BYTE lfItalic;          // cursiva si es distinto de 0
  BYTE lfUnderline;       // subrayado si es distinto de 0
```

```
  BYTE lfStrikeOut;           // tachado si es distinto de 0
  BYTE lfCharSet;             // juego de caracteres (ANSI, UNICODE, etc.)
  BYTE lfOutPrecision;        // precisión de lo que se visualiza
  BYTE lfClipPrecision;       // precisión (región de corte)
  BYTE lfQuality;             // parecido al tipo de letra real
  BYTE lfPitchAndFamily;      // ancho y familia del tipo de letra
  TCHAR lfFaceName[LF_FACESIZE]; // nombre del tipo de imprenta
} LOGFONT;
```

No hay porqué iniciar todos y cada uno de los miembros de la estructura ***LOGFONT***. Normalmente es suficiente con definir la altura y el nombre del tipo de letra que se quiera usar; los otros campos tomarán valores por omisión.

3.6.3 Altura de la fuente

Para calcular la altura que tendrá la fuente que queramos utilizar deberemos especificarla en unidades lógicas, cuando normalmente la especificamos en puntos. Para ello debemos seguir los pasos siguientes:

- ✓ Obtener el contexto de dispositivo del área de trabajo (ventana de la aplicación).
- ✓ Obtener información del contexto de dispositivo.
- ✓ Calcular cuántos píxeles son los puntos en los que se ha especificado la altura de la fuente. Para esto hay que saber que una pulgada son **72 puntos**.
- ✓ Convertir los píxeles anteriores a unidades lógicas. Para ello usaremos la función ***DPtoLP*** a través del contexto de dispositivo.
- ✓ Asignar la altura de la fuente en unidades lógicas y con signo menos, al miembro ***lfHeight*** de la estructura ***LOGFONT***. El signo menos es porque para un valor negativo, Windows toma el valor absoluto como la altura deseada para la fuente en unidades lógicas.

Ejemplo:

```
int CRelojView::UnidadesLogicas(int nPuntosAltFuente)
{
  CClientDC dc(this);

  POINT pt;

  pt.x = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSX) / 72;
  pt.y = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSY) / 72;

  dc.DPtoLP(&pt);

  return pt.y;
}
```

3.6.4 Color

La ***GDI*** (***Graphics Device Interface***) de Windows emplea la funcionalidad de la clase ***CBrush***, derivada de ***CGdiObject*** para manipular pinceles. Los pinceles permiten rellenar los interiores de las figuras, tales como rectángulos y elipses, o establecer el color de fondo de los controles, tales como cajas de texto y etiquetas.

Para establecer o utilizar un pincel tenemos que:

- ✓ Crear un objeto ***CBrush***
- ✓ Iniciarlo con el color deseado a través de un constructor de la clase o por medio de la función ***CreateSolidBrush***.

Ejemplo:

```
CBrush m_BrochaFondoHora;
m_BrochaFondoHora.CreateSolidBrush(RGB(255, 255, 0));
```

La clase ***CPen*** se usa para poner el color de primer plano, es decir, define el borde de una figura cerrada. La línea puede ser sólida o con estilo. Por defecto la línea es de un pincel de ancho y de color negro. Para definir un borde creamos un objeto de la clase ***CPen***:

```
CPen lapiz (PS_SOLID, 3, RGB(0,0,255));
```

A continuación se muestran los estilos de pincel en una tabla. Estos estilos solo van a funcionar con un ancho de línea de un pixel, en cuanto cambiemos solo funciona con PS_SOLID.

PS_SOLID	Dibuja sólido.
PS_DASH	Líneas discontinuas.
PS_DOT	Dibuja puntos.
PS_DASHDOT	Dibuja (.-.-.-.-.-.)
PS_DASHDOTDOT	Dibuja (-..-..-..-..)

Cada vez que un control está a punto de repintarse, envía un mensaje **WM_CTLCOLOR** al procedimiento de ventana de su ventana padre. Esto hace que se llame a la función controladora del mensaje **OnCtlColor** heredada de la clase **CWnd** la cual nos permitirá escribir el código necesario para alterar los colores que el procedimiento de ventana utiliza para pintar los controles.

La macro **RGB** permite especificar un color cualquiera. Tiene 3 parámetros que especifican el nivel de color rojo, verde y azul. Esta función devuelve un entero

de cuatro bytes (32 bits) de tipo **COLORREF** que representa el color **RGB** especificado:

```
COLORREF RGB( BYTE byRed, BYTE byGreen, BYTE byBlue );
```

El ejemplo siguiente muestra cómo establecer el color de fondo de una caja de texto cuyo identificador es **IDC_HORA**; para ello se utiliza la brocha **m_BrochaFondoHora**. Observe que esto se lleva a cabo en la función **OnCtlColor**:

```
HBRUSH CRelojView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
  HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

  if (pWnd->GetDlgCtrlID()==IDC_HORA)
  {
    return (HBRUSH)m_BrochaFondoHora.GetSafeHandle();
  }

  return hbr;
}
```

Este otro ejemplo muestra cómo establecer el color de fondo y primer plano para el texto de una caja de texto cuyo identificador es ***IDC_HORA*** las variables *m_rgbFondoHora* y *m_rgbTextoHora* son de tipo **COLORREF**:

```
HBRUSH CRelojView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
  HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

  if (pWnd->GetDlgCtrlID()==IDC_HORA)
  {
    pDC->SetBkColor(m_rgbFondoHora);
    pDC->SetTextColor(m_rgbTextoHora);
  }

  return hbr;
}
```

El ejemplo completo (color de fondo y del texto) se muestra a continuación:

```
HBRUSH CRelojView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
  HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

  if (pWnd->GetDlgCtrlID()==IDC_HORA)
  {
    pDC->SetBkColor(m_rgbFondoHora);
    pDC->SetTextColor(m_rgbTextoHora);

    return (HBRUSH)m_BrochaFondoHora.GetSafeHandle();
  }

  return hbr;
}
```

3.6.5 Cajas de diálogo estándar

Existen una serie de clases derivadas de **CDialog** que permiten visualizar las cajas de diálogo más comúnmente empleadas. Éstas son:

- ✓ **CFileDialog** Abrir ó guardar un fichero

- ✓ **CColorDialog** Seleccionar un color
- ✓ **CFontDialog** Seleccionar un tipo de letra
- ✓ **CFindReplaceDialog** Sustituir una cadena por otra
- ✓ **CPageSetupDialog** Poner las características de la página a imprimir
- ✓ **CPrintDialog** Imprimir un documento

Para crear una caja de diálogo estándar existen dos formas:

- ✓ **Forma directa:** crear un objeto de alguna de las clases predefinidas en la biblioteca MFC.
- ✓ **Forma indirecta:** crear un objeto de una clase derivada de alguna de las clases predefinidas en la biblioteca MFC.

En ambas formas, una vez creado el objeto, para visualizar la caja de diálogo hay que invocar a la función miembro **DoModal** a través del objeto.

La diferencia de usar la forma directa de la indirecta es que en la forma indirecta al ser una clase derivada podemos añadir nueva funcionalidad a esta clase derivada (datos o funciones miembro) con el fin de personalizar la caja de diálogo.

A continuación se muestra un ejemplo de cómo funciona la forma directa:

```
void CAboutDlg::OnBtAbrir()
{
  CFileDialog dlgAbrir(true);

  // Modificar aquí el objeto dlgAbrir para que
  // muestre inicialmente ciertos datos
  // ...

  if (dlgAbrir.DoModal()==IDOK)
  {
    // .Extraer aquí del objeto dlgAbrir los
    // datos que haya aceptado finalmente el
    // usuario
    // ...
  }
}
```

Específicamente, veremos cómo aplicar esta forma para añadir un diálogo **Color** y un diálogo **Fuente**.

Para añadir un diálogo *Color* debemos seguir los pasos siguientes:

- ✓ Crear un objeto de tipo **CColorDialog**. Como parámetro opcional podemos establecer un color por omisión (al llamar el constructor de la clase, este es un valor de tipo **COLORREF**).
- ✓ Opcionalmente podemos modificar los valores por omisión (es decir, personalizar la caja de diálogo). Esto se hace a través de una estructura

m_cc de tipo **CHOOSECOLOR** que agrupa los parámetros básicos de la caja de diálogo. Esta estructura es un miembro de la clase **CColorDialog**.

- ✓ Para visualizar la caja de diálogo, invoque a la función miembro **DoModal** a través del objeto.

- ✓ Si la función **DoModal** devuelve **IDOK**, puede usar la función miembro **CColorDialog::GetColor** para obtener el color elegido por el usuario.

Ejemplo:

```
void CRelojView::OnBtcolor()
{
  CColorDialog dlgColor(m_rgbFondoHora);

  if (dlgColor.DoModal() == IDOK)
  {
    m_rgbFondoHora = dlgColor.GetColor();
    m_BrochaFondoHora.DeleteObject();
    m_BrochaFondoHora.CreateSolidBrush(m_rgbFondoHora);
    Invalidate();
  }
}
```

Para añadir un diálogo *Fuente* debemos seguir los pasos siguientes:

- ✓ Crear un objeto de tipo **CFontDialog**.

- ✓ Opcionalmente podemos modificar los valores por omisión (es decir, personalizar la caja de diálogo). Esto se hace a través de una estructura **m_cf** de tipo **CHOOSEFONT** que agrupa los parámetros básicos de la caja de diálogo. Esta estructura es un miembro de la clase **CFontDialog**.

- ✓ Para visualizar la caja de diálogo, invoque a la función miembro **DoModal** a través del objeto.

- ✓ Si la función **DoModal** devuelve **IDOK**, puede usar las funciones miembro de la clase **CFontDialog** para obtener la información introducida por el usuario. Entre estas tenemos: **IsBold**, **IsItalic**, **IsStrikeOut**, **IsUnderline**, **GetFaceName**, **GetStyleName**, **GetSize**, **GetColor** y **GetCurrentFont**.

- ✓ A partir de la información solicitada por el usuario, crear y establecer la fuente. Esto se logra creando un objeto de tipo **CFont** y llamando a la función **SetFont**.

Ejemplo:

```
void CRelojView::OnBtfuente()
{
  CFontDialog dlgFuente;

  m_FuenteHora.GetLogFont(dlgFuente.m_cf.lpLogFont);
  dlgFuente.m_cf.Flags |= CF_INITTOLOGFONTSTRUCT;
  dlgFuente.m_cf.rgbColors = m_rgbTextoHora;

  if (dlgFuente.DoModal() == IDOK)
  {
    // Crear la fuente con lpLogFont y establecerla
  }
}
```

3.6.6 Menús

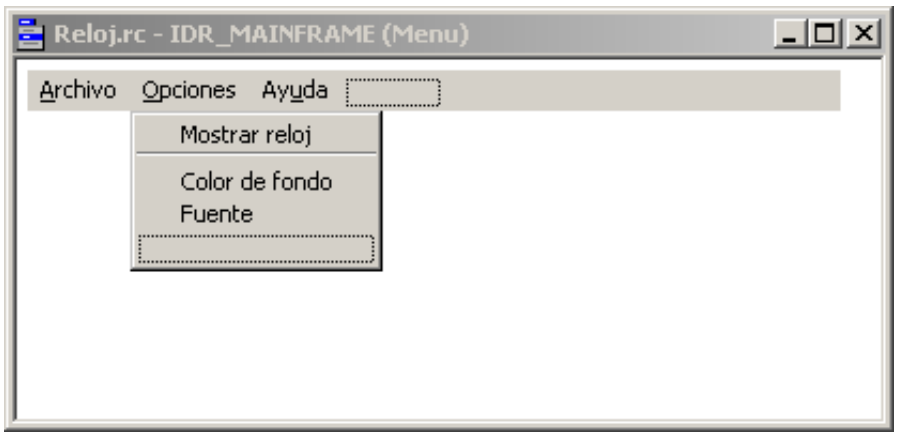
Un menú es una forma de proveer al usuario de un conjunto de órdenes, lógicamente relacionadas, agrupadas baja un mismo título. Un menú forma parte de una barra de menús. Se trata de una lista desplegable que incluye:

- ✓ Órdenes
- ✓ Submenús
- ✓ Separadores

Para trabajar con un menú debemos seguir los pasos siguientes:

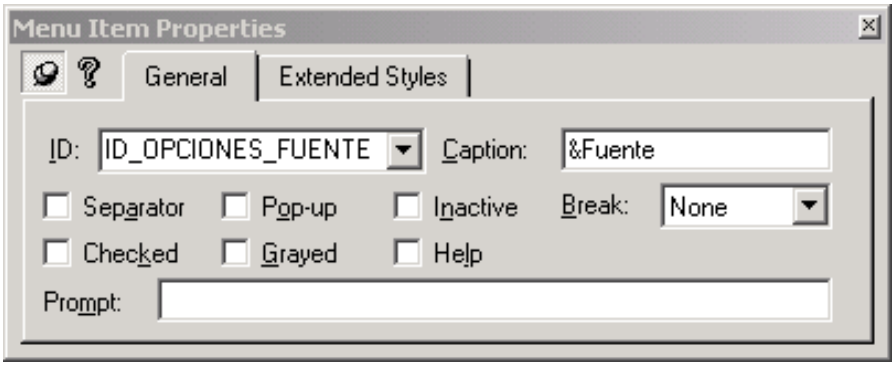
- ✓ Crear el menú. Esto se hace con el editor de menús.
- ✓ Establecer las propiedades de cada uno de los elementos del menú.
- ✓ Programar las acciones o eventos que se deben producir al usar el menú. Esto se hace con ayuda de **ClassWizard**.

La figura siguiente muestra un ejemplo de cómo crear un menú **"Opciones"** desde el editor de menús:



Para crear un submenú tiene que poner a valor trae la propiedad *Pop-up* del elemento del menú. Para insertar un separador hay que activar la propiedad *Separador*.

La figura siguiente muestra cómo establecer las propiedades:



Luego debemos añadir la funcionalidad a la que debe responder el menú. Para ello debemos usar **ClassWizard**.

Para definir cómo debe responder cada orden de un menú al evento *clic*, hay que escribir una función para cada una de ellas. Las órdenes de un menú sólo responden al evento *clic*; cuando un usuario selecciona una orden de un menú se envía un mensaje **WM_COMMAND** (macro **ON_COMMAND**).

Cada vez que se despliega un menú, Windows envía los mensajes de iniciación **WM_INITMENU** y **WM_INITMENUPOPUP** (manipulados a través de la macro **ON_UPDATE_COMMAND_UI**). Estos métodos nos ayudan a actualizar el estado de las órdenes del menú (activadas o desactivadas).

Los controladores de mensajes pueden ser ejecutados desde:

- ✓ La ventana marco
- ✓ La vista

Al igual que los botones de pulsación, se genera un mensaje del tipo **WM_COMMAND** cuando seleccionamos un elemento de un menú.

En cambio, el controlador de los mensajes **WM_INIT**... se ejecutará justo antes de mostrar el contenido del menú. Este controlador tiene un parámetro de tipo **CCmdUI**. Esta es una clase que define varios datos miembro relativos al elemento del menú susceptible de ser actualizado, entre ellos el identificador **m_nID** del elemento. Otras funciones de esta clase son: **Enable**, **SetCheck**, **SetRadio**, **SetText**, ... (Para mayor información consultar la ayuda de MSDN). Los botones de una barra de herramientas utilizan los mismos controladores.

Ejemplo:

```
void CRelojView::OnUpdateOpcionesMostrarreloj(CCmdUI* pCmdUI)
{
  pCmdUI->SetCheck(m_pDlgRelojAnalogico->IsWindowVisible());
}
```

En este ejemplo, ponemos una marca al elemento "**Mostrar Reloj**" del menú "**Opciones**" si el diálogo apuntado por el puntero **m_pDlgRelojAnalogico** se encuentra visible.

3.7 UNIDAD V: EL TECLADO Y LA SUBCLASIFICACIÓN

Título del tema:

El teclado y la subclasificación

Objetivos:

- Entender el concepto de subclasificación y aprender cómo manejar los eventos asociados al teclado

Contenido:

- Subclasificación
- Interceptar la tecla pulsada
- Controles

Duración:

- 6 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

3.7 UNIDAD V: EL TECLADO Y LA SUBCLASIFICACIÓN

3.7.1 Subclasificación

La subclasificación es una técnica que permite a una aplicación interceptar y procesar mensajes enviados o pasados a una ventana particular antes de que la ventana principal tenga una oportunidad de procesarlos.

Específicamente, los pasos para subclasificar una ventana son:

- ✓ Definir una clase derivada de la clase del control o de la ventana.
- ✓ Definir un objeto de esa clase derivada.
- ✓ Subclasificar el control o la ventana con el objeto definido: esto quiere decir que el procedimiento de ventana del control será reemplazado por el del objeto.

Ejemplo:

Supongamos que necesitas una caja de texto que acepte sólo cadenas en hexadecimal. Si usas un control de edición simple, tienes que decir algo cuando el usuario transcribe algo diferente a números hexadecimales dentro de la caja de texto, es decir, si el usuario escribe "zb+q" dentro de la caja de texto, no puedes hacer nada con ello, excepto rechazar toda la cadena de texto. Esto al menos carece de profesionalidad. En esencia, necesitas la habilidad de examinar cada carácter que el usuario escribió dentro de la caja de texto en el momento que él lo transcribió.

Ahora examinaremos cómo hacerlo. Cuando el usuario escribe algo dentro de la caja de texto, Windows envía el mensaje **WM_CHAR** al procedimiento de ventana del control de edición. Este procedimiento de ventana reside dentro de Windows así que no podemos modificarlo. Pero podemos **redirigir el flujo de mensajes a nuestro propio procedimiento de ventana**. Así que nuestro procedimiento de ventana obtendrá primero un impacto [shot] de cualquier mensaje de Windows antes de que sea enviado al control de edición. Si nuestro procedimiento de ventana resuelve actuar sobre el mensaje, puede aprovechar ahora para hacerlo. Pero si no desea manejar el mensaje, lo pasa al

procedimiento de ventana principal. De esta manera, nuestro procedimiento de ventana se inserta dentro de Windows y del control de edición. Veamos el siguiente flujo:

Antes de la Subclasificación
Windows ==> procedimiento de ventana del control de edición

Después de la Subclasificación
Windows ==> nuestro procedimiento de ventana ----> procedimiento de ventana del control de edición

Aplicando los pasos de la subclasificación a este ejemplo tenemos:

✓ Definir una clase derivada de la clase del control o de la ventana:

```
class CEditHex : public CEdit
{
// Construction
public:
 CEditHex();
 UINT m_IdCajaTexto;

// Attributes
public:

// Operations
public:

 BOOL SubclaseEdit(UINT, CWnd *);

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

// Implementation
public:
 virtual ~CEditHex();

 // Generated message map functions
protected:
 //{{AFX_MSG(CEditHex)
 afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
 //}}AFX_MSG

 DECLARE_MESSAGE_MAP()
};
```

✓ Definir un objeto de esa clase derivada:

```
class CSubclasifiView : public CFormView
{
...
CEditHex m_objhex;
```

```
...
}
```

- ✓ Subclasificar el control o la ventana con el objeto definido: esto quiere decir que el procedimiento de ventana del control será reemplazado por el del objeto. Para hacer esto, haremos uso de la función **SubclassDlgItem:**

```
BOOL CEditHex::SubclaseEdit(UINT nID, CWnd *pWndPadre)
{
 this->m_IdCajaTexto = nID;
 return (this->SubclassDlgItem(nID,pWndPadre));
}
```

```
void CSubclasifiView::OnInitialUpdate()
{
 CFormView::OnInitialUpdate();
 GetParentFrame()->RecalcLayout();
 ResizeParentToFit();

 this->m_objhex.SubclaseEdit(IDC_CAJA1, this);
}
```

Para hacer la validación (permitir en la entrada solamente dígitos hexadecimales 0-9 A-F) debemos añadir el método **OnChar** que responde al mensaje **WM_CHAR** para nuestra clase derivada de **CEdit**:

```
void CEditHex::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
 bool digito = false, car = false;

 if (nChar < '0' || nChar > '9')
      digito = true;
 if (nChar < 'A' || nChar > 'F')
      car = true;
 if (digito && car)
      {
      MessageBeep(MB_OK);
      ((CSubclasifiView *)GetParent())->m_mensaje =
                          "Error: Caracter incorrecto...";
      ((CSubclasifiView *)GetParent())->UpdateData(FALSE);
      }
 else
      CEdit::OnChar(nChar, nRepCnt, nFlags);
}
```

Otro ejemplo:

```
class CEditHora : public CEdit
{
  // ...
};

CEditHora m_ctrlHoraAlarma;

BEGIN_MESSAGE_MAP(CEditHora, CEdit)
  //{{AFX_MSG_MAP(CEditHora)
  ON_WM_CHAR()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
void CRelojView::DoDataExchange(CDataExchange* pDX)
{
  CFormView::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CRelojView)
  DDX_Control(pDX, IDC_HORA_ALARMA, m_ctrlHoraAlarma);
  DDX_Control(pDX, IDC_HORA, m_ctrlHora);
  //}}AFX_DATA_MAP
}
```

3.7.2 Interceptar la tecla pulsada

Cada vez que el usuario pulsa una tecla que produce un carácter, Windows coloca en la cola de mensajes del objeto que tiene el foco tres mensajes:

WM_KEYDOWN	**CWnd::OnKeyDown**
WM_CHAR	**CWnd::OnChar**
WM_KEYUP	**CWnd::OnKeyUp**

El mensaje **WM_CHAR** se genera solamente cuando se introduce un carácter ASCII (se excluyen las teclas de función F1 a F12, teclas de movimiento o la tecla *Supr*).

Las funciones controladoras de estos mensajes utilizan los argumentos **nChar**, **nRepCnt** y **nFlags** que contienen, respectivamente, el código de la tecla virtual correspondiente a la tecla pulsada, el número de repeticiones que se producen cuando el usuario mantiene la tecla pulsada, y el estado de la tecla:

```
void CEditHex::OnXXX(UINT nChar, UINT nRepCnt, UINT nFlags)
{
...
        CEdit::OnXXX(nChar, nRepCnt, nFlags);
}
```

El código dado por **nChar** no hace distinción entre mayúsculas y minúsculas, excepto para **OnChar**. Para hacer esta distinción tenemos que ver si el argumento **nChar** es **VK_SHIFT** primero y un carácter entre la A y la Z después.

El código **VK_RETURN** (tecla **Enter**) sólo se puede interceptar en **OnKeyUp**.

```
void CEditHora::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
  if ((nChar < '0' || nChar > '9') &&
       nChar != ':' && nChar != VK_BACK)
  {
    MessageBeep (0xFFFFFFFF);
  }
  else
    CEdit::OnChar(nChar, nRepCnt, nFlags);
}
```

Para que un control subclasificado pueda procesar todas las entradas del teclado este debe responder al mensaje **WM_GETDLGCODE**. Esto quiere decir que debemos implementar la función controladora **OnGetDlgCode**. Esta función devuelve el valor **DLGC_WANTALLKEYS**:

```
UINT CEditHora::OnGetDlgCode()
{
  return DLGC_WANTALLKEYS;
}
```

3.7.3 Controles

Una ***casilla de verificación*** es un control que indica si una opción particular está activada o desactivada. Cuando hay varias, se pueden seleccionar varias (cada una tiene su propio **ID**). La funcionalidad para manipular este tipo de controles es proporcionada por la clase **CButton**.

Si hacemos clic sobre una casilla de verificación, ésta envía un mensaje de tipo **BN_CLICKED** a la caja de diálogo. Para saber si una determinada opción está seleccionada, hay que verificar el valor devuelto por la función **GetCheck** de la clase **CButton**. Este valor puede ser:

0 La caja está vacía
1 La caja aparece con una señal
2 La caja aparece en gris

Existen 3 formas para obtener la información sobre el estado de una casilla:

- ✓ Con un puntero a **CButton**.
- ✓ Con una variable de categoría "**Control**".
- ✓ Con una variable de categoría valor, tipo **bool**.

A continuación, se muestran estas 3 formas:

```
CButton *p = (CButton *)GetDlgItem(IDC_CHECK1);
if (p->GetCheck())
  AfxMessageBox("IDC_CHECK1");

if (m_ctrlIDC_CHECK1.GetCheck())
  AfxMessageBox("IDC_CHECK1");

UpdateData(true);
if (m_bIDC_CHECK1)
  AfxMessageBox("IDC_CHECK1");
```

Un ***botón de opción*** es un control que indica si una determinada opción está activada o desactivada. Cuando hay varios forman un grupo, del que sólo se puede seleccionar uno. La funcionalidad para manipular este tipo de controles es proporcionada por la clase **CButton**.

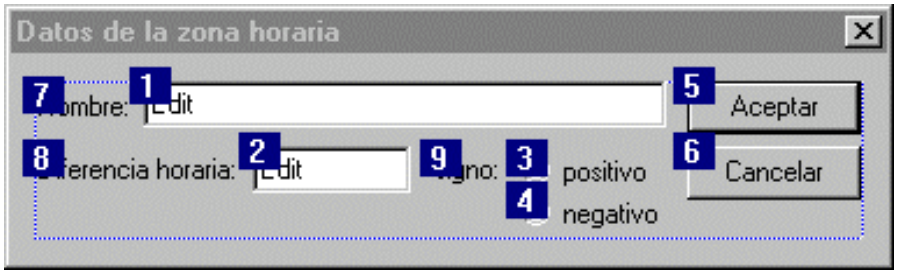
Cuando se hace clic sobre un botón de opción, Windows envía el mensaje de notificación **BN_CLICKED** a su ventana padre.

Para saber si una determinada opción está seleccionada, hay que verificar el valor devuelto por la función **GetCheck** de la clase **CButton**. Este valor puede ser:

0 el círculo aparece vacío
1 el círculo aparece seleccionado

La propiedad **Group** especifica cuál es el primer control de un grupo de controles. Un grupo está formado por los controles que hay, siguiendo el orden *Tab*, entre el primer control que tiene la propiedad **Group** a **true** hasta el siguiente control que tenga la propiedad **Group** también a **true**:

El orden *Tab* especifica el orden en el que serán enfocados los controles utilizando la tecla *Tab*. Para modificar este orden, desde el editor de recursos, debemos presionar **Ctrl+d**:



De forma análoga a las casillas de verificación, para saber si un botón está o no seleccionado, podemos aplicar las 3 formas siguientes:

- ✓ Con punteros a la clase **CButton**: CButton *pBtOp1,..., *pBtOpn
- ✓ Usando variables de categoría control: una por cada botón. Inconveniente: *ClassWizard* sólo muestra los botones con la propiedad **Group** = *true*.

- ✓ Usando variables de categoría valor, tipo int asociada con el botón con la propiedad **Group** = *true*. Toma los valores –1, 0, 1, ... dependiendo del botón seleccionado (-1 = no selección).

Ejemplo:

```
UpdateData(true);

if (m_nBotonOp == 0)
  AfxMessageBox("IDC_RADIO1");
if (m_nBotonOp == 1)
  AfxMessageBox("IDC_RADIO2");
if (m_nBotonOp == 2)
  AfxMessageBox("IDC_RADIO3");
```

En este ejemplo, habrá que iniciar la variable ***m_nBotonOp*** en el constructor de su clase (por omisión vale -1) para que quede seleccionado un botón por omisión.

Un ***marco*** es un rectángulo utilizado para agrupar controles relacionados entre sí con el simple fin de realzar el aspecto de una caja de diálogo. Un marco ni procesa mensajes del ratón, ni del teclado, ni manda mensajes **WM_COMMAND** a su ventana padre.

3.8 UNIDAD VI: TRABAJANDO CON MENÚS

Título del tema:

Trabajando con menús

Objetivos:

- Aprender a desarrollar aplicaciones que hagan uso de menús para mejorar su aspecto y funcionalidad

Contenido:

- Menús
- Clase CMenu
- Operaciones con menús
- Menús dinámicos
- Aceleradores
- Menús flotantes

Duración:

- 6 horas

Bibliografía básica:

- Ceballos Sierra, Francisco Javier. Visual C++, aplicaciones para Win32. Editorial RAMA.

3.8 UNIDAD VI: TRABAJANDO CON MENÚS

3.8.1 Menús

Cuando trabajamos con menús debemos de tener en cuenta lo siguiente:

- ✓ La ventana propietaria de la barra de menús es la ventana marco.
- ✓ Un clic en un elemento de un menú envía un mensaje WM_COMMAND.
- ✓ La ventana marco da preferencia a su ventana activa para que procese el mensaje.

Para que la vista pueda acceder a la barra de menús debe:

- ✓ Obtener un puntero a su ventana padre que es la propietaria de dicha barra. Para ello, hay que usar la función **CWnd::GetParent**.
- ✓ Obtener un puntero a la barra de menús. Para ello, hay que usar la función **CWnd::GetMenu**:

```
CMenu *pMenu = GetParent()->GetMenu();
```

3.8.2 Clase CMenu

Esta clase proporciona la funcionalidad necesaria para manipular menús. Entre sus funciones miembro tenemos:

- ✓ **GetSubMenu(n)**: obtiene un puntero al menú especificado de la barra. El primer menú es el 0. La variable n toma los valores 0,1,2,... según la cantidad de elementos que tenga el menú:

CMenu* GetSubMenu(int nPos) const;

- ✓ **AppendMenu**: añade un elemento nuevo al final del menú:

BOOL AppendMenu (UINT nFlags, UINT nIDNewItem = 0, LPCTSTR lpszNewItem = NULL);

- ✓ **RemoveMenu**: quita un elemento existente de un menú:

BOOL RemoveMenu(UINT nPosition, UINT nFlags);

- ✓ **InsertMenu**: inserta un elemento de un menú en una posición especificada y mueve los otros elementos del menú por debajo de este nuevo elemento:

BOOL InsertMenu(UINT nPosition, UINT nFlags, UINT nIDNewItem = 0, LPCTSTR lpszNewItem = NULL);

- ✓ **ModifyMenu**: permite modificar un elemento existente de un menú:

BOOL ModifyMenu(UINT nPosition, UINT nFlags, UINT nIDNewItem = 0, LPCTSTR lpszNewItem = NULL);

- ✓ **EnableMenuItem**: habilita, inhabilita sin poner en gris o bien inhabilita poniendo en gris, un elemento de un menú:

UINT EnableMenuItem(UINT nIDEnableItem, UINT nEnable);

3.8.3 Operaciones con menús

Ejemplo de la función ***AppendMenu***:

```
pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                  ID_ZONA_ELIMINAR,
                  "&Eliminar");

pMenu->AppendMenu(MF_SEPARATOR);

pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                  ID_BASE_ZONA + m_nZonas,
                  aszNombre);
```

Ejemplo de la función ***RemoveMenu***:

```
pMenu->RemoveMenu(2, MF_BYPOSITION);

pMenu->RemoveMenu(ID_ZONA_ELIMINAR, MF_BYCOMMAND);
```

Ejemplo de la función ***InsertMenu***:

```
pMenu->InsertMenu(0,
                  MF_BYPOSITION|MF_ENABLED|MF_STRING,
                  ID_ZONA_ANADIR,
                  "&Añadir");
```

Ejemplo de la función ***ModifyMenu***:

```
pMenu->ModifyMenu(ID_ZONA_ANADIR,
                  MF_BYCOMMAND,
                  ID_ZONA_ANADIR,
                  "&No añadir");

pMenu->ModifyMenu(0,
                  MF_BYPOSITION,
                  ID_ZONA_ANADIR,
                  "&No añadir");
```

Ejemplo de la función ***EnableMenuItem***:

```
pMenu->EnableMenuItem(0, MF_BYPOSITION|
                         (m_bDespertadorActivado?
                          MF_GRAYED:MF_ENABLED));
```

Otra alternativa para habilitar o deshabilitar los elementos de un menú es usando la función **CCmdUI::Enable** en el controlador de los mensajes: **WM_INITMENU** y **WM_INITMENUPOPUP** (macro **ON_UPDATE_COMMAND_UI**).

Recordemos que la clase **CCmdUI** tiene las funciones: **Enable**, **SetCheck**, **SetRadio**, **SetText**, …

3.8.4 Menús dinámicos

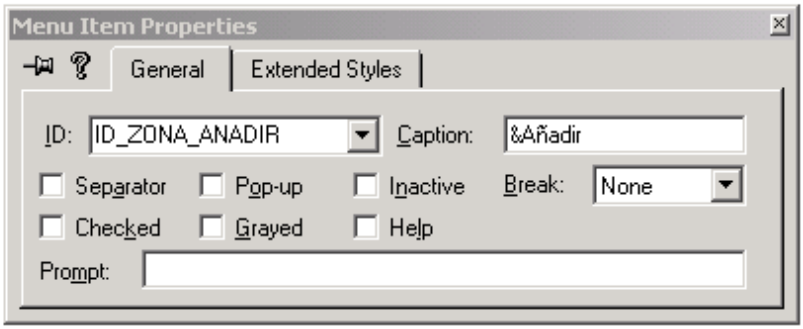
También podemos trabajar con menús dinámicos, es decir, podemos añadir elementos a un menú en tiempo de ejecución. Añadir un elemento a un menú implica tener lo siguiente:

- ✓ Un identificador
- ✓ Un controlador

Cada elemento que va a componer un menú tiene que tener un identificador definido previamente. La razón es que una vez añadido un elemento, no podríamos añadir una nueva función para él; mientras que con un identificador es posible asociar una función común a todos los posibles elementos nuevos.

Además, un menú dinámico tiene como base un menú estático con al menos una orden. Durante el diseño hay que crear los identificadores y esto exige que al menos definamos uno que sirva como referencia para poder asociar al procedimiento común.

Como ejemplo, la figura siguiente muestra un menú **"Zona"** que tiene un único elemento **"Añadir"**. A partir de este menú, se irán añadiendo nuevas órdenes:



A continuación se muestra la función **OnZonaAnadir** la cual se manda a llamar cuando hacemos clic sobre el elemento **Añadir** del menú **Zona** y permite añadir una nueva orden:

```
void CRelojView::OnZonaAnadir()
{
  CMenu *pMenu = GetParent()->GetMenu();
  pMenu = pMenu->GetSubMenu(1);

  char aszNombre[] = "Título";
  pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                    ID_ZONA_ANADIR+100 + m_nZonas,
                    aszNombre);
  m_nZonas++;
}
```

Para añadir el controlador para los elementos añadidos tendremos que:

- ✓ Añadir las entradas adecuadas al mapa de mensajes de la clase que tiene que procesar el mensaje **WM_COMMAND**.

- ✓ Añadir la declaración y la definición de la función controladora del mensaje **WM_COMMAND**.

Las macros que se pueden utilizar son las siguientes:

ON_COMMAND(nID, fm)
afx_msg void fm()

ON_COMMAND_EX(nID, fm)
afx_msg void fm(UINT nID)

ON_COMMAND_EX_RANGE(nIDi, nIDf, fm)

ON_COMMAND_RANGE(nIDi, nIDf, fm)
afx_msg void fm(UINT nID)

La macro **ON_COMMAND_EX** es utilizada en lugar de la macro **ON_COMMAND** cuando necesitamos disponer del ID del elemento seleccionado.

La macro **ON_COMMAND_EX_RANGE** se usa para simplificar varias entradas **ON_COMMAND_EX** (en el caso de que existan varios IDs asociados a un mismo controlador). En el siguiente mapa de mensajes podemos ver esta macro:

```
#define ID_BASE_ZONA (ID_ZONA_ANADIR+100)
#define NUM_MAX_ZONAS 4

IMPLEMENT_DYNCREATE(CRelojView, CFormView)

BEGIN_MESSAGE_MAP(CRelojView, CFormView)
  //{{AFX_MSG_MAP(CRelojView)
  // ...
  //}}AFX_MSG_MAP
  ON_COMMAND_EX_RANGE(ID_BASE_ZONA,
                      ID_BASE_ZONA + NUM_MAX_ZONAS - 1,
                      OnZona)
END_MESSAGE_MAP()
```

En este ejemplo, todos los identificadores que van desde **ID_BASE_ZONA** hasta **ID_BASE_ZONA+NUM_MAX_ZONAS-1** están asociados a la función **OnZona**. En las siguientes figuras podemos ver la declaración y la definición de la función **OnZona**:

```
class CRelojView : public CFormView
{
  // ...
protected:
  //{{AFX_MSG(CRelojView)
  // ...
  //}}AFX_MSG
  afx_msg void OnZona(UINT nID);

  DECLARE_MESSAGE_MAP()
};
```

```
void CRelojView::OnZona(UINT nID)
{
  CMenu * pMenu = GetParent()->GetMenu();
  pMenu = pMenu->GetSubMenu(1);
  // ...
  CString sNombre;
  pMenu->GetMenuString(nID, sNombre, MF_BYCOMMAND);

  // ...
}
```

3.8.5 Aceleradores

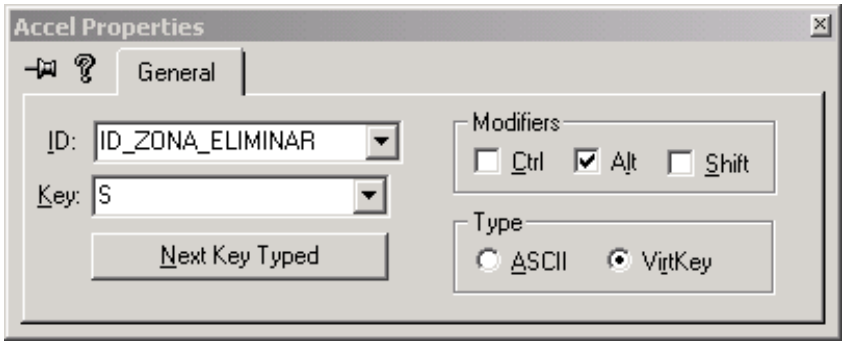
Los *aceleradores de teclado* son combinaciones de teclas utilizadas para acceso rápido a las órdenes de los menús de uso más común.

Estas combinaciones de teclas generan mensajes **WM_COMMAND**.

Se implementan por medio del *asistente de recursos* (recurso *Accelerator*).

La ventaja de utilizar aceleradores de teclado, en lugar de interceptar los mensajes **WM_CHAR**, **WM_KEYDOWN** y **WM_KEYUP** que son enviados al procedimiento de ventana que tiene el foco de entrada en ese momento, es que no hay que duplicar el código escrito para los menús.

Ejemplo:



3.8.6 Menús flotantes

Un menú desplegable flotante es visualizado sobre una ventana, independiente de la barra de menús invocando a la función **CMenu::TrackPopupMenu**:

BOOL TrackPopupMenu(UINT nFlags, int x, int y, CWnd* pWnd, LPCRECT lpRect = NULL);

donde:

nFlags define la ubicación y comportamiento del menú.

x e ***y*** especifican la posición en coordenadas de pantalla (referida a las especificaciones de nFlags).

pWnd identifica la ventana propietaria del menú.

lpRect apunta a una estructura ***RECT*** u objeto ***CRect*** que define el área donde el usuario puede hacer clic para ocultar el menú; si este parámetro es ***NULL*** el menú se oculta cuando se hace clic fuera de él.

Ejemplo:

```
void CRelojView::OnRButtonDown(UINT nFlags, CPoint point)
{
  CMenu * pMenu = GetParent()->GetMenu();
  pMenu = pMenu->GetSubMenu(1);

  ClientToScreen(&point);
  pMenu->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,
                        point.x, point.y,
                        this,
                        NULL);

  CFormView::OnRButtonDown(nFlags, point);
}
```

LABORATORIO DE LA ASIGNATURA: PROGRAMACIÓN VISUAL

Capítulo 4: LABORATORIO DE PROGRAMACIÓN VISUAL

4.1 INTRODUCCIÓN

En el capítulo 3 de este documento se presentó la planificación temporal del contenido teórico y práctico de la asignatura Programación Visual. Se planteó que para la realización de las prácticas se cuenta con un total de 15 semanas para un total de 30 horas (cada período semanal es de 2 horas). Debido a que se cuenta con 25 computadoras en el laboratorio donde se desarrollarán las prácticas y a que la cantidad de alumnos es aproximadamente de 50, se harán 2 grupos de 25 estudiantes.

Esto significa que durante una semana habrán dos sesiones de laboratorio (una para el grupo A y otra para el grupo B) para que todos puedan desarrollar la misma práctica propuesta para esa semana. Cabe aclarar que esto se hace así porque el mismo profesor de teoría imparte también el laboratorio, es decir, sólo hay un profesor para esta asignatura.

El enunciado de la práctica será dado por el profesor con uno o dos días de anticipación, para que los estudiantes ya tengan una idea de cómo van a dar solución a la práctica y cualquier duda o consulta que tengan se la preguntarán al profesor en el laboratorio.

El profesor garantizará que en cada una de las máquinas esté instalado el software a utilizar que en este caso es el Microsoft Visual Studio (específicamente el compilador de Visual C++). Otros aspectos a tener en cuenta en la organización del laboratorio son los siguientes:

- La primera sesión es para hacer una introducción al laboratorio, explicar el manejo básico de la herramienta y la realización de una práctica de repaso usando el lenguaje C.

- Cada una de las prácticas va enfocada al afianzamiento del aprendizaje de cada uno de los temas de la teoría.

- Cada práctica debe ser realizada en un tiempo estipulado, el cual debe ser de fiel cumplimiento por parte de los estudiantes.

- Cada estudiante debe entregar en tiempo y forma una memoria con la solución de las prácticas que indique el profesor. Esto equivale al 20% de la nota final de la asignatura.

4.1.1 Objetivos

Los objetivos del **"Laboratorio de Programación Visual"** son:

- Afianzar los conocimientos adquiridos en las clases teóricas de la asignatura.
- Conocer el manejo de la herramienta de trabajo (Micosoft Visual C++).
- Ayudar al alumno en el análisis y resolución de problemas usando la metodología de programación orientada a objetos y la construcción de aplicaciones con interfaces gráficas.
- Sentar las bases necesarias para la aplicación de estos conocimientos en otras asignaturas que tienen estrecha relación, como por ejemplo, la asignatura de Ingeniería de Software que se imparte en V año.

4.1.2 Planificación Temporal

Como se ha mencionado en el capitulo 4 inciso 4.1 para el desarrollo del laboratorio se cuenta con 15 laboratorios de dos horas durante el semestre para un total de 30 horas.

El número de horas asignadas para cada tema se ha calculado en base a la profundidad con que se requiere abordar cada uno de ellos. Se deberán hacer un total de 9 prácticas usando el compilador de Visual C++.

La planificación es la siguiente:

PRÁCTICA	SEMANA	CONTENIDO	HORAS
1, 2 y 3	1-5	Ventana principal, gráficos y diálogos	10
4 y 5	6-8	Fuentes y color, cajas de diálogo estándar	6
6	9-10	Subclasificación y diálogos	4
7	11-12	Menús dinámicos y flotantes	4
8 y 9	13-15	Otros controles. Seriación	6
		TOTAL_HORAS	**30**

4.1.3 Contenido del Temario de Laboratorio

PRÁCTICAS 1, 2 Y 3: INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL. VENTANA PRINCIPAL, GRÁFICOS, DIÁLOGOS.

Crear la aplicación

- Organización del laboratorio
- Manejo de la herramienta
- El asistente AppWizard
- Pequeños ajustes
- Una caja de edición y una etiqueta
- Dar la hora (al principio)
- Dar la hora (cuando la pida el usuario). Un botón
- Desfase horario

Reloj analógico. Una caja de diálogo

- Crear la caja de diálogo no modal
- Datos del reloj analógico: centro, tamaño y hora
- Pintar las manecillas v1.0. Líneas
- Pintar las manecillas v1.1. Pinceles, brochas y elipses
- Repintar al cambiar de tamaño. Invalidate()
- Dar la hora ¡todo el tiempo! Temporizador
- Mostrar / ocultar el reloj analógico con un botón

Cambiar la hora. El ratón

- Capturar los mensajes del ratón y calcular la nueva hora
- Cambiar la hora. Puntero a la vista

Posibles mejoras (opcional)

PRÁCTICAS 4 Y 5: FUENTES Y COLOR, CAJAS DE DIÁLOGO ESTÁNDAR

Introducción

- Ajustar los minutos

Fuentes y color

- Fuentes
- Color

Cajas de diálogo estándar

Menús

Posibles mejoras (Opcional)

PRÁCTICA 6: SUBCLASIFICACIÓN Y DIÁLOGOS

Subclasificación y diálogos

- Una nueva clase de control
- Utilización de la nueva clase. Alarma
- Reutilización. Un nuevo diálogo con un control de la misma clase.

Posibles mejoras (Opcional)

PRÁCTICA 7: MENÚS DINÁMICOS Y FLOTANTES

Menús dinámicos y flotantes

- Estado de los elementos de menú
- Menús dinámicos: añadir / eliminar opciones en tiempo de ejecución
- Aceleradores
- Menús flotantes

Posibles mejoras (Opcional)

PRÁCTICAS 8 Y 9: OTROS CONTROLES. SERIACIÓN

Otros controles

- Casilla de verificación
- Listas
- Cambio de las dimensiones de la ventana de diálogo

Seriación

- Guardar los datos de las zonas horarias en un fichero
- Obtener los datos de las zonas horarias de un fichero y almacenarlos en el documento
- Añadir al menú zona los elementos correspondientes a las zonas recuperadas del fichero

4.2 PROGRAMA DEL LABORATORIO DE PROGRAMACIÓN VISUAL

ASIGNATURA:	PROGRAMACIÓN VISUAL - LABORATORIO
CURSO:	2004/2005
DEPARTAMENTO:	COMPUTACIÓN – DC
TITULACIÓN:	INGENIERÍA EN SISTEMAS DE INFORMACIÓN
AÑO:	IV
PROFESOR:	LIC. ÁLVARO ALTAMIRANO OSORIO.

TEMARIO DE LABORATORIO:

PRÁCTICAS 1, 2 Y 3: INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL. VENTANA PRINCIPAL, GRÁFICOS, DIÁLOGOS.

Crear la aplicación

- Organización del laboratorio
- Manejo de la herramienta
- El asistente AppWizard
- Pequeños ajustes
- Una caja de edición y una etiqueta
- Dar la hora (al principio)
- Dar la hora (cuando la pida el usuario). Un botón
- Desfase horario

Reloj analógico. Una caja de diálogo

- Crear la caja de diálogo no modal

- Datos del reloj analógico: centro, tamaño y hora
- Pintar las manecillas v1.0. Líneas
- Pintar las manecillas v1.1. Pinceles, brochas y elipses
- Repintar al cambiar de tamaño. Invalidate()
- Dar la hora ¡todo el tiempo! Temporizador
- Mostrar / ocultar el reloj analógico con un botón

Cambiar la hora. El ratón

- Capturar los mensajes del ratón y calcular la nueva hora
- Cambiar la hora. Puntero a la vista

Posibles mejoras (opcional)

PRÁCTICAS 4 Y 5: FUENTES Y COLOR, CAJAS DE DIÁLOGO ESTÁNDAR

Introducción

- Ajustar los minutos

Fuentes y color

- Fuentes
- Color

Cajas de diálogo estándar

Menús

Posibles mejoras (Opcional)

PRÁCTICA 6: SUBCLASIFICACIÓN Y DIÁLOGOS

Subclasificación y diálogos

- Una nueva clase de control
- Utilización de la nueva clase. Alarma
- Reutilización. Un nuevo diálogo con un control de la misma clase.

Posibles mejoras (Opcional)

PRÁCTICA 7: MENÚS DINÁMICOS Y FLOTANTES

Menús dinámicos y flotantes

- Estado de los elementos de menú
- Menús dinámicos: añadir / eliminar opciones en tiempo de ejecución
- Aceleradores
- Menús flotantes

Posibles mejoras (Opcional)

PRÁCTICAS 8 Y 9: OTROS CONTROLES. SERIACIÓN

Otros controles

- Casilla de verificación
- Listas
- Cambio de las dimensiones de la ventana de diálogo

Seriación

- Guardar los datos de las zonas horarias en un fichero
- Obtener los datos de las zonas horarias de un fichero y almacenarlos en el documento
- Añadir al menú zona los elementos correspondientes a las zonas recuperadas del fichero

MATERIAL DE ESTUDIO:

PÁGINA WEB:

- http: \\www.isi.unanleon.edu.ni\~alvaro\

BIBLIOGRAFÍA:

- Ceballos Sierra, Francisco Javier. Visual C++, Aplicaciones para Win32, 2da. Edición. Editorial RAMA

LABORATORIO E INFORMACIÓN DEL PROFESOR:

- **No de laboratorio**: 2
- **Horario de tutorías**:

  Lunes 16:00 - 18:00

  Viernes 16:00 - 18:00
- **Correo**: alvaro@unanleon.edu.ni

EVALUACIÓN DEL LABORATORIO:

CONTENIDO DE LAS MEMORIAS:

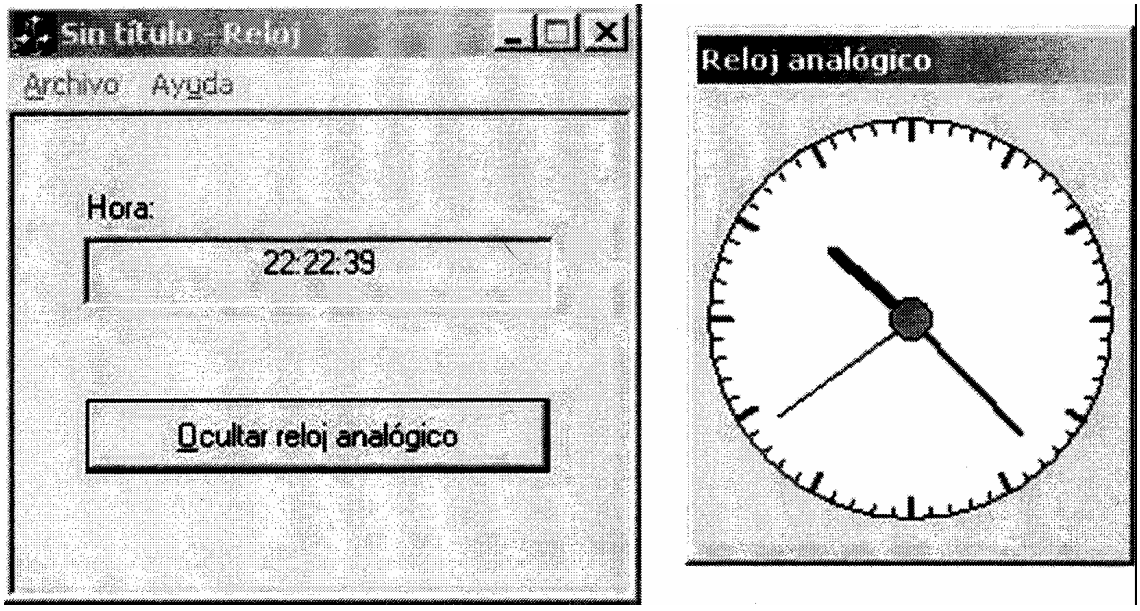
- Enunciado de la práctica
- Código **comentado** de los programas
- Mejoras adicionales
- Conclusiones y comentarios

4.3 PRÁCTICAS 1, 2 Y 3: INTRODUCCIÓN A LA PROGRAMACIÓN VISUAL. VENTANA PRINCIPAL, GRÁFICOS, CAJAS DE DIÁLOGO.

4.3.1 Enunciado de la práctica

Introducción

Se realizará una aplicación con el siguiente aspecto:



En el punto 1 se creará una aplicación que muestre la hora del sistema en una ventana. En la misma ventana habrá un botón que el usuario tendrá que pulsar cuando quiera que se actualice la hora mostrada.

En el punto 2 se añadirá a la aplicación la ventana del reloj analógico. La hora, además de aparecer en la caja de texto, se "dibujará" en forma de reloj de agujas. Esto requerirá ciertos cálculos trigonométricos sencillos. Además se

mejorará el programa para que actualice la hora cada segundo, de forma que el reloj se pondrá en marcha.

En el punto 3 se dotará a la aplicación de la funcionalidad necesaria para permitir que el usuario cambie la hora arrastrando las agujas con el ratón.

1. Crear la aplicación

1.1 El asistente AppWizard

Crear una aplicación SDI llamada Reloj. Crearla sin barra de herramientas, sin barra de estado y sin presentación preliminar. Seleccionar cero en la caja del número de archivos en "recent file list". Seleccionar CFormView como clase base de la vista.

Compilar y ejecutar para ver el resultado.

1.2 Pequeños ajustes

Editar CMainFrame::PreCreateWindow() y añadir la siguiente línea para que el usuario no pueda cambiar el tamaño de la ventana:

```
cs.style &= ~(FWS_ADDTOTITLE|WS_THICKFRAME|WS_MAXIMIZEBOX);
```

Quitar el menú Edición y las órdenes del menú Archivo, excepto Salir.

Compilar y ejecutar para ver el resultado.

Con la ayuda de ClassWizard, añadir la función OnInitialUpdate() a CRelojView. Añadir las siguientes líneas al final de dicha función:

```
GetParentFrame()->RecalcLayout();
ResizeParentToFit();
```

Compilar y ejecutar para ver el resultado.

1.3 Una caja de edición y una etiqueta

Editar el diálogo IDD_RELOJ_FORM. Quitar la etiqueta que hay en él y añadir una caja de edición con id IDC_HORA, texto centrado y read-only. Centrarla en la ventana y ponerle encima una etiqueta con el texto "Hora: ".

Compilar y ejecutar para ver el resultado.

1.4 Dar la hora (al principio)

Añadir las siguientes líneas al final de CRelojView::OnInitialUpdate():

```
CTime Hora = CTime::GetCurrentTime();
CString HoraTxt = Hora.Format("%H:%M:%S");
SetDlgItemText(IDC_HORA, HoraTxt);
```

Compilar y ejecutar para ver el resultado.

1.5 Dar la hora (cuando la pida el usuario). Un botón

Quitar las tres últimas líneas añadidas y ponerlas en una nueva función miembro protegida llamada MostrarHoraActual(). Poner una llamada a esta función al final de CRelojView::OnInitialUpdate().

Editar el diálogo IDD_RELOJ_FORM. Añadir un botón debajo de la caja de edición. Ponerle id IDC_ACTUALIZAR y texto “Actualizar”.

Con ayuda de ClassWizard, añadir la función OnActualizar() que responda al mensaje BN_CLICKED del botón. Poner en esa función una llamada a MostrarHoraActual().

Compilar y ejecutar para ver el resultado.

1.6 Desfase horario

Añadir a CRelojView el siguiente dato miembro privado:

```
CTimeSpan m_DesfaseHorario;
```

Cambiar la primera línea de CRelojView::MostrarHoraActual() por:

```
CTime Hora = CTime::GetCurrentTime() + m_DesfaseHorario;
```

Compilar y ejecutar para ver el resultado.

Inicializar m_DesfaseHorario en el constructor de CRelojView así:

m_DesfaseHorario = CTimeSpan(0, 0, 0, 0);

Compilar y ejecutar para ver el resultado.

2. Reloj analógico. Una caja de diálogo

2.1 Crear la caja de diálogo no modal

Con el editor de recursos, crear un nuevo diálogo con id IDD_RELOJ_ANALOGICO y título “Reloj analógico”. Quitar el estilo “System menu”. Poner el estilo “Visible”. Darle un tamaño más o menos cuadrado.

Entrar en ClassWizard, que ofrecerá la posibilidad de crear una nueva clase para el diálogo que acabamos de diseñar. Aceptar y llamar a la clase CDlgRelojAnalogico. Observar que el código de la clase se almacena en los ficheros DlgRelojAnalogico.h y DlgRelojAnalogico.cpp. Sin salir de ClassWizard, añadir las funciones OnOk y OnCancel. Estas funciones responden al mensaje BN_CLICKED de los botones identificados por IDOK e IDCANCEL, respectivamente. Editarlas y eliminar la llamada que hacen a su versión en la clase base. De esta forma se evita que se cierre el diálogo cuando se haga clic en alguno de estos dos botones o se pulse la tecla correspondiente. Finalmente elimine los botones.

Añadir al constructor CDlgRelojAnalogico:

```
Create(IDD_RELOJ_ANALOGICO);
```

Añadir a la clase CRelojView el siguiente dato miembro e iniciarlo a NULL en el constructor de la clase:

```
CDlgRelojAnalogico *m_pDlgRelojAnalogico;
```

Habrá que poner #include “DlgRelojAnalogico.h” al principio de RelojView.h

Añadir la siguiente línea a la función miembro OnInitialUpdate de CRelojView antes de la llamada a MostrarHoraActual():

```
m_pDlgRelojAnalogico = new CDlgRelojAnalogico (this);
```

Añadir la siguiente línea al destructor de CRelojView:

```
delete m_pDlgRelojAnalogico;
```

Compilar y ejecutar para ver el resultado.

2.2 Datos del reloj analógico: centro, tamaño y hora

Añadir los siguientes datos miembros privados a la clase CDlgRelojAnalogico:

```
int m_nCentroX;
int m_nCentroY;
int m_nRadio;

CTime m_Hora;
```

Añadir también la siguiente función miembro pública:

```
void SetHora(const CTime & Hora) ( m_Hora = Hora; )
```

Iniciar m_Hora a la hora actual y el resto de los miembros a 0, en el constructor de CDlgRelojAnalogico antes de la llamada a Create.

Con ayuda de ClassWizard, añadir a la clase CDlgRelojAnalogico la función miembro OnSize() para manejar el mensaje WM_SIZE. Editar el código y añadir al final las siguientes líneas:

```
if (nType == SIZE_MAXIMIZED || nType == SIZE_RESTORED)
{
m_nCentroX = cx / 2;
m_nCentroY = cy / 2;

m_nRadio = m_nCentroX < m_nCentroY ? m_nCentroX : m_nCentroY;
}
```

Añadir las siguientes líneas al final de CRelojView::MostrarHoraActual():

```
m_pDlgRelojAnalogico->SetHora(Hora);
m_pDlgRelojAnalogico->Invalidate();
```

2.3 Pintar las manecillas v1.0. Líneas

Con la ayuda de ClassWizard, añadir la función OnPaint() a CDlgRelojAnalogico para manejar el mensaje WM_PAINT. Editar el código y dejarlo así:

```
void CDlgRelojAnalogico::OnPaint()
{
CPaintDC dc(this);         // device context for painting

if (m_nRadio > 10)
 {
 double dAlfa, dX, dY;

 // Manecilla de las horas
 dAlfa = (m_Hora.GetHour()%12) * 3.1416 * 2 / 12;
 dAlfa += (m_Hora.GetMinute()) * 3.1416 * 2 / 12 / 60;

 dX = sin(dAlfa) * m_nRadio;
 dY = -cos(dAlfa) * m_nRadio;

 dc.MoveTo(m_nCentroX, m_nCentroY);
 dc.LineTo(m_nCentroX+int(dX*.5), m_nCentroY+int(dy*.5));

 // Manecilla de los minutos
 // Escriba su código aquí

 // Manecilla de los segundos
 // Escriba su código aquí
 }
}
```

Hará falta poner #include<math.h> al principio de CDlgRelojAnalogico.cpp.

Compilar y ejecutar para ver el resultado.

2.4 Pintar las manecillas v1.1. Pinceles, brochas y elipses

Modificar CDlgRelojAnalogico::OnPaint() y dejarlo así (esto se puede simplificar e incluso eliminar). Ahora se ha establecido el origen de coordenadas en el centro y la coordenada Y aumenta hacia arriba:

```
void CDlgRelojAnalogico::OnPaint()
{
CPaintDC dc(this);   // device context for painting

if (m_nRadio > 10)
 {
 //TODO: Add your message handler code here
 double dAlfa, dX, dY;
 int i;

 CPen LapizGordoNegro(PS_SOLID,2,RGB(0,0,0));
 CPen LapizGordoRojo(PS_SOLID,2,RGB(192,0,0));
 CPen LapizMuyGordoAzul(PS_SOLID,4,RGB(0,0,192));
 CPen *pLapizNormal;

 CBrush BrochaGris(RGB(128,128,128));
 CBrush *pBrochaNormal;

 // Modo de proyección
 dc.SetMapMode(MM_ANISOTROPIC);
 // Tamaño de la ventana lógica
```

```
dc.SetWindowExt(m_nCentroX*2, m_nCentroY*2);
// Tamaño de la superficie de proyección
dc.SetViewportExt(m_nCentroX*2, -m_nCentroY*2);
// Origen de coordenadas
dc.SetViewportOrg(m_nCentroX, m_nCentroY);

// Esfera

dc.Ellipse(-int(m_nRadio*.95), -int(m_nRadio*.95),
           int(m_nRadio*.95), int(m_nRadio*.95));

// Marcas

for (i=0; i<60; i++)
{
dAlfa = i * 3.1416 * 2 / 60;

dX = sin(dAlfa) * m_nRadio;
dY = cos(dAlfa) * m_nRadio;

if (i%5 == 0)
 {
 pLapizNormal = dc.SelectObject(&LapizGordoNegro);

 dc.MoveTo(int(dX*.85), int(dY*.85));
 dc.LineTo(int(dX*.95), int(dY*.95));

 dc.SelectObject(pLapizNormal);
 }
else
 {
 dc.MoveTo(int(dX*.9), int(dY*.9));
 dc.LineTo(int(dX*.95), int(dY*.95));
 }
}

// Manecilla de las horas
dAlfa = (m_Hora.GetHour()%12) * 3.1416 * 2 / 12;
dAlfa += (m_Hora.GetMinute()) * 3.1416 * 2 / 60 / 12;

dX = sin(dAlfa) * m_nRadio;
dY = cos(dAlfa) * m_nRadio;

dc.SelectObject(&LapizMuyGordoAzul);

dc.MoveTo(0, 0);
dc.LineTo(int(dX*.5), int(dY*.5));

// Manecilla de los minutos

// Escriba su código aquí

// Manecilla de los segundos

// Escriba su código aquí

// Botón central

pBrochaNormal = dc.SelectObject(&BrochaGris);

dc.Ellipse(-int(m_nRadio*.1), -int(m_nRadio*.1),
           int(m_nRadio*.1), int(m_nRadio*.1));
```

```
 dc.SelectObject(pBrochaNormal);
 }
}
```

Compilar y ejecutar para ver el resultado.

2.5 Repintar al cambiar de tamaño. Invalidate()

Cambiar la propiedad “Border” del diálogo IDD_RELOJ_ANALOGICO. Ponerle borde “Resizing”.

Compilar y ejecutar para ver el resultado. Cambiar el tamaño de la ventana del reloj analógico.

Añadir la siguiente línea al final del bloque if de CDlgRelojAnalogico::OnSize():

```
Invalidate();
```

Compilar y ejecutar para ver el resultado.

2.6 Dar la hora ¡todo el tiempo! Temporizador

Añadir el siguiente símbolo a los recursos de IDD_RELOJ_FORM para identificar al temporizador (menú View \ Resource Symbols):

```
ID_1SEGUNDO
```

Añadir el siguiente dato miembro privado a la clase CRelojView:

```
UINT m_nIdTimer;
```

Añadir la siguiente línea al final de CRelojView::OnInitialUpdate():

```
m_nIdTimer = SetTimer(ID_1SEGUNDO, 1000, NULL);
```

Con ayuda de ClassWizard, añadir la función miembro OnDestroy() a la clase CRelojView para que maneje el mensaje WM_DESTROY. Editar el código y añadirle al principio las siguientes líneas:

```
if (m_nIdTimer)
 KillTimer(ID_1SEGUNDO);
```

Con ayuda de ClassWizard, añadir la función miembro OnTimer a la clase CRelojView para que maneje el mensaje WM_TIMER. Editar el código y añadirle al principio las siguientes líneas:

```
if (IsWindow(m_hWnd) && nIDEvent==m_nIdTimer)
 MostrarHoraActual();
```

Compilar y ejecutar para ver el resultado.

Con la ayuda de ClassWizard, eliminar la función CRelojView::OnActualizar() (ya no es necesaria). El código de la función deberá ser eliminado a mano. Eliminar también el botón “Actualizar” del diálogo IDD_RELOJ_FORM.

Compilar y ejecutar para ver el resultado.

2.7 Mostrar/ocultar el reloj analógico con un botón

Modificar la aplicación para mostrar u ocultar el reloj analógico con un botón IDC_MOSTRAR. Cuando el reloj analógico no esté visible el botón mostrará el título “Mostrar reloj analógico”, y cuando esté visible, “Ocultar reloj analógico”. El diálogo IDD_RELOJ_ANALOGICO tiene la propiedad “Visible” activada. Lo primero que hay que hacer es ir al editor de recursos y desactivarla.

Utilizar la función **ShowWindow** con el argumento **SW_SHOW** para mostrar el diálogo y con el argumento **SW_HIDE** para ocultarlo. Para saber si la ventana está o no visible utilizar la función **IsWindowVisible**. Para modificar el título del botón utilizar **SetDlgItemText**.

Compilar y ejecutar para ver el resultado.

3. Cambiar la hora. El ratón

3.1 Capturar los mensajes del ratón y calcular la nueva hora

Con ayuda de ClassWizard, añadir a CDlgRelojAnalogico las funciones miembro OnMouseMove(), OnLButtonDown() y OnRButtonDown() para que manejen los mensajes WM_MOUSEMOVE, WM_LBUTTONDOWN y WM_RBUTTONDOWN respectivamente. Editar las tres funciones y añadir la llamada:

```
ArrastrarAgujas(nFlags, point);
```

Añadir a CDlgRelojAnalogico la función miembro protegida ArrastrarAgujas, según se indica a continuación:

```
void CDlgRelojAnalogico::ArrastrarAgujas(UINT nFlags, CPoint
point)
{
double dAlfa, dHora;
int nHoras, nMinutos, nSegundos;

if (m_nRadio > 10 &&
    nFlags & (MK_LBUTTON|MK_RBUTTON) &&
    (point.x != m_nCentroX || point.y != m_nCentroY))
 {
 dAlfa = atan2(m_nCentroX – point.x, point.y – m_nCentroY);

 dHora = dAlfa/3.1416/2*12 + 6;

 if (nFlags & MK_RBUTTON)
   dHora += 12;

 nSegundos = int(dHora*3600);
 if (nSegundos < 0)
   nSegundos = 0;
 else if (nSegundos > 3600*24-1)
   nSegundos = 3600*24-1;

 nHoras = nSegundos / 3600;
 nSegundos %= 3600;
 nMinutos = nSegundos / 60;
 nSegundos %= 60;

 // Ahora habría que pasarle la hora a la vista ...
```

```
 TRACE("%02d : %02d : %02d\n", nHoras, nMinutos, nSegundos);
 }
}
```

Compilar y ejecutar para ver el resultado.

3.2 Cambiar la hora. Puntero a la vista

Añadir el siguiente dato miembro privado a la clase CDlgRelojAnalogico:

```
CRelojView *m_pVista;
```

Será necesario hacer una declaración previa de CRelojView.

Añadir la siguiente línea al constructor de CDlgRelojAnalogico:

```
m_pVista = (CRelojView *)pParent;
```

Añadir a CRelojView la siguiente función miembro pública:

```
void CRelojView::CambiarHora(int nHora, int nMinuto, int
nSegundo)
{
CTime Hora = CTime::GetCurrentTime();

CTime HoraNueva(Hora.GetYear(),
               Hora.GetMonth(),
               Hora.GetDay(),
               nHora, nMinuto, nSegundo);

m_DesfaseHorario = HoraNueva – Hora;

MostrarHoraActual();
}
```

Cambiar la macro TRACE del final de CDlgRelojAnalogico::ArrastrarAgujas() por:

```
m_pVista->CambiarHora(nHoras, nMinutos, nSsegundos);
```

Hará falta poner #include "RelojDoc.h"e #include "RelojView.h", en este orden, al principio de DlgRelojAnalogico.cpp.

Compilar y ejecutar para ver el resultado.

Posibles mejoras (opcional)

Las agujas del reloj que pinta la aplicación son muy simples. Se podría hacer un dibujo más elaborado. Este podría ser fijo, pero también podría permitirse al usuario que lo modificase. Para esto último habría que inventar una interfaz para editar el dibujo de las agujas. Puesto que habrá que pintar las agujas en todos los ángulos posibles, es más fácil manejar agujas vectoriales (definidas por líneas y/o polígonos) que agujas de mapas de bits (definidas por matrices de píxeles). Una vez que las agujas tienen superficie, se puede mejorar la forma en la que el ratón las arrastra, fijándolas al cursor del ratón por el punto en el que se las ha empezado a arrastrar, y no por el segmento central de la aguja.

4.4 PRÁCTICAS 4 Y 5: FUENTES Y COLOR, CAJAS DE DIÁLOGO ESTÁNDAR

4.4.1 Enunciado de la práctica

Introducción

Se continuará trabajando con el programa desarrollado anteriormente.

Como ampliación al puntos tres se refinará la interfaz de usuario en lo que respecta a cambiar la hora con el ratón. El botón derecho servirá para arrastrar la manecilla de los minutos y se recurrirá al estado de la tecla Ctrl para que el botón izquierdo adquiera la funcionalidad que antes tenía el botón derecho.

En el punto 4 se modificará el aspecto del reloj digital. En primer lugar se cambiará la fuente de texto. Habrá que resolver los problemas derivados de la existencia de tres unidades de medida distintas (puntos, unidades lógicas y unidades físicas). En segundo lugar se cambiará el color del texto y del fondo. Se comprobará que en realidad hay dos colores de fondo: el del propio texto y el del control.

En el punto 5 se usará una caja de diálogo estándar. Mediante esta caja de diálogo, que el sistema proporciona ya implementada, el usuario podrá escoger el color de la fuente de una forma intuitiva.

En el punto 6 se sustituirán los botones por opciones de menú. Se comprobará que este cambio no afecta significativamente el funcionamiento del programa.

3.3 Ajustar los minutos

Se pretende refinar la interfaz de usuario para que sea más fácil cambiar la hora del reloj. Actualmente resulta muy difícil ajustar los minutos. Se usará para ello el botón derecho del ratón. La funcionalidad actual del botón derecho será soportada mediante la tecla Ctrl. Recapitulando, la funcionalidad del ratón deberá ser:
No se exige que al arrastrar la aguja de los minutos se cambie también la hora (si la aguja de los minutos atraviesa la raya de las doce). No obstante, el código listado a continuación da una pista de cómo puede hacerse esto último:

Eventos	*Reacción*
Pulsación/arrastre del botón izquierdo	Ajuste de **horas** y **minutos**. **PM** si tecla Ctrl pulsada; **AM** en caso contrario
Pulsación/arrastre del botón derecho	Ajuste de **minutos**

```
nHoras = nHorasAntes;
nMinutos = ...;      //  <--  Calcular  según  la  posición  del
ratón
nSegundos = nSegundosAntes;

if (nMinutos > nMinutosAntes &&       // Si pasamos por las doce
    nMinutos – nMinutosAntes > 30)          // hacia atrás...
    nHoras--;                         // ...una hora menos
else if (nMinutos < nMinutosAntes &&
        nMinutosAntes – nMinutos > 30)    //  Si  pasamos  hacia
delante...
    nHoras++;                         // ...una hora más

if (nHoras == -1)
   nHoras = 23;
else if (nHoras == 24)
   nHoras = 0;
```

4. Fuentes y color

4.1 Fuentes

Añadir a la clase CRelojView el siguiente dato miembro privado:

```
CFont m_FuenteHora;
```

Con la ayuda de ClassWizard, añadir a la variable miembro m_ctrlHora (categoría Control y tipo CEdit) ligada a la caja de texto IDC_HORA.

Añadir el siguiente fragmento de código al final de CRelojView::OnInitialUpdate():

```
LOGFONT lf;

memset(&lf, 0, sizeof(LOGFONT));
strncpy(lf.lfFaceName, “Times New Roman”, LF_FACESIZE);
lf.lfFaceName[LF_FACESIZE-1] = ‘\0’;
lf.lfHeight = 16;
m_FuenteHora.CreateFontIndirect(&lf);
m_ctrlHora.SetFont(&m_FuenteHora);
```

Compilar y ejecutar para ver el resultado. Comparar el tamaño de letra con el de la letra Times New Roman 16 del WordPad.

Consultar en la ayuda el significado exacto del campo lfHeight de la estructura LOGFONT.

Cambiar el signo del valor que se asigna a lf.lfHeight:

```
lf.lfHeight = -16;
```

Compilar y ejecutar para ver el resultado.

Volver a cambiar esa misma línea por la siguiente:

```
lf.lfheight = - UnidadesLogicas(16);
```

Añadir a la clase CRelojView la siguiente función miembro protegida:

```
int CRelojView::UnidadesLogicas(int nPuntosAltFuente)
{
CClientDC dc(this);

POINT pt;

pt.x = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSX) / 72;
pt.y = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSY) / 72;

dc.DPtoLP(&pt);

return pt.y;
}
```

Compilar y ejecutar para ver el resultado.

4.2 Color

Añadir a la clase CRelojView el dato miembro privado m_BrochaFondoHora de la clase CBrush. Usar la función CBrush::CreateSolidBrush() (consultar la ayuda) para inicializar la brocha con el color amarillo en CRelojView::OnInitialUpdate() (al principio).

Con la ayuda de ClassWizard, añadir a la clase CRelojView la función miembro OnCtlColor(), que tratará los mensajes WM_CTLCOLOR.

Añadir el código que se indica a continuación:

```
HBRUSH CRelojView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT
nCtlColor)
{
HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

if (pWnd->GetDlgCtrlID() == IDC_HORA)
 {
 pDC->SetBkColor(RGB(255,0,0)); // rojo
 pDC->SetTextColor(RGB(0,0,255));    // azul

 return (HBRUSH) m_BrochaFondoHora.GetSafeHandle();
 }

 return hbr;
}
```

Compilar y ejecutar para ver el resultado.

Modificar los colores hasta alcanzar un resultado agradable a la vista. Se usará el mismo color para el fondo del texto y para el fondo del control. Guardar los colores en dos nuevos datos miembros privados:

```
COLORREF m_rgbTextoHora;
COLORREF m_rgbFondoHora;
```

Compilar y ejecutar para ver el resultado.

Nota: La caja de texto IDC_HORA debe tener activada la propiedad "Read Only", pero no la propiedad "Disabled", que hace que el texto aparezca en color gris.

5. Cajas de diálogo estándar

Se dispone de las siguientes clases:

Clase	*Propósito*
CFileDialog	Abrir o guardar un fichero
CColorDialog	Seleccionar un color
CFontDialog	Seleccionar un tipo de letra
CFindReplaceDialog	Sustituir una cadena por otra
CPageSetupDialog	Establecer las características de la página a imprimir
CPrintDialog	Imprimir un documento

Se deberá añadir al programa la funcionalidad necesaria para que el usuario pueda cambiar el tipo de letra y el color de fondo del reloj digital.

Las cajas de diálogo estándar se suelen usar como se indica en el siguiente fragmento de código:

```
void CAboutDlg::OnButton1()
{
CFileDialog dlgAbrir(true);

// ...aquí se modifica el objeto dlgAbrir para que
// muestre inicialmente ciertos datos

if (dlgAbrir.DoModal() == IDOK)
 {

 // ...aquí se extrae del objeto dlgAbrir los datos
 // que haya aceptado finalmente el usuario

 // ...
 }
}
```

Una vez que el usuario haya seleccionado el color, se volverá a crear la brocha, llamando de nuevo a CreateSolidBrush(). Pero antes habrá que destruir la brocha vieja. Esto se hará con una llamada DeleteObject(). Algo similar habrá que hacer con la fuente. Para este último caso, observe la estructura CHOOSEFONT que contiene, entre otros, los siguientes campos:

Campo	*Significado*
lpLogFont	Apunta a una estructura de tipo LOGFONT
Flags	Combinación binaria de varios valores. El valor CF_INITTOLOGFONTSTRUCT indica que hay que usar la información apuntada por el campo lpLogFont
rgbColors	Color del texto

6. Menús

Sustituir todos los botones de la ventana principal por órdenes de un menú "Opciones" que incluya una orden por cada botón eliminado.

Posibles mejoras (opcional)

Cualquier programa puede cambiar el cursor del ratón para ayudar al usuario a intuir cuál es la funcionalidad del ratón en un contexto determinado. Por ejemplo: al pasar el ratón por el área cliente de la ventana de un procesador de texto o por cualquier caja de edición, el cursor del ratón toma la forma de una barra vertical, indicando que el usuario puede seleccionar texto o hacer un clic y teclear...

La aplicación Reloj podría poner al cursor del ratón un dibujo especial cuando éste pasase por encima de las agujas del reloj analógico. Esto se puede hacer añadiendo (con ayuda de ClassWizard) a la clase CDlgRelojAnalogico una función que responda al mensaje WM_SETCURSOR. Dentro de esa función habrá que determinar si se debe cambiar el cursor del ratón o no y, en caso afirmativo, cambiarlo llamando a la función SetCursor(). Si se quiere usar un cursor de ratón dibujado con el editor de recursos (en lugar de uno de los proporcionados por el sistema) habrá que llamar a la función LoadCursor(). Esta última función pertenece a la clase de la aplicación, así que habrá que llamar a la función AfxGetApp() para acceder a ella.

Otra posibilidad es ajustar el tamaño de la caja de texto al tamaño de la fuente seleccionada en cada momento. Para ello son necesarias, entre otras, las funciones CDC::GetTextExtent(), CWnd::GetWindowRect(), CWnd::SetWindowPos() y CWnd::ScreenToClient(). Esta última es necesaria para traducir las coordenadas devueltas por CWnd::GetWindowRect().

4.5 PRÁCTICA 6: SUBCLASIFICACIÓN Y DIÁLOGOS

4.5.1 Enunciado de la práctica

Introducción

Se continuará trabajando con el programa desarrollado anteriormente.

Se añadirá una alarma al reloj. El usuario podrá introducir la hora de la alarma a través del teclado. El control de edición correspondiente sólo aceptará caracteres numéricos y el carácter dos puntos (':'). Para ello será necesario crear previamente una nueva clase de control, derivada de CEdit. Una función miembro de esta nueva clase se ocupará de filtrar la entrada (ignorando cualquier carácter no válido). Mediante ClassWizard se vinculará la caja de edición de la alarma a un objeto de esta nueva clase. La alarma sonará emitiendo un pitido cada segundo durante cinco minutos a partir de la hora establecida por el usuario.

En el punto 6.3 se creará un nuevo diálogo que permitirá al usuario definir una zona horaria. En prácticas posteriores se añadirá la funcionalidad necesaria para que el programa pueda almacenar una lista de zonas horarias y mostrar en cualquier momento la hora correspondiente a una de ellas. El nuevo diálogo se probará en esta práctica sin esperar a añadir toda esa funcionalidad. Se usará la macro TRACE para ver el resultado.

6. Subclasificación y diálogos

6.1 Una nueva clase de control

Añadir al formulario principal una caja de edición con el identificador IDC_HORA_ALARMA.

Con ayuda de Class Wizard, crear una clase llamada CEditHora derivada de CEdit.

También con ayuda de Class Wizard, añadir a la clase CRelojView la variable miembro m_ctrlHoraAlarma (categoría Control y tipo CEditHora) ligada a la caja de texto IDC_HORA_ALARMA.

Con ayuda de Class Wizard, añadir a la clase CEditHora la función OnChar(), que deberá responder al mensaje WM_CHAR. Programarla de forma que sólo llame a la implementación de la clase base en caso de que la tecla pulsada sea la de un número, la del símbolo ':', o la tecla VK_BACK. Cuando el usuario pulse cualquier otra tecla, deberá emitirse un pitido (ver función MessageBeep()).

Compilar y ejecutar para ver el resultado.

Opcionalmente se puede cambiar la fuente y el color de fondo de la nueva caja de edición.

6.2 Utilización de la nueva clase. Alarma

Añadir a CEditHora las siguientes funciones miembro públicas:

```
void CEditHora::SetHora(int nHoras, int nMinutos, int nSegundos)
{
CString sHora;
sHora.Format("%02d:%02d:%02d", nHoras, nMinutos, nSegundos);
SetWindowText(sHora);
}

bool CEditHora::GetHora(int & nHoras,
                        int & nMinutos,
                        int & nSegundos) const
{
CString sHora;
GetWindowText(sHora);

int nDatosLeidos = sscanf(sHora, "%d:%d:%d",
                        &nHoras, &nMinutos, &nSegundos);

if (nDatosLeidos == 2)          // Formato "HH:MM" (2 datos)
     nSegundos = 0;             // Se asume SS=0
else if (nDatosLeidos != 3)     // Formato "HH:MM:SS" (3 datos)
     return false;          // Ninguno de ellos... hora no valida

if (nHoras < 0 || nHoras > 23 ||
     nMinutos < 0 || nMinutos > 59 ||
     nSegundos < 0 || nSegundos > 59)
          return false;         // Rango mal... hora no valida

return true;          // Hora valida
}
```

Modificar la función CRelojView::OnTimer() según se indica a continuación:

```
void CRelojView::OnTimer(UINT nIDEvent)
{
if (IsWindow(m_hWnd) && nIDEvent == ID_1SEGUNDO)
     {
     MostrarHoraActual();

     int nHorasAlarma,nMinutosAlarma,nSegundosAlarma, nSegundos;
```

```
      // Si la alarma tiene una hora valida...
      if (m_ctrlHoraAlarma.GetHora(nHorasAlarma,
                                   nMinutosAlarma,
                                   nSegundosAlarma) )
            {
            nSegundosAlarma += nMinutosAlarma * 60 +
                               nHorasAlarma * 3600;

            // Total segundos de la alarma
            CTime Hora = CTime::GetCurrentTime() +
                                        m_DesfaseHorario;

            // Total segundos de la hora actual
            nSegundos = Hora.GetSecond() +
                        Hora.GetMinute() * 60 +
                        Hora.GetHour() * 3600;

            if (nSegundos < nSegundosAlarma)
                  {
                  // Aun no ha llegado la hora de la alarma,
                  // pero hay un caso especial en el que debe
                  // sonar: acaba de ser medianoche, estamos en
                  // los primeros minutos del dia y la alarma
                  // estaba puesta en los ultimos minutos de ayer.

                  if (nSegundosAlarma – nSegundos > 86400 - 300)
                        MessageBeep (0xFFFFFFFF);
                  }
            else
                  {
                  // La hora de la alarma ya ha pasado. Si hace
                  // mas de cinco minutos, NO debe sonar. Si aun
                  // no han transcurrido los cinco minutos, SI
                  // debe sonar.

                  if (nSegundos – nSegundosAlarma < 300)
                        MessageBeep(0xFFFFFFFF);
                  }
            }
      }

CFormView::OnTimer(nIDEvent);
}
```

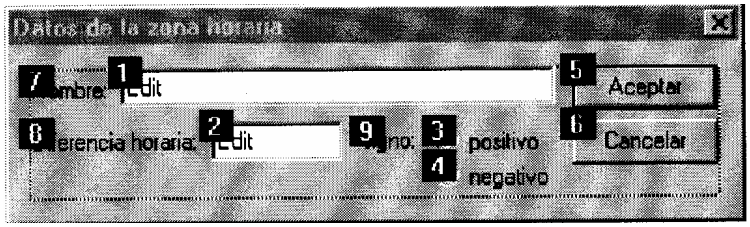
Compilar y ejecutar para ver el resultado.

6.3 Reutilización. Un nuevo diálogo con un control de la misma clase

A continuación añadiremos al programa la funcionalidad necesaria para mostrar al usuario la hora en diferentes zonas horarias. El usuario podrá agregar zonas horarias introduciendo sus datos en una ventana. Diseñaremos ahora esa ventana y la probaremos accediendo a ella con un simple botón.

Añadir un nuevo diálogo con el identificador IDD_DATOS_ZONA_HORARIA:

Habrá que poner especial cuidado en el orden de los controles (el orden en que van siendo recorridos por el foco cuando el usuario pulsa la tecla TAB). Este orden puede ser modificado mediante la orden "Layout \ Tab order... Ctrl+D" del menú. El orden deberá quedar como se indica a continuación:



El control IDC_RADIO_POSITIVO y el botón "Aceptar" deberán tener activada la propiedad "Group". El control IDC_RADIO_NEGATIVO deberá tenerla desactivada. De esta forma se consigue que los dos "radio button" formen parte de un grupo. Así, al marcarse uno, se desmarca el otro.

Con ayuda de ClassWizard, crear una nueva clase para el diálogo IDD_DATOS_ZONA_HORARIA y llamarla CDlgDatosZonaHoraria.

La caja de edición del nombre (IDC_NOMBRE_ZONA) estará vinculada a un dato de tipo CString llamado m_sNombreZona (usar ClassWizard). La caja de edición de la zona horaria (IDC_DIFERENCIA_HORARIA) estará vinculada a un control CEditHora llamado m_ctrlDiferenciaHoraria.

El primer "radio button" (IDC_RADIO_POSITIVO) estará vinculado a un dato de tipo int llamado m_nSigno.

Añadir un botón titulado "Probar dlg. zona horaria" en la vista del programa. Con ayuda de ClassWizard, añadir una función que responda a la pulsación del botón. Dicha función deberá crear un objeto de la clase CDlgDatosZonaHoraria, y mostrarlo como una ventana modal. En caso de que el usuario acepte los datos introducidos, el programa deberá mostrarlos con TRACE().

```
void CRelojView::OnProbar()
{
CDlgDatosZonaHoraria dlgZona;

if (dlgZona.DoModal() == IDOK)
      {
      CString sNombreZona;
      int nSigno;
      int nHoras, nMinutos, nSegundos;

      sNombreZona = dlgZona.m_sNombreZona;
      nSigno = dlgZona.m_nSigno;
```

```
      dlgZona.m_ctrlDiferenciaHoraria.GetHora(nHoras,
                                              nMinutos,
                                              nSegundos);

      TRACE("Nombre: %s\nDesfase horario: %s %02d:%02d:%02d\n",
            SNombreZona, nSigno?"-":"+",
            nHoras, nMinutos, nSegundos);
      }
}
```

Compilar y ejecutar para ver el resultado.

Como se ha llamado a CEditHora::GetHora() desde la función que ha mostrado la ventana, ocurre un error en tiempo de ejecución. La razón es que la ventana modal se destruye cuando el usuario la cierra. Con ella se destruyen los "objetos windows" de sus controles. Por eso, cuando la ventana ya no existe, no se puede llamar a CEditHora::GetHora() (que a su vez llama a GetWindowText()).

Añadir variables miembro públicas a CDlgDatosZonaHoraria para guardar en ellas la hora cuando el usuario pulse el botón "Aceptar". Cambiar la función CRelojView::OnProbar para que utilice esas variables en lugar de llamar a CDlgDatosZonaHoraria::GetHora().

La función CDlgDatosZonaHoraria::OnOK() se puede implementar de forma que no permita al usuario aceptar datos no válidos:

```
void CDlgDatosZonaHoraria::OnOK()
{
// TODO: Add extra validation here

UpdateData(true);

if (m_sNombreZona == "")
      {
      AfxMessageBox("Por favor, introduzca un nombre de
                                          zona valido. ");
      return;
      }

if (!m_ctrlDiferenciaHoraria.GetHora(m_nHoras,
                                     m_nMinutos,
                                     m_nSegundos))
      {
      AfxMessageBox("Por favor, introduzca una diferencia
                                          horaria valida. ");
      return;
      }
if (m_nSigno != 0 && m_nSigno != 1)
      {
      AfxMessageBox("Por favor, especifique el signo de la
                                     diferencia horaria. ");
      return;
      }

CDialog::OnOK();
}
```

Compilar y ejecutar para ver el resultado.

El código generado por ClassWizard inicializa el dato miembro m_nSigno con el valor –1 (ver constructor de la clase CDlgDatosZonaHoraria). Cambiar ese valor por el valor 0 para que aparezca seleccionado por defecto el signo positivo. Esto permitirá eliminar el código que se añadió para testear m_nSigno.

Compilar y ejecutar para ver el resultado.

Posibles mejoras (opcional)

Además de trabajar con horas, el programa podría almacenar y mostrar la fecha. Se podría escribir la fecha en el reloj analógico. Sería interesante hacer que el tamaño de fuente empleado fuese proporcional al tamaño del reloj.

Se podría permitir al usuario cambiar la fecha usando el control Active X “Calendar Control”. Habría que insertarlo en un nuevo diálogo.

4.6 PRÁCTICA 7: MENÚS DINÁMICOS Y FLOTANTES

4.6.1 Enunciado de la práctica

Introducción

Se continuará trabajando con el programa desarrollado anteriormente.

En el punto 7.1 se añadirá al programa opciones de menú para que el usuario pueda activar y desactivar el despertador. Esto se hará de tres maneras, practicando con las distintas posibilidades existentes: cambiar el texto de una opción de menú, poner o quitar la marca ✓ (check) a una opción de menú y habilitar o inhabilitar una opción de menú.

En el punto 7.2 se añadirá al programa un menú de zonas horarias. Las opciones de este menú se añadirán y se eliminarán en tiempo de ejecución. En principio no se cambiará la hora cuando el usuario seleccione una zona horaria. Cuando esto ocurra, el programa se limitará a mostrar un mensaje.

La funcionalidad del menú de zonas horarias se completará en prácticas posteriores.

En el punto 7.3 se añadirán aceleradores para todas las acciones implementadas anteriormente (activar despertador, desactivar...). Se comprobará que a veces no funcionan porque las opciones de menú están inhabilitadas (se habilitarán al desplegarse el menú, pero eso no sirve para los aceleradores). Se subsanará el error habilitando antes las opciones de menú.

En el punto 7.4 se añadirán menús flotantes a la aplicación.

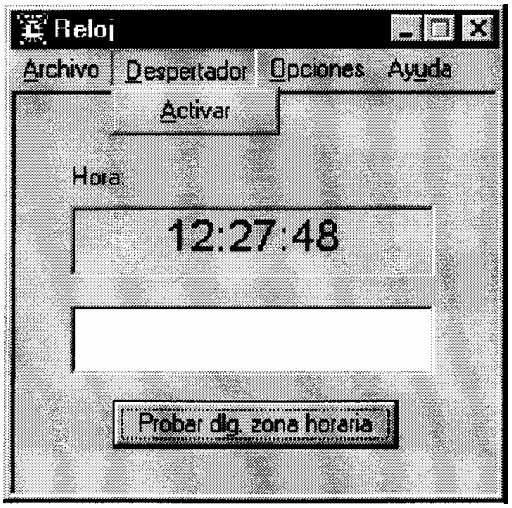
7. Menús dinámicos y flotantes

7.1 Estado de los elementos de menú

Añadir a la vista una variable miembro privada llamada m_bDespertadorActivado de tipo bool. Modificar la función OnTimer() de forma que sólo funcione la alarma

cuando m_bDespertadorActivado valga true. Iniciar la nueva variable miembro con el valor false en el constructor de la clase.

Añadir al menú la orden “Despertador \ Activar” según indica la siguiente figura:



Con ayuda de ClassWizard, añadir la siguiente función miembro (seleccionando el mensaje COMMAND para el identificador ID_DESPERTADOR_ACTIVAR en la clase CRelojView):

```
void CRelojView::OnDespertadorActivar()
{
m_bDespertadorActivado = !m_bDespertadorActivado;
}
```

Compilar y ejecutar para ver el resultado. ¿Cómo puede el usuario saber si el despertador está activado? Sólo puede saberlo esperando a que llegue la hora en que tiene que sonar...

De nuevo con ayuda de ClassWizard, añadir la siguiente función miembro (esta vez seleccionando el mensaje UPDATE_COMMAND_UI en lugar de COMMAND).

```
void CRelojView::OnUpdateDespertadorActivar(CCmdUI *pCmdUI)
{
pCmdUI->SetText(m_bDespertadorActivado ?
                “&Desactivar” : “&Activar”);
}
```

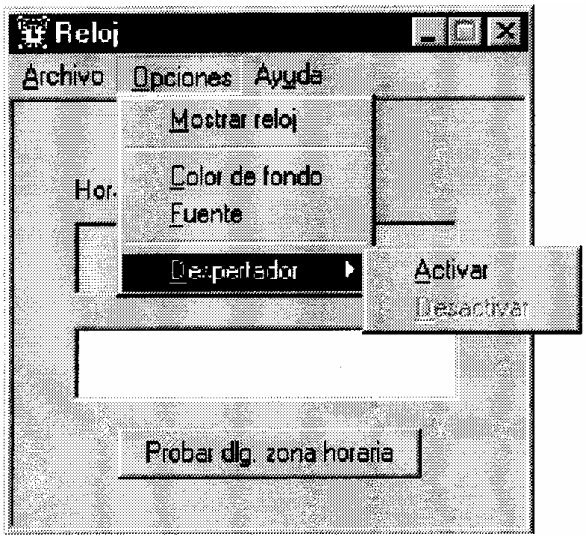
Compilar y ejecutar para ver el resultado.

Probar también el siguiente código:

```
void CRelojView::OnUpdateDespertadorActivar(CCmdUI *pCmdUI)
{
pCmdUI->SetCheck(m_bDespertadorActivado ? 1 : 0);
}
```

Compilar y ejecutar para ver el resultado.

Añadir un submenú “Despertador...” al menú “Opciones” según indica la siguiente figura:



Con ayuda de ClassWizard, añadir las funciones necesarias para que el submenú active y desactive el despertador, apareciendo habilitada únicamente la opción del submenú que corresponda en cada caso.

NOTA: En las funciones OnUpdateXXXXX(CCmdUI *pCmdUI) hay que usar el parámetro pCmdUI para manipular la orden de menú.

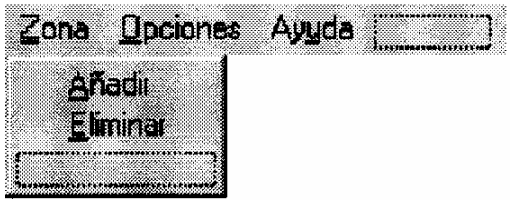
Compilar y ejecutar para ver el resultado.

7.2 Menús dinámicos: añadir/eliminar opciones en tiempo de ejecución

Añadir el siguiente dato miembro privado a la clase CRelojView e inicializarlo a cero en el constructor:

```
int m_nZonas;
```

Añadir un menú “Zona” según indica la siguiente figura:



Añadir a CRelojView::OnInitialUpdate() el siguiente código:

```
CMenu *pMenu = GetParent()->GetMenu();     // Menú principal
pMenu = pMenu->GetSubMenu(0);         // Primer submenú: “Zona”

if (m_nZonas == 0)
{
pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                     ID_ZONA_ELIMINAR,     // Opción “Eliminar”
                     “&Eliminar”);
pMenu->AppendMenu(MF_SEPARATOR);
}
```

```
char aszNombre[] = "Zona -";
aszNombre[strlen(aszNombre)-1] = 'A' + m_nZonas;

pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                      ID_BASE_ZONA + m_nZonas,   // Zona n-ésima
                      aszNombre);

m_nZonas++;

if (m_nZonas == NUM_MAX_ZONAS)
      pMenu->RemoveMenu(ID_ZONA_ANADIR,     // Opción "Añadir"
                            MF_BYCOMMAND);
}
```

Definir las macros ID_BASE_ZONA y NUM_MAX_ZONAS como se indica a continuación:

```
#define ID_BASE_ZONA (ID_ZONA_ELIMINAR+100)
#define NUM_MAX_ZONAS 4
```

Compilar y ejecutar para ver el resultado.

Con ayuda de ClassWizard, añadir la función CRelojView::OnZonaEliminar(), que deberá ejecutarse en respuesta a la orden "Eliminar" del menú "Zona". Esta función deberá eliminar la última zona del submenú. Si se había alcanzado el número máximo de zonas, habrá que insertar la opción "Añadir". Si la zona eliminada era la única que había, habrá que eliminar también la propia opción "Eliminar" y el separador.

Compilar y ejecutar para ver el resultado.

Añadir la siguiente función miembro protegida a la clase CRelojView:

```
bool CRelojView::OnZona(UINT nID)
{
CMenu *pMenu = GetParent()->GetMenu();
pMenu = pMenu->GetSubMenu(0);

CString sNombre;

pMenu->GetMenuString(nID, sNombre, MF_BYCOMMAND);

AfxMessageBox(sNombre, MB_OK|MB_ICONINFORMATION);

return true;
}
```

Para que OnZona() pueda responder a los mensajes, habrá que usar el modificador afx_msg en su declaración:

```
afx_msg bool OnZona(UINT nID);
```

Añadir la siguiente entrada al mapa de mensajes de la clase CRelojView:

```
ON_COMMAND_EX_RANGE(ID_BASE_ZONA,
                        ID_BASE_ZONA + NUM_MAX_ZONAS – 1,
                              OnZona)
```

Compilar y ejecutar para ver el resultado.

7.3 Aceleradores

Añadir los aceleradores correspondientes a las órdenes del menú.

NOTA: Si los aceleradores de teclado no funcionan después de haber usado el menú, puede deberse a que las opciones del menú están deshabilitadas. Para evitar que esto ocurra, hay que habilitarlas expresamente al final de OnRButtonUp(), OnOpcionesDespertadorActivar() y OnOpcionesDespertadorDesactivar().

7.4 Menús flotantes

Con la ayuda de ClassWizard, añadir a la clase CRelojView la siguiente función miembro, que deberá ejecutarse cuando el usuario pulse el botón derecho del ratón:

```
void CRelojView::OnRButtonDown(UINT nFlags, CPoint point)
{
CMenu *pMenu = GetParent()->GetMenu();
pMenu = pMenu->GetSubMenu(0);

ClientToScreen(&point);
pMenu->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,
                          point.x, point.y, this, NULL);

CFormView::OnRButtonDown(nFlags, point);
}
```

Compilar y ejecutar para ver el resultado.

Modificar CRelojView::OnRButtonDown() de forma que la tecla Ctrl haga que se despliegue el submenú “Despertador” en lugar del menú “Zona”.

Compilar y ejecutar para ver el resultado. ¿Se inhabilitan las opciones “Activar” y “Desactivar” cuando es debido?

Retocar el código añadido para que habilite o inhabilite las opciones del menú justo antes de mostrarlo. Usar la función CMenu::EnableMenuItem().

Posibles mejoras (opcional)

Se podría mostrar la hora de la alarma en el reloj analógico. Se puede hacer dibujando una nueva aguja, o pintando un punto en el borde del reloj... Sería interesante que el dibujo indicara de alguna manera si la alarma está activada o no, y si la hora es AM o PM. También sería interesante que el usuario pudiese cambiar la hora de la alarma con el ratón.

4.7 PRÁCTICAS 8 Y 9: OTROS CONTROLES. SERIACIÓN

4.7.1 Enunciado de la práctica

Introducción

Se continuará trabajando con el programa desarrollado anteriormente.

En el punto 8.1 se añadirá al programa una casilla de verificación que además de permitir al usuario activar o desactivar la alarma fácilmente, le mostrará en todo momento el estado de la alarma.

En el punto 8.2 se creará un nuevo diálogo en el que se usará un control de tipo lista. El diálogo servirá para que el usuario pueda eliminar zonas horarias. La lista mostrará los nombres de las diferentes zonas horarias.

En el punto 8.3 se hará que el diálogo mencionado anteriormente pueda cambiar de tamaño, y que los controles contenidos en él también cambien de tamaño y/o posición para aprovechar todo el espacio disponible. A continuación se completará la funcionalidad del programa en lo que respecta a añadir, usar y eliminar zonas horarias. El programa almacenará los datos de las zonas horarias en el documento. Habrá que poner un cuidado especial en mantener la consistencia entre esos datos y las opciones del menú. Cuando se añada una zona horaria habrá que añadirla en los dos sitios...

En el punto 9 se implementará la seriación de los datos de las zonas horarias. El programa guardará estos datos en un archivo al cerrarse, de forma que cuando se abra de nuevo, se cargarán de nuevo los datos y el usuario podrá volver a usar las zonas horarias que introdujo la última vez que usó el programa.

8. Otros controles

8.1 Casilla de verificación

Eliminar la función CRelojView::OnProbar() y el botón “Probar dlg. zona horaria”.

Añadir al formulario IDD_RELOJ_FORM una casilla de verificación (check box) con el texto "Despertador activado" y el identificador IDC_DESPERTADOR_ACTIVADO. Con ayuda de ClassWizard, enlazarlo a una nueva variable miembro de la clase CButton llamada m_ctrlDespertadorActivado (categoría "Control").

Añadir el código necesario para que la casilla de verificación active y desactive el despertador. Cuando el usuario active o desactive el despertador mediante las órdenes del menú, la casilla de verificación deberá cambiar de estado.

El estado de la casilla de verificación deberá concordar en todo momento con el valor de la variable miembro m_bDespertadorActivado. Es conveniente que OnInitialUpdate() asegure esta concordancia desde el principio.

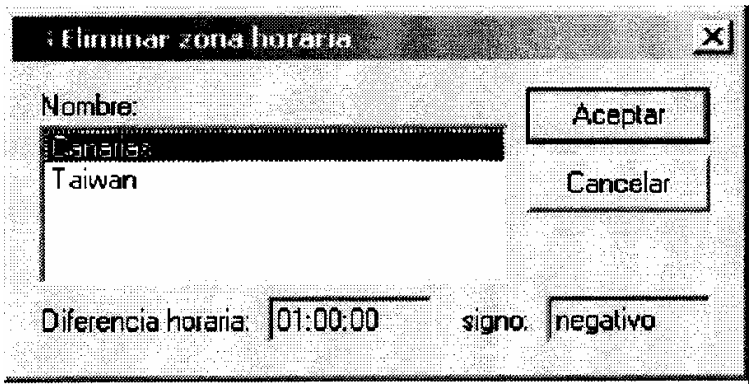
Compilar y ejecutar para ver el resultado.

8.2 Listas

Añadir a los recursos un nuevo diálogo con el identificador IDD_ELIMINAR_ZONA_HORARIA y el título "Eliminar Zona Horaria". Copiar los controles del diálogo IDD_DATOS_ZONA_HORARIA y pegarlos en el nuevo diálogo (también se puede copiar y pegar el diálogo entero y después cambiarle el identificador y el título).

Sustituir la caja de edición IDC_NOMBRE_ZONA por una lista (list box) con el identificador IDC_LISTA_ZONAS. Desactivar la propiedad "Tab stop" y activar la propiedad "Read Only" en el control IDC_DIFERENCIA_HORARIA. Sustituir los "radio button" por una caja de edición (también "Read Only" y no "Tab stop") con el identificador IDC_SIGNO.

Seleccionar "Resizing" en la propiedad "Border" del diálogo. Cambiar los identificadores de las etiquetas "Diferencia horaria:" y "Signo:" por IDC_STATIC_DIFERENCIA_HORARIA e IDC_STATIC_SIGNO respectivamente.



Con ayuda de ClassWizard, crear una nueva clase llamada CDlgEliminarZona para el diálogo IDD_ELIMINAR_ZONA_HORARIA.

Añadir a la nueva clase las siguientes variables miembro privadas:

```
int m_nTamMinX;       // El tamaño inicial de la ventana
int m_nTamMinY;       // sera su tamaño mínimo
```

Iniciar m_nTamMinX a –1 en el constructor de la clase.

8.3 Cambio de las dimensiones de la ventana de diálogo

Con ayuda de ClassWizard, añadir las funciones miembro OnInitDialog() y OnSizing(), que responderán a los mensajes WM_INITDIALOG y WM_SIZING respectivamente. Para que WM_SIZING aparezca en la lista de mensajes de la ventana de ClassWizard, hay que cambiar a "Window" el filtro de mensajes de la clase CDlgEliminarZona ("ClassInfo" \ "Advanced options" \ "Message filter"). Una vez añadida la función, ya se puede volver a poner el filtro de mensajes que había antes.

Aplicar la limitación de tamaño

```
void CDlgEliminarZona::OnSizing(UINT fwSide, LPRECT pRect)
{
CDialog::OnSizing(fwSide, pRect);

int nTamX = pRect->right – pRect->left;    // Nuevo tamaño
int nTamY = pRect->bottom – pRect->top;

// Si el rectángulo de la ventana es demasiado pequeño,
// hay que ampliarlo por la esquina/lado por la
// que el usuario la esté intentando redimensionar.

if (nTamX < m_nTamMinX)                // Ancho insuficiente...
     if (fwSide==WMSZ_LEFT ||          // ...ampliar hacia la
          fwSide==WMSZ_TOPLEFT ||      // derecha o hacia la izq.
          fwSide==WMSZ_BOTTOMLEFT)
          pRect->left = pRect->right – m_nTamMinX;
     else
          pRect->right = pRect->left + m_nTamMinX;

                                       // Alto insuficiente...
     /* ...COMPLETAR... */             // ...ampliar hacia
                                       // arriba o hacia abajo

}
```

Tomar como tamaño mínimo el tamaño inicial del diálogo

```
BOOL CDlgEliminarZona::OnInitDialog()
{
CDialog::OnInitDialog();

CRect rVentana;

GetWindowRect(&rVentana);

m_nTamMinX = rVentana.Width();  // Guardamos el tamaño inicial.
m_nTamMinY = rVentana.Height(); // Ese será el tamaño mínimo

return TRUE;
}
```

Añadir a la vista un botón provisional que muestre un diálogo modal de la clase CDlgEliminarZona.

Compilar y ejecutar para ver el resultado.

Coordenadas relativas de los controles a la esquina inferior derecha y tamaño relativo de la lista respecto del área cliente

Añadir a la nueva clase las siguientes variables miembro privadas:

```
int m_nPosRelEtiquetasY;        // Posiciones y tamaños
int m_nPosRelCajasY;            // relativos al tamaño
int m_nPosRelBotonesX;          // de la ventana
int m_nTamRelListaX;
     int m_nTamRelListaY;
```

Añadir a CDlgEliminarZona::OnInitDialog() el código que se indica a continuación:

```
BOOL CDlgEliminarZona::OnInitDialog()
{
CDialog::OnInitDialog();
CRect rVentana, rCliente, rControl;
CWnd *pControl;

GetWindowRect(&rVentana);  // Tam. y pos incluyendo el marco

GetClientRect(&rCliente);  // Area cliente: lo de dentro
ClientToScreen(&rCliente);

m_nTamMinX = rVentana.Width();  // Guardamos el tamaño inicial
m_nTamMinY = rVentana.Height(); // Ese será el tamaño mínimo

// Coordenadas relativas al borde inferior
// (para los controles que se van a quedar
// pegados al borde inferior de la ventana)
// Distancia desde los controles hasta el
// borde inferior. SIGNO POSITIVO.

pControl = GetDlgItem(IDC_STATIC_DIFERENCIA_HORARIA);
pControl->GetWindowRect(rControl);
m_nPosRelEtiquetasY = rCliente.bottom – rControl.top;

pControl = GetDlgItem(IDC_DIFERENCIA_HORARIA);
pControl->GetWindowRect(rControl);
m_nPosRelCajasY = rCliente.bottom - rControl.top;

          // Coordenadas relativas al borde derecho
          // (para los controles que se van a quedar
          // pegados al borde derecho de la ventana)
          // Distancia desde los controles hasta el
          // borde derecho. SIGNO POSITIVO.

pControl = GetDlgItem(IDOK);
pControl->GetWindowRect(rControl);
m_nPosRelBotonesX = rCliente.right - rControl.left;

          // Tamaño relativo al tamaño del área de cliente
          // (para el control que se va a hinchar con la
          // ventana)
          // Tamaño del área cliente menos tamaño del control.
          // SIGNO POSITIVO.

pControl = GetDlgItem(IDC_LISTA_ZONAS);
pControl->GetWindowRect(rControl);
m_nTamRelListaX = rCliente.Width() - rControl.Width();
```

```
m_nTamRelListaY = rCliente.Height() - rControl.Height();

return TRUE;
}
```

Redimensionar el diálogo conservando la posición de los controles. El tamaño de la lista variará proporcionalmente

Con ayuda de ClassWizard, añadir la función miembro OnSize(), que responderá al mensaje WM_SIZE.

```
void CDlgEliminarZona::OnSize(UINT nType, int cx, int cy)
{
      if (m_nTamMinX != -1)
      {
      CRect rControl, rCliente;
      CWnd *pControl;

      GetClientRect(rCliente);
      ClientToScreen(rCliente);

      // Controles que quedan pegados al borde inferior

      pControl = GetDlgItem(IDC_STATIC_DIFERENCIA_HORARIA);
      pControl->GetWindowRect(rControl);
      rControl.top = rCliente.bottom - m_nPosRelEtiquetasY;
      ScreenToClient(rControl);
      pControl->SetWindowPos(NULL,
                              rControl.left,rControl.top,
                              0, 0,
                              SWP_NOSIZE|SWP_NOZORDER);
      /*
      ...COMPLETAR...

      Usar m_nPosRelEtiquetasY para colocar la
      etiqueta IDC_STATIC_SIGNO

      Usar m_nPosRelCajasY para colocar las
      cajas de edición IDC_DIFERENCIA_HORARIA
      e IDC_SIGNO

      */

      // Controles que quedan pegados al borde derecho

      /* ... COMPLETAR ... */

      // Control que se hincha al agrandarse la ventana

      pControl = GetDlgItem(IDC_LISTA_ZONAS);
      pControl->SetWindowPos(NULL,
                           0, 0,
                           rCliente.Width()- m_nTamRelListaX,
                           rCliente.Height() - m_nTamRelListaY,
                           SWP_NOMOVE|SWP_NOZORDER);
      }

      CDialog::OnSize(nType, cx, cy);
}
```

Compilar y ejecutar para ver el resultado.

Clase de objetos para almacenar los datos referentes a las zonas horarias

Con ayuda de ClassWizard, añadir una nueva clase llamada CZonaHoraria y derivada de CDocument. Modificar manualmente el código para hacer que el constructor sea público. Añadir los siguientes datos miembro públicos:

```
int m_nHoras;
int m_nMinutos;
int m_nSegundos;
bool m_bPositivo;
CString m_sNombre;
```

El documento almacenará un array de objetos CZonaHoraria. Añadir a la clase CRelojDoc el siguiente dato miembro público:

```
CObArray *m_pArrayZonas;
```

Reservar memoria en el constructor para el objeto CObArray y liberarla en el destructor.

Liberar la memoria de los objetos CZonaHoraria referenciados por m_pArrayZonas

Con ayuda de ClassWizard, redefinir la función DeleteContents() para la clase CRelojDoc:

```
void CRelojDoc::DeleteContents()
{
	int i, nZonas = m_pArrayZonas->GetSize();

	for (i=0; i<nZonas; i++)
		delete m_pArrayZonas->GetAt(i);

	m_pArrayZonas->RemoveAll();

	CDocument::DeleteContents();
}
```

Modificar CRelojView::OnZonaAnadir() (ver código más adelante) de forma que muestre un diálogo modal de la clase CDlgDatosZonaHoraria y cree un nuevo objeto de la clase CZonaHoraria (guardando en él los datos) y lo almacene en el array de objetos del documento. El texto de la nueva opción del menú será el nombre de la zona horaria.

Modificar también CRelojView::OnZonaEliminar() (ver código más adelante) de forma que permita eliminar la última zona del array de zonas contenido en el documento.

```
void CRelojView::OnZonaAnadir()
{
	CDlgDatosZonaHoraria dlg;

	if (dlg.DoModal() != IDOK)
		return;

	int i, nZonas = GetDocument()->m_pArrayZonas->GetSize();

	CZonaHoraria *pZona;
```

```
      for (i=0; i<nZonas; i++)  // No puede haber duplicados
      {
      pZona = (CZonaHoraria *) GetDocument()->
                                        m_pArrayZonas->GetAt(i);

      if (pZona->m_sNombre == dlg.m_sNombreZona)
      {
      AfxMessageBox("Ya existe una zona con ese nombre",
                                        MB_OK|MB_ICONEXCLAMATION);
      return;
      }
      }

      CMenu *pMenu = ...;  //Primer submenú "Zona"
      pMenu = pMenu->GetSubMenu(1);

      // Si la que se añade es la primera zona, hay
      // que añadir al submenú "Zona" la opción "Eliminar" y
      // un separador. ...

      pZona = new CZonaHoraria;

      // Almacenar los datos en el objeto

      GetDocument()->m_pArrayZonas->Add(pZona);

      // Añadimos la nueva zona al menú

      pMenu->AppendMenu(....);

      // Incrementar el número de zonas

      // Si se alcanza el número máximo de zonas, eliminar del
      // submenú "Zona" la opción "Añadir"

}

void CRelojView::OnZonaEliminar()
{

CMenu *pMenu = GetParent()->GetMenu();     // Menú Principal
pMenu = pMenu->GetSubMenu(0);         //Primer submenú "Zona"

int nZonas = GetDocument()->m_pArrayZonas->GetSize();

if (nZonas == NUM_MAX_ZONAS)
      pMenu->InsertMenu(0,MF_BYPOSITION|MF_ENABLED|MF_STRING,
                              ID_ZONA_ANADIR,
                              "&Añadir");

      pMenu->RemoveMenu(ID_BASE_ZONA + nZonas – 1, // Última zona
                              MF_BYCOMMAND);


      delete GetDocument()->m_pArrayZonas->GetAt(nZonas-1);

      nZonas--;

      GetDocument()->m_pArrayZonas->SetSize(nZonas);

      if (nZonas == 0)
            {
            pMenu->RemoveMenu(2, MF_BYPOSITION);
```

```
            pMenu->RemoveMenu(ID_ZONA_ELIMINAR, MF_BYCOMMAND);
            }
}
```

Compilar y ejecutar para ver el resultado.

Eliminar el botón de prueba y la función correspondiente (los que servían para probar el diálogo "Eliminar Zona").

Visualizar el diálogo IDD_ELIMINAR_ZONA_HORARIA que permitirá eliminar la zona seleccionada de la lista.

Con la ayuda de ClassWizard, añadir a la clase CDlgEliminarZona una variable pública con el nombre m_ctrlListaZonas, de tipo CListBox (categoría Control) vinculada a la lista fija IDC_LISTA_ZONAS.

Añadir al final de CDlgEliminarZona::OnInitDialog() el código necesario para cargar en la lista fija los nombres de las zonas horarias almacenadas en el documento. Se puede acceder al documento a través de la vista, cuya dirección deberá recibir como parámetro el constructor de CDlgEliminarZona. Este mismo constructor la almacenará en una variable miembro añadida a tal efecto.

Compilar y ejecutar para ver el resultado.

Añadir a la clase CDlgEliminarZona la siguiente variable miembro pública:

```
CString m_sZonaSeleccionada;
```

Con ayuda de ClassWizard, añadir a la clase CDlgEliminarZona la función miembro OnOK(), que deberá responder a la pulsación del botón "Aceptar" guardando en m_sZonaSeleccionada el nombre de la zona que haya seleccionado el usuario en la lista. En caso de que no haya ninguna zona seleccionada, habrá que mostrar un mensaje de error.

```
void CDlgEliminarZona::OnOK()
{
      int nSel = m_ctrlListaZonas.GetCurSel();

      if (nSel == LB_ERR)
      {
      AfxMessageBox("Por favor, seleccione una zona horaria. ");
      return;
      }

      m_ctrlListaZonas.GetText(nSel, m_sZonaSeleccionada);
      CDialog::OnOK();
}
```

Compilar y ejecutar para ver el resultado.

Añadir a la clase CDlgEliminarZona la siguiente variable miembro privada:

```
CRelojDoc *m_pDoc;
```

Añadir a CRelojView::OnZonaEliminar() el código necesario para que muestre un diálogo modal de la clase CDlgEliminarZona. Durante su construcción deberá guardar en m_pDoc la dirección del documento.

Modificar el código de CRelojView::OnZonaEliminar() de forma que no elimine la última zona horaria sino la seleccionada por el usuario.

```
void CRelojView::OnZonaEliminar()
{
      CMenu *pMenu = ...          // Primer submenú: "Zona"

      int nZonas = ...;           // Obtener no. de zonas

      CDlgEliminarZona dlg(this);

      // Llamar a DoModal() para el objeto dlg...

      // Averiguar cuál es la zona seleccionada por el usuario
      // y comprobar que existe.

      // Si el no. de zonas es el máximo, al quitar una, hay
      // que añadir al submenú "Zona" la orden "Añadir".

      // Eliminar del submenú "Zona" todas las opciones
      // desde la correspondiente a la zona que hay que
      // eliminar. Después hay que volver a añadirlas
      // (todas excepto la que estamos eliminando) con
      // sus nuevos identificadores. Hay que hacer lo mismo
      // con el array de zonas del documento.

      // Liberar mem. del objeto CzonaHoraria

      // Decrementar no. de zonas

      // Si ya no quedan zonas, hay que quitar del submenú
      // "Zona" el separador y la opción "Eliminar".
}
```

Añadir el código necesario para mostrar los datos de la zona horaria seleccionada en cada momento. Con ayuda de ClassWizard, añadir a la clase CDlgEliminarZona la función miembro OnSelchangeListaZonas(), que deberá responder al mensaje LBN_SELCHANGE provocado por el control IDC_LISTA_ZONAS.

```
void CDlgEliminarZona::OnSelchangeListaZonas()
{
int nSel = m_ctrlListaZonas.GetCurSel();

if (nSel == LB_ERR)
      {
      // Poner cadenas vacías en IDC_DIFERENCIA_HORARIA
      // y en IDC_SIGNO
      }
else
      {
      // Buscar por el array del documento la zona
      // seleccionada (mediante el nombre), leer sus datos,
      // y mostrarlos en IDC_DIFERENCIA_HORARIA y en IDC_SIGNO
      }
}
```

Compilar y ejecutar para ver el resultado.

Añadir a la vista una caja de texto y una etiqueta para que muestren la hora correspondiente a una de las zonas horarias del documento. La zona horaria a utilizar será la última añadida o la última seleccionada por el usuario en el menú "Zona". El número de la zona horaria seleccionada se guardará en la vista en una variable miembro privada llamada m_nZonaSeleccionada.

```
class CRelojView : public CFormView
{
private:

//...
      int m_nZonaSeleccionada;
//...
};

CRelojView::CRelojView():CFormView(CRelojView::IDD)
{
//...

m_nZonaSeleccionada = -1;
}

void CRelojView::OnZonaAnadir()
{
//...

int i, nZonas = GetDocument()->m_pArrayZonas->GetSize();

//...

m_nZonaSeleccionada = nZonas;

nZonas++;

//...
}

void CRelojView::OnZonaEliminar()
{
//...

int nZonas = GetDocument()->m_pArrayZonas->GetSize();

//...

nZonas--;

if (m_nZonaSeleccionada>=nZonas)
      m_nZonaSeleccionada = nZonas-1;

//...
}

bool CRelojView::OnZona(UINT nID)
{
m_nZonaSeleccionada = nID - ID_BASE_ZONA;

return true;
}
```

```
void CRelojView::MostrarHoraActual()
{
CTime Hora = CTime::GetCurrentTime() + m_DesfaseHorario;
CString HoraTxt = Hora.Format("%H:%M:%S");
SetDlgItemText(IDC_HORA, HoraTxt);

if (m_nZonaSeleccionada != -1)
     {
     CZonaHoraria *pZona;
     pZona = (CZonaHoraria *)GetDocument()->m_pArrayZonas->
                                GetAt(m_nZonaSeleccionada);

     if (pZona->m_bPositivo)
          Hora += CTimeSpan(0, pZona->m_nHoras,
                                pZona->m_nMinutos,
                                pZona->m_nSegundos);
     else
          Hora -= CTimeSpan(0, pZona->m_nHoras,
                                pZona->m_nMinutos,
                                pZona->m_nSegundos);

     HoraTxt = Hora.Format("%H:%M:%S");
     SetDlgItemText(IDC_HORA_ZONA, HoraTxt);
     HoraTxt = "Hora en " + pZona->m_sNombre + ":";
     SetDlgItemText(IDC_STATIC_HORA_ZONA, HoraTxt);
     }
     else
     {
     SetDlgItemText(IDC_HORA_ZONA, "");
     SetDlgItemText(IDC_STATIC_HORA_ZONA, "Hora en...");
     }

     m_pDlgRelojAnalogico->SetHora(Hora);
     m_pDlgRelojAnalogico->Invalidate();
}
```

Compilar y ejecutar para ver el resultado.

9. Seriación

Reemplazar la macro DECLARE_DYNCREATE(CZonaHoraria) de ZonaHoraria.h por DECLARE_SERIAL(CZonaHoraria). Reemplazar también la macro IMPLEMENT_DYNCREATE(CZonaHoraria, CDocument) de ZonaHoraria.cpp por IMPLEMENT_SERIAL(CZonaHoraria, Cdocument, 1).

Modificar la función miembro Serialize() para la clase CZonaHoraria según se indica a continuación:

```
void CZonaHoraria::Serialize(CArchive& ar)
{
     if (ar.IsStoring())
     {
     ar << m_sNombre
        << int(m_bPositivo)
        << m_nHoras <<m_nMinutos << m_nSegundos;
     }
     else
     {
     int nPositivo;

     ar >> m_sNombre
```

```
        >> nPositivo
        >> m_nHoras >> m_nMinutos >> m_nSegundos;

      m_bPositivo = nPositivo ? true : false;
      }
}
```

Guardar los datos de las zonas horarias en un fichero

Añadir a la clase CRelojDoc el siguiente dato miembro público:

```
      bool m_bArrayModificado;
```

Iniciarlo a `false` en el constructor de la clase CRelojDoc. Ponerlo a `true` en CRelojView::OnZonaAnadir() y en CRelojView::OnZonaEliminar().

Añadir a CRelojDoc::DeleteContents() el código que se indica a continuación:

```
void CRelojDoc::DeleteContents()
{
      // Guardar datos

      if (m_bArrayModificado)
      {
      CFile fiDatosReloj;        // Abrir

      fiDatosReloj.Open("reloj.dat",CFile::modeCreate
                                        |CFile::modeWrite);

      CArchive arDatosReloj(&fiDatosReloj, CArchive::store);

      arDatosReloj << m_pArrayZonas;  // Seriar array (guardar)

      arDatosReloj.Close();           // Cerrar
      fiDatosReloj.Close();
      }

      // Borrar datos
      int i, nZonas = m_pArrayZonas->GetSize();

      for (i=0; i<nZonas; i++)
            delete m_pArrayZonas->GetAt(i);

      m_pArrayZonas->RemoveAll();

      CDocument::DeleteContents();
}
```

Obtener los datos de las zonas horarias de un fichero y almacenarlos en el documento

Añadir a CRelojDoc::OnNewDocument() el código que se indica a continuación:

```
BOOL CRelojDoc::OnNewDocument()
{
      if (!CDocument::OnNewDocument())
            return FALSE;

      CFile fiDatosReloj;  // Si el archivo existe y se abre...

      if (fiDatosReloj.Open("reloj.dat", CFile::modeRead))
```

```
      {
      CArchive arDatosReloj(&fiDatosReloj, CArchive::load);

      delete m_pArrayZonas;           // Eliminar obj. anterior
      arDatosReloj >> m_pArrayZonas;  // Seriar array (cargar)

      arDatosReloj.Close();                 // Cerrar
      fiDatosReloj.Close();
      }
      return TRUE;
}
```

Compilar y ejecutar para ver el resultado. ¿Se crea el archivo reloj.dat al cerrarse la aplicación? ¿Aparecen los datos en el menú al arrancarla de nuevo?

Aún hay que añadir el código necesario para llevar al menú los datos leídos del archivo. Poner puntos de parada en CRelojDoc::OnNewDocument y en CRelojView::OnInitialUpdate(). ¿Cuántas veces se ejecutan y en qué orden?

En caso de que el número de zonas cargadas coincida con el máximo, habrá que borrar la orden “Añadir” del menú “Zona”.

Añadir al menú zona los elementos correspondientes a las zonas recuperadas del fichero

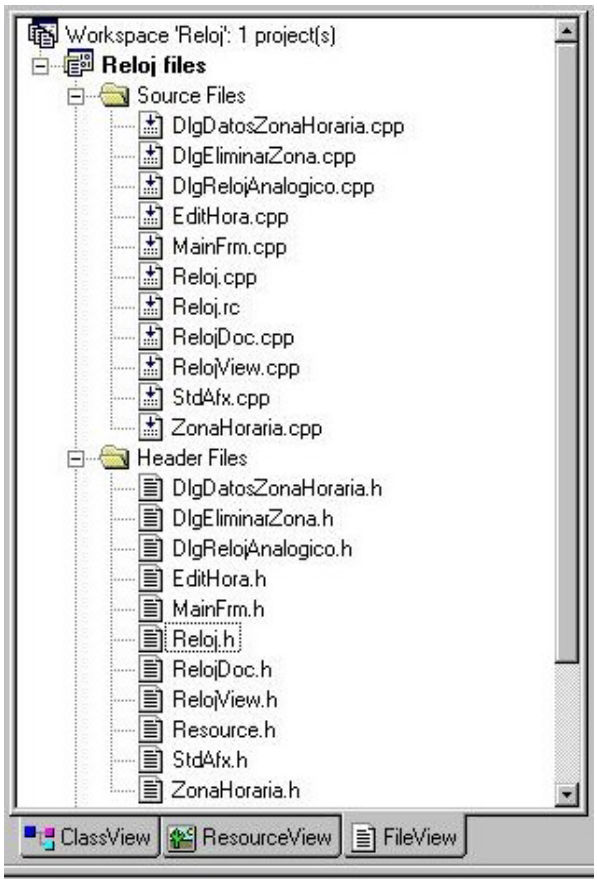
Modificar CRelojView::OnInitialUpdate() como se indica a continuación:

```
void CRelojView::OnInitialUpdate()
{
      //...
      CRelojDoc *pDoc = GetDocument();
      int i, nZonas = pDoc->m_pArrayZonas->GetSize();
      CMenu *pMenu = GetParent()->GetMenu();
      pMenu = pMenu->GetSubMenu(0);

      if (nZonas > 0)
      {
      pMenu->AppendMenu(MF_SEPARATOR);

      CZonaHoraria *pZona;

      for (i=0; i<nZonas; i++)
      {
      pZona = (CZonaHoraria *) pDoc->m_pArrayZonas->GetAt(i);

      pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                        ID_BASE_ZONA + i,
                        pZona->m_sNombre);
            }
      }
      else
            pMenu->RemoveMenu(ID_ZONA_ELIMINAR, MF_BYCOMMAND);

      m_ctrlDespertadorActivado.SetCheck(
                                 m_bDespertadorActivado?1:0);
}
```

4.8 SOLUCIÓN A LAS PRÁCTICAS DE LABORATORIO

4.8.1 Código fuente

A continuación se presenta el código fuente completo de cada una de las clases que forman parte de la aplicación del ***"Reloj"*** la cual se ha venido realizando de forma continuada en cada una de las prácticas guiadas cuyos enunciados fueron dados anteriormente. El orden alfabético de los archivos que componen la aplicación es el siguiente:



// Archivo: DlgDatosZonaHoraria.h

```
#include "EditHora.h"

#if
!defined(AFX_DLGDATOSZONAHORARIA_H__6318EEB9_F9E4_4A8A_A010_654F925FF9A3_
_INCLUDED_)
#define
AFX_DLGDATOSZONAHORARIA_H__6318EEB9_F9E4_4A8A_A010_654F925FF9A3__INCLUDED
_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgDatosZonaHoraria.h : header file
//

/////////////////////////////////////////////////////////////////////////////
////
// CDlgDatosZonaHoraria dialog

class CDlgDatosZonaHoraria : public CDialog
{
    // Construction
  public:
    CDlgDatosZonaHoraria(CWnd* pParent = NULL);   // standard constructor
    int m_Horas, m_Minutos, m_Segundos;
    // Dialog Data
    //{{AFX_DATA(CDlgDatosZonaHoraria)
    enum { IDD = IDD_DATOS_ZONA_HORARIA };
    CEditHora  m_ctrlDiferenciaHoraria;
    CString    m_sNombreZona;
    int        m_nSigno;
  //}}AFX_DATA


  // Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CDlgDatosZonaHoraria)
  protected:
    virtual void DoDataExchange(CDataExchange* pDX);
  // DDX/DDV support
  //}}AFX_VIRTUAL

  // Implementation
  protected:

  // Generated message map functions
  //{{AFX_MSG(CDlgDatosZonaHoraria)
    virtual void OnOK();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_DLGDATOSZONAHORARIA_H__6318EEB9_F9E4_4A8A_A010_654F925FF9A3_
_INCLUDED_)
```

// Archivo: DlgDatosZonaHoraria.cpp
// implementation file

```
#include "stdafx.h"
#include "Reloj.h"
#include "DlgDatosZonaHoraria.h"

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

/////////////////////////////////////////////////////////////////////////////
////
// CDlgDatosZonaHoraria dialog


CDlgDatosZonaHoraria::CDlgDatosZonaHoraria(CWnd* pParent /*=NULL*/)
  : CDialog(CDlgDatosZonaHoraria::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDlgDatosZonaHoraria)
   m_sNombreZona = _T("");
   m_nSigno = 0;
   //}}AFX_DATA_INIT
   m_Horas = 0;
   m_Minutos = 0;
   m_Segundos = 0;
}


void CDlgDatosZonaHoraria::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CDlgDatosZonaHoraria)
   DDX_Control(pDX, IDC_DIFERENCIA_HORARIA, m_ctrlDiferenciaHoraria);
   DDX_Text(pDX, IDC_NOMBRE_ZONA, m_sNombreZona);
   DDX_Radio(pDX, IDC_RADIO_POSITIVO, m_nSigno);
   //}}AFX_DATA_MAP
}



BEGIN_MESSAGE_MAP(CDlgDatosZonaHoraria, CDialog)
  //{{AFX_MSG_MAP(CDlgDatosZonaHoraria)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
////
// CDlgDatosZonaHoraria message handlers

void CDlgDatosZonaHoraria::OnOK()
{
   UpdateData(true);
   if (m_sNombreZona == "")
  {
    AfxMessageBox("Por favor,
                     introduzca un nombre de zona valida. ");
    return;
  }
  if (!m_ctrlDiferenciaHoraria.GetHora(
                                  m_Horas, m_Minutos, m_Segundos))
  {
  AfxMessageBox("Por favor,
              introduzca una diferencia horaria valida. ");
  return;
```

```
  }

  CDialog::OnOK();
}
```

// Archivo: DlgEliminarZona.h

```
#if
!defined(AFX_DLGELIMINARZONA_H__1C69DE38_6522_482A_AC54_8D3418ABD010__INC
LUDED_)
#define
AFX_DLGELIMINARZONA_H__1C69DE38_6522_482A_AC54_8D3418ABD010__INCLUDED_

#include "RelojDoc.h"// Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgEliminarZona.h : header file
//

/////////////////////////////////////////////////////////////////////////////
////
// CDlgEliminarZona dialog
class CRelojView;

class CDlgEliminarZona : public CDialog
{
    // Construction
  private:

    int m_nTamMinX;               // El tamaño inicial de la ventana

  int m_nTamMinY;                         // será su tamaño mínimo
  int m_nPosRelEtiquetasY;  // Posiciones y tamaños
  int m_nPosRelCajasY;            // relativos al tamaño
  int m_nPosRelBotonesX;          // de la ventana
  int m_nTamRelListaX;
  int m_nTamRelListaY;
  CRelojView *m_pView;
  CRelojDoc *m_pDoc;

public:
  CString m_sZonaSeleccionada;
  CDlgEliminarZona(CRelojView *, CWnd* pParent = NULL);   // standard
constructor

// Dialog Data
  //{{AFX_DATA(CDlgEliminarZona)
  enum { IDD = IDD_ELIMINAR_ZONA_HORARIA };
  CListBox     m_ctrlListaZonas;
  CString      m_DiferenciaHoraria;
  CString      m_Signo;
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CDlgEliminarZona)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:

  // Generated message map functions
  //{{AFX_MSG(CDlgEliminarZona)
```

```
  afx_msg void OnSizing(UINT fwSide, LPRECT pRect);
  virtual BOOL OnInitDialog();
  afx_msg void OnSize(UINT nType, int cx, int cy);
  virtual void OnOK();
  afx_msg void OnSelchangeListaZonas();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_DLGELIMINARZONA_H__1C69DE38_6522_482A_AC54_8D3418ABD010__INC
LUDED_)
```

// Archivo: DlgEliminarZona.cpp
// implementation file

```
#include "stdafx.h"
#include "Reloj.h"
#include "DlgEliminarZona.h"
#include "RelojView.h"
#include "ZonaHoraria.h"

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

////////////////////////////////////////////////////////////////////////////
////
// CDlgEliminarZona dialog


CDlgEliminarZona::CDlgEliminarZona(CRelojView *pView, CWnd* pParent
/*=NULL*/)
  : CDialog(CDlgEliminarZona::IDD, pParent)
{
  this->m_nTamMinX = -1;
  this->m_pView = pView;
  m_pDoc = pView->GetDocument();
  //{{AFX_DATA_INIT(CDlgEliminarZona)
  m_DiferenciaHoraria = _T("");
  m_Signo = _T("");
  //}}AFX_DATA_INIT
}


void CDlgEliminarZona::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CDlgEliminarZona)
  DDX_Control(pDX, IDC_LISTA_ZONAS, m_ctrlListaZonas);
  DDX_Text(pDX, IDC_DIFERENCIA_HORARIA, m_DiferenciaHoraria);
  DDX_Text(pDX, IDC_SIGNO, m_Signo);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgEliminarZona, CDialog)
  //{{AFX_MSG_MAP(CDlgEliminarZona)
```

```
  ON_WM_SIZING()
  ON_WM_SIZE()
  ON_LBN_SELCHANGE(IDC_LISTA_ZONAS, OnSelchangeListaZonas)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
////
// CDlgEliminarZona message handlers

void CDlgEliminarZona::OnSizing(UINT fwSide, LPRECT pRect)
{
  CDialog::OnSizing(fwSide, pRect);

  int nTamX = pRect->right - pRect->left;    //Nuevo tamaño
  int nTamY = pRect->bottom - pRect->top;

  // Si el rectángulo de la ventana es demasiado pequeño,
  // hay que ampliarlo por la esquina/lado por la
  // que el usuario la esté intentando redimensionar.

  if (nTamX < m_nTamMinX)             // Ancho insuficiente...
        if (fwSide == WMSZ_LEFT ||    // ... ampliar hacia la
        fwSide == WMSZ_TOPLEFT || // derecha o hacia la izq.
              fwSide == WMSZ_BOTTOMLEFT)
        pRect->left = pRect->right - m_nTamMinX;
  else
        pRect->right = pRect->left + m_nTamMinX;

  if (nTamY < m_nTamMinY)              // Alto insuficiente...
        if (fwSide == WMSZ_LEFT ||       // ... ampliar hacia
        fwSide == WMSZ_TOPLEFT ||    // arriba o hacia abajo
              fwSide == WMSZ_BOTTOMLEFT)
        pRect->top = pRect->bottom - m_nTamMinY;
  else
        pRect->bottom = pRect->top + m_nTamMinY;
}

BOOL CDlgEliminarZona::OnInitDialog()
{
  CDialog::OnInitDialog();

  int i, nZonas;

  CRect rVentana, rCliente, rControl;
  CWnd *pControl;

 // Tamaño y pos. incluyendo el marco
 GetWindowRect(&rVentana);

  GetClientRect(&rCliente); // Area cliente: lo de dentro
  ClientToScreen(&rCliente);

  // Guardamos el tamaño inicial
 m_nTamMinX = rVentana.Width();

 // Ese será el tamaño mínimo
  m_nTamMinY = rVentana.Height();

        // Coordenadas relativas al borde inferior
        // (para los controles que se van a quedar
        // pegados al borde inferior de la ventana)
        // Distancia desde los controles hasta el
        // borde inferior. SIGNO POSITIVO.

  pControl = GetDlgItem(IDC_STATIC_DIFERENCIA_HORARIA);
  pControl->GetWindowRect(rControl);
  m_nPosRelEtiquetasY = rCliente.bottom - rControl.top;
```

```
 pControl = GetDlgItem(IDC_DIFERENCIA_HORARIA);
 pControl->GetWindowRect(rControl);
 m_nPosRelCajasY = rCliente.bottom - rControl.top;

        // Coordenadas relativas al borde derecho
        // (para los controles que se van a quedar
        // pegados al borde derecho de la ventana)
        // Distancia desde los controles hasta el
        // borde derecho. SIGNO POSITIVO.

 pControl = GetDlgItem(IDOK);
 pControl->GetWindowRect(rControl);
 m_nPosRelBotonesX = rCliente.right - rControl.left;

        // Tamaño relativo al tamaño del área de cliente
        // (para el control que se va a hinchar con la
        // ventana)
        // Tamaño del área cliente menos tamaño del control.
        // SIGNO POSITIVO.

 pControl = GetDlgItem(IDC_LISTA_ZONAS);
 pControl->GetWindowRect(rControl);
 m_nTamRelListaX = rCliente.Width() - rControl.Width();
 m_nTamRelListaY = rCliente.Height() - rControl.Height();

 // Código necesario para cargar en la lista fija los
 // nombres de las zonas horarias almacenadas en el doc.

 nZonas = m_pView->GetDocument()->m_pArrayZonas->GetSize();
 CZonaHoraria *pZona;

 for(i = 0; i < nZonas; i++)
 {
        pZona = (CZonaHoraria *)m_pView->
                    GetDocument()->m_pArrayZonas->GetAt(i);

        //Agregar a la lista
        m_ctrlListaZonas.AddString(pZona->m_sNombre);
 }

 // return TRUE unless you set the focus to a control
 // EXCEPTION: OCX Property Pages should return FALSE
 return TRUE;

}

void CDlgEliminarZona::OnSize(UINT nType, int cx, int cy)
{
 if (m_nTamMinX != -1)
 {
        CRect rControl, rCliente;
        CWnd *pControl;

        GetClientRect(rCliente);
        ClientToScreen(rCliente);

        // Controles que quedan pegados al borde inferior

        pControl = GetDlgItem(IDC_STATIC_DIFERENCIA_HORARIA);
        pControl->GetWindowRect(rControl);
        rControl.top = rCliente.bottom - m_nPosRelEtiquetasY;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,
        rControl.left,  rControl.top, 0, 0,
               SWP_NOSIZE|SWP_NOZORDER);

        pControl = GetDlgItem(IDC_DIFERENCIA_HORARIA);
```

```
        pControl->GetWindowRect(rControl);
        rControl.top = rCliente.bottom - m_nPosRelCajasY;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,
                            rControl.left,rControl.top, 0, 0,
                                  SWP_NOSIZE|SWP_NOZORDER);

        pControl = GetDlgItem(IDC_STATIC_SIGNO);
        pControl->GetWindowRect(rControl);
        rControl.top = rCliente.bottom - m_nPosRelEtiquetasY;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,rControl.left,
                                  rControl.top,0, 0,
                                  SWP_NOSIZE|SWP_NOZORDER);

        pControl = GetDlgItem(IDC_SIGNO);
        pControl->GetWindowRect(rControl);
        rControl.top = rCliente.bottom - m_nPosRelCajasY;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,
                     rControl.left, rControl.top,0, 0,
              SWP_NOSIZE|SWP_NOZORDER);

        // Controles que quedan pegados al borde derecho

        pControl = GetDlgItem(IDOK);
        pControl->GetWindowRect(rControl);
        rControl.left = rCliente.right - m_nPosRelBotonesX;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,
                     rControl.left, rControl.top,0, 0,
                            SWP_NOSIZE|SWP_NOZORDER);

        pControl = GetDlgItem(IDCANCEL);
        pControl->GetWindowRect(rControl);
        rControl.left = rCliente.right - m_nPosRelBotonesX;
        ScreenToClient(rControl);
        pControl->SetWindowPos(NULL,rControl.left,
                                  rControl.top,0, 0,
                            SWP_NOSIZE|SWP_NOZORDER);

        // Control que se hincha al agrandarse la ventana

        pControl = GetDlgItem(IDC_LISTA_ZONAS);
        pControl->SetWindowPos(NULL,0, 0,
                     rCliente.Width() - m_nTamRelListaX,
              rCliente.Height() - m_nTamRelListaY,
        SWP_NOMOVE|SWP_NOZORDER);
  }

  CDialog::OnSize(nType, cx, cy);
}

void CDlgEliminarZona::OnOK()
{
  int nSel = m_ctrlListaZonas.GetCurSel();

  if (nSel == LB_ERR)
        {
        AfxMessageBox("Por favor,
           seleccione una zona horaria", MB_ICONINFORMATION);
        return;
        }

  m_ctrlListaZonas.GetText(nSel, m_sZonaSeleccionada);
  CDialog::OnOK();
}
```

```
void CDlgEliminarZona::OnSelchangeListaZonas()
{
int nSel = m_ctrlListaZonas.GetCurSel();

if (nSel == LB_ERR)
  {
  m_DiferenciaHoraria = "";
  m_Signo = "";
  UpdateData(false);
  }
else
  {
  int i, nZonas, h, m, s;
  bool signo;

  // Obtenemos el numero de zonas del array
  nZonas = m_pDoc->m_pArrayZonas->GetSize();

  // Obtenemos el elemento de la lista seleccionado
  m_ctrlListaZonas.GetText(nSel, m_sZonaSeleccionada);

  // Buscamos el elemento seleccionado en el arreglo
  for(i = 0; i < nZonas; i++)
        {
        if( m_sZonaSeleccionada == ((CZonaHoraria *)
              m_pDoc->m_pArrayZonas->GetAt(i))->m_sNombre)
        break;
        }

  signo = ((CZonaHoraria *)m_pDoc->m_pArrayZonas->
                                          GetAt(i))->m_bPositivo;

  // Ponemos el signo: positivo o negativo

  if (signo == true)
        m_Signo = "Diferencia +";
  else
        m_Signo = "Diferencia -";

  // Obtenemos los valores de la hora, minutos y segundos
  h =    ((CZonaHoraria *)m_pDoc->m_pArrayZonas->
                                            GetAt(i))->m_nHoras;
  m =    ((CZonaHoraria *)m_pDoc->m_pArrayZonas->
                                          GetAt(i))->m_nMinutos;
  s =    ((CZonaHoraria *)m_pDoc->m_pArrayZonas->
                                          GetAt(i))->m_nSegundos;

  // Crear un objeto de tipo CTime y asignarselo
  // a m_DiferenciaHoraria

  CTime hora(2002,10,25,h,m,s);
  m_DiferenciaHoraria = hora.Format("%H:%M:%S");

  UpdateData(false);
  }
}
```

// Archivo: DlgRelojAnalogico.h

```
#if
!defined(AFX_DLGRELOJANALOGICO_H__ABD14418_FEDA_4E4C_9683_BD19B90157D5__I
NCLUDED_)
#define
AFX_DLGRELOJANALOGICO_H__ABD14418_FEDA_4E4C_9683_BD19B90157D5__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgRelojAnalogico.h : header file
//

/////////////////////////////////////////////////////////////////////////////
////
// CDlgRelojAnalogico dialog
class CRelojView;

class CDlgRelojAnalogico : public CDialog
{
  int m_nCentroX;
  int m_nCentroY;
  int m_nRadio;
  CTime m_Hora;
  CRelojView *m_pVista;
// Construction
public:
  CDlgRelojAnalogico(CWnd* pParent = NULL);   // standard constructor
  void SetHora(const CTime &Hora) { m_Hora = Hora; }

// Dialog Data
  //{{AFX_DATA(CDlgRelojAnalogico)
  enum { IDD = IDD_RELOJ_ANALOGICO };
        // NOTE: the ClassWizard will add data members here
  //}}AFX_DATA


// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CDlgRelojAnalogico)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  void ArrastrarAgujas(UINT,CPoint);
  // Generated message map functions
  //{{AFX_MSG(CDlgRelojAnalogico)
  virtual void OnOK();
  virtual void OnCancel();
  afx_msg void OnSize(UINT nType, int cx, int cy);
  afx_msg void OnPaint();
  afx_msg void OnMouseMove(UINT nFlags, CPoint point);
  afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
  afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
  afx_msg BOOL OnEraseBkgnd(CDC* pDC);
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_DLGRELOJANALOGICO_H__ABD14418_FEDA_4E4C_9683_BD19B90157D5__I
NCLUDED_)
```

// Archivo: DlgRelojAnalogico.cpp
// implementation file

```
#include "stdafx.h"
#include "Reloj.h"
#include "RelojDoc.h"
#include "RelojView.h"
#include "DlgRelojAnalogico.h"
#include <math.h>

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

/////////////////////////////////////////////////////////////////////////////
////
// CDlgRelojAnalogico dialog


CDlgRelojAnalogico::CDlgRelojAnalogico(CWnd* pParent /*=NULL*/)
  : CDialog(CDlgRelojAnalogico::IDD, pParent)
{
  m_Hora = CTime::GetCurrentTime();
  m_nCentroX = 0;
  m_nCentroY = 0;
  m_nRadio = 0;
  Create(IDD_RELOJ_ANALOGICO);
  m_pVista = (CRelojView *)pParent;
  //{{AFX_DATA_INIT(CDlgRelojAnalogico)
// NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
}

void CDlgRelojAnalogico::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CDlgRelojAnalogico)
  // NOTE: the ClassWizard will add DDX and DDV calls here
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgRelojAnalogico, CDialog)
  //{{AFX_MSG_MAP(CDlgRelojAnalogico)
  ON_WM_SIZE()
  ON_WM_PAINT()
  ON_WM_MOUSEMOVE()
  ON_WM_LBUTTONDOWN()
  ON_WM_RBUTTONDOWN()
  ON_WM_ERASEBKGND()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
////
// CDlgRelojAnalogico message handlers

void CDlgRelojAnalogico::OnOK()
{
  // TODO: Add extra validation here

}

void CDlgRelojAnalogico::OnCancel()
{
  // TODO: Add extra cleanup here
```

```
}

void CDlgRelojAnalogico::OnSize(UINT nType, int cx, int cy)
{
  CDialog::OnSize(nType, cx, cy);

  // TODO: Add your message handler code here
  if (nType == SIZE_MAXIMIZED || nType == SIZE_RESTORED)
        {
        m_nCentroX = cx / 2;
        m_nCentroY = cy / 2;
        m_nRadio = m_nCentroX < m_nCentroY ?
                                          m_nCentroX : m_nCentroY;
        Invalidate();
        }

}

void CDlgRelojAnalogico::OnPaint()
{
  CPaintDC dcFisico(this); // device context for painting

  CDC dcSegP1;              // Contexto de dispositivo
  CBitmap mapaSegP1;        // y mapa en segundo plano
  bool bSegP1;

CBitmap *pmapaAnterior; // Mapa original del CDC de seg. pl.

  CRect r;
  GetClientRect (r);

  if (dcSegP1.CreateCompatibleDC(&dcFisico) &&
        mapaSegP1.CreateCompatibleBitmap(&dcFisico,
                                          r.Width(),
                                    r.Height() ))
  {
  pmapaAnterior = dcSegP1.SelectObject(&mapaSegP1);
  bSegP1 = true;
  }
  else
        bSegP1 = false;

  // Creamos una referencia llamada dc y que apuntará al
  // CDC de segundo plano si es que hemos podido crearlo
  // (si no, apuntará al CDC físico). De esta forma no
  // necesitamos cambiar el código antiguo:

  CDC &dc = * (bSegP1 ? &dcSegP1 : &dcFisico);

  dc.FillSolidRect(r,GetSysColor(COLOR_3DFACE));

// Ahora pintamos igual que antes (usando la referencia dc):

  if (m_nRadio > 10)
        {
        double dAlfa, dX, dY;
        int i;

        CPen LapizGordoNegro(PS_SOLID, 2, RGB(0,0,0));
        CPen LapizGordoRojo(PS_SOLID, 2, RGB(192,0,0));
        CPen LapizMuyGordoAzul(PS_SOLID, 4, RGB(0,0,192));
        CPen *pLapizNormal;

        CBrush BrochaGris(RGB(128,128,128));
        CBrush *pBrochaNormal;

        //Modo de proyeccion
```

```
        dc.SetMapMode(MM_ANISOTROPIC);

        //Tamaño de la ventana logica
        dc.SetWindowExt(m_nCentroX*2, m_nCentroY*2);

        //Tamaño de la superficie de proyeccion
        dc.SetViewportExt(m_nCentroX*2, -m_nCentroY*2);

        //Origen de coordenadas
        dc.SetViewportOrg(m_nCentroX, m_nCentroY);

        //Esfera
        dc.Ellipse(-int(m_nRadio*.95), -int(m_nRadio*.95),
                          int(m_nRadio*.95),int(m_nRadio*.95));

        //Marcas
        for (i=0; i<60; i++)
              {
              dAlfa = i * 3.1416 * 2 / 60;

              dX = sin(dAlfa) * m_nRadio;
              dY = cos(dAlfa) * m_nRadio;

              if (i%5 == 0)
                    {
                    pLapizNormal =
                          dc.SelectObject(&LapizGordoNegro);

                    dc.MoveTo(int(dX*.85), int(dY*.85));
                    dc.LineTo(int(dX*.95), int(dY*.95));

                    dc.SelectObject(pLapizNormal);
                    }
              else
                    {
                    dc.MoveTo(int(dX*.9), int(dY*.9));
                    dc.LineTo(int(dX*.95), int(dY*.95));
                    }
              }

        //Manecilla de las horas
        dAlfa = (m_Hora.GetHour()%12) * 3.1416 * 2 / 12;
        dAlfa += (m_Hora.GetMinute()) * 3.1416 * 2 / 60 / 12;

              dX = sin(dAlfa) * m_nRadio;
              dY = cos(dAlfa) * m_nRadio;

              dc.SelectObject(&LapizGordoNegro);

              dc.MoveTo(0,0);
              dc.LineTo(int(dX*.5), int(dY*.5));

              //Manecilla de los minutos
              dAlfa = (m_Hora.GetMinute()) * 3.1416 * 2 / 60;

              dX = sin(dAlfa) * m_nRadio;
              dY = cos(dAlfa) * m_nRadio;

              dc.SelectObject(&LapizGordoRojo);

              dc.MoveTo(0,0);
              dc.LineTo(int(dX*.5), int(dY*.5));


              //Manecilla de los segundos
              dAlfa = (m_Hora.GetSecond()) * 3.1416 * 2 / 60;

              dX = sin(dAlfa) * m_nRadio;
```

```
                dY = cos(dAlfa) * m_nRadio;

                dc.SelectObject(&LapizMuyGordoAzul);

                dc.MoveTo(0,0);
                dc.LineTo(int(dX*.5), int(dY*.5));


                //Boton Central
                pBrochaNormal = dc.SelectObject(&BrochaGris);

                dc.Ellipse(-int(m_nRadio*.1), -int(m_nRadio*.1),
                                    int(m_nRadio*.1),
                                    int(m_nRadio*.1));

                dc.SelectObject(pBrochaNormal);


                // Ahora, si hemos pintado en segundo plano,
                // copiamos la imagen y seleccionamos el bitmap
                // original del CDC de segundo plano (para que
                // se destruya correctamente):

                if (bSegP1)
                {
                BITMAP bmap;
                mapaSegP1.GetObject(sizeof(bmap),&bmap);
                int m_nAnchoF = bmap.bmWidth, m_nAltoF =
                                                        bmap.bmHeight;

                dc.DPtoLP(&r);

                dc.SetMapMode(MM_TEXT);

                dcFisico.BitBlt(0,0, r.Width(), -r.Height(),
                &dcSegP1, -m_nCentroX,-m_nCentroY, SRCCOPY);

                dcSegP1.SelectObject(pmapaAnterior);
                }
        }
}

void CDlgRelojAnalogico::OnMouseMove(UINT nFlags, CPoint point)
{
  ArrastrarAgujas(nFlags, point);
  CDialog::OnMouseMove(nFlags, point);
}

void CDlgRelojAnalogico::OnLButtonDown(UINT nFlags, CPoint point)
{
  ArrastrarAgujas(nFlags, point);
  CDialog::OnLButtonDown(nFlags, point);
}

void CDlgRelojAnalogico::OnRButtonDown(UINT nFlags, CPoint point)
{
  ArrastrarAgujas(nFlags, point);
  CDialog::OnRButtonDown(nFlags, point);
}

void CDlgRelojAnalogico::ArrastrarAgujas(UINT nFlags, CPoint point)
{
  double dAlfa, dHora;
  int nHoras,nMinutos,nSegundos;

  if(m_nRadio > 10 && nFlags & (MK_LBUTTON)
        &&(point.x!=m_nCentroX||point.y!=m_nCentroY))
  {
```

```
        dAlfa = atan2(m_nCentroX-point.x,
                                                point.y-m_nCentroY);
        dHora = dAlfa/3.1416/2*12+6;

        if (nFlags & MK_CONTROL)
              dHora+=12;

        nSegundos = int (dHora*3600);
        if (nSegundos<0)
              nSegundos=0;
        else if (nSegundos>3600*24-1)
              nSegundos=3600*24-1;

        nHoras=nSegundos/3600;
        nSegundos %=3600;
        nMinutos = nSegundos/60;
        nSegundos %= 60;

        //pasarle la hora a la vista
        m_pVista->CambiarHora(nHoras,nMinutos,nSegundos);
        }
        else if(m_nRadio > 10 && nFlags & (MK_RBUTTON)
        &&(point.x!=m_nCentroX||point.y!=m_nCentroY))
        {
        dAlfa = atan2(m_nCentroX-point.x,-(m_nCentroY
                                                  - point.y));

        nHoras=m_Hora.GetHour();
        int nMinutosAntes=m_Hora.GetMinute();
        nSegundos =m_Hora.GetSecond();
        nMinutos=int (dAlfa/3.1416/2*60+30);

        if (nMinutos<0)
              nMinutos=0;
        else if (nMinutos>59)
              nMinutos = 59;
        //Tecla Control
        if(nMinutos>nMinutosAntes &&
                              nMinutos-nMinutosAntes>30)
              nHoras--;
        else if (nMinutos<nMinutosAntes &&
                              nMinutosAntes-nMinutos>30)
              nHoras++;
        if (nHoras==-1)
              nHoras=23;
        else if(nHoras==24)
              nHoras=0;

        //pasarle la hora a la vista
        m_pVista->CambiarHora(nHoras,nMinutos,nSegundos);
  }
}

BOOL CDlgRelojAnalogico::OnEraseBkgnd(CDC* pDC)
{
  return true;
// Devolver true indica que ya no hay que borrar nada

  // No llamamos a la implementación de la clase
  // base porque no queremos borrar el fondo.
  // return CDialog::OnEraseBkgnd(pDC);
}
```

// Archivo: EditHora.h

```
#if
!defined(AFX_EDITHORA_H__473AA3D6_218D_4E0A_8461_F3D1D13DB60A__INCLUDED_)
#define AFX_EDITHORA_H__473AA3D6_218D_4E0A_8461_F3D1D13DB60A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// EditHora.h : header file
//

/////////////////////////////////////////////////////////////////////////////
////
// CEditHora window

class CEditHora : public CEdit
{
// Construction
public:
  CEditHora();

// Attributes
public:
  void SetHora(int,int,int);
  bool GetHora(int &, int &, int &) const;
// Operations
public:

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

// Implementation
public:
  virtual ~CEditHora();

  // Generated message map functions
protected:
  //{{AFX_MSG(CEditHora)
  afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
  //}}AFX_MSG

  DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_EDITHORA_H__473AA3D6_218D_4E0A_8461_F3D1D13DB60A__INCLUDED_)
```

// Archivo: EditHora.cpp
// implementation file

```
#include "stdafx.h"
#include "Reloj.h"
#include "EditHora.h"

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

/////////////////////////////////////////////////////////////////////////////
////
// CEditHora

CEditHora::CEditHora()
{
}

CEditHora::~CEditHora()
{
}

BEGIN_MESSAGE_MAP(CEditHora, CEdit)
  //{{AFX_MSG_MAP(CEditHora)
  ON_WM_CHAR()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
////
// CEditHora message handlers

void CEditHora::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
  if ( (nChar < '0' || nChar > '9') && nChar != ':' && nChar != VK_BACK)
        MessageBeep(0xFFFFFFFF);
  else
        CEdit::OnChar(nChar, nRepCnt, nFlags);
}

void CEditHora::SetHora(int nHoras, int nMinutos, int nSegundos)
{
  CString sHora;
  sHora.Format("%02d:%02d:%02d",nHoras,nMinutos,nSegundos);
  SetWindowText(sHora);
}

bool CEditHora::GetHora(int &nHoras, int &nMinutos,
                                        int &nSegundos) const
{
  CString sHora;
  GetWindowText(sHora);
  int nDatosLeidos = sscanf(sHora, "%d:%d:%d",
                                  &nHoras, &nMinutos, &nSegundos);
  if (nDatosLeidos == 2)
        nSegundos = 0;
  else if (nDatosLeidos != 3)
        return false;

  if (nHoras < 0 || nHoras > 23 ||
        nMinutos < 0 || nMinutos > 59 ||
        nSegundos < 0 || nSegundos > 59)
        return false;
```

```
  return true;
}
```

// Archivo: MainFrm.h

```
#if
!defined(AFX_MAINFRM_H__59F296A8_2852_42AF_B16B_A4F2B1ACBFDC__INCLUDED_)
#define AFX_MAINFRM_H__59F296A8_2852_42AF_B16B_A4F2B1ACBFDC__INCLUDED_

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

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
  CMainFrame();
  DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CMainFrame)
  virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
  //}}AFX_VIRTUAL

// Implementation
public:
  virtual ~CMainFrame();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
  //{{AFX_MSG(CMainFrame)
        // NOTE - the ClassWizard will add and remove member functions
here.
        //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MAINFRM_H__59F296A8_2852_42AF_B16B_A4F2B1ACBFDC__INCLUDED_)
```

// Archivo: MainFrm.cpp
// implementation of the CMainFrame class

```
#include "stdafx.h"
#include "Reloj.h"

#include "MainFrm.h"

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

/////////////////////////////////////////////////////////////////////////////
////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
  //{{AFX_MSG_MAP(CMainFrame)
        // 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()

/////////////////////////////////////////////////////////////////////////////
////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
  // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
  if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
  // TODO: Modify the Window class or styles here by modifying
  //  the CREATESTRUCT cs
  cs.style &= ~(FWS_ADDTOTITLE|WS_THICKFRAME|WS_MAXIMIZEBOX);
  return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
  CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
  CFrameWnd::Dump(dc);
}
```

```
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
////
// CMainFrame message handlers
```

// Archivo: Reloj.h
// main header file for the RELOJ application

```
#if
!defined(AFX_RELOJ_H__A425FA6C_3A9C_414D_8917_4F95320BAFCE__INCLUDED_)
#define AFX_RELOJ_H__A425FA6C_3A9C_414D_8917_4F95320BAFCE__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

/////////////////////////////////////////////////////////////////////////////
////
// CRelojApp:
// See Reloj.cpp for the implementation of this class
//

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

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CRelojApp)
  public:
  virtual BOOL InitInstance();
  //}}AFX_VIRTUAL

// Implementation
  //{{AFX_MSG(CRelojApp)
  afx_msg void OnAppAbout();
  // NOTE - the ClassWizard will add and remove member functions here.
  //    DO NOT EDIT what you see in these blocks of generated code !
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};


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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RELOJ_H__A425FA6C_3A9C_414D_8917_4F95320BAFCE__INCLUDED_)
```

// Archivo: Reloj.cpp
// Defines the class behaviors for the application.

```
#include "stdafx.h"
#include "Reloj.h"

#include "MainFrm.h"
#include "RelojDoc.h"
#include "RelojView.h"

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

/////////////////////////////////////////////////////////////////////////////
////
// CRelojApp

BEGIN_MESSAGE_MAP(CRelojApp, CWinApp)
  //{{AFX_MSG_MAP(CRelojApp)
  ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
  // 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
  // Standard file based document commands
  ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
  ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
////
// CRelojApp construction

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

/////////////////////////////////////////////////////////////////////////////
////
// The one and only CRelojApp object

CRelojApp theApp;

/////////////////////////////////////////////////////////////////////////////
////
// CRelojApp initialization

BOOL CRelojApp::InitInstance()
{
  AfxEnableControlContainer();

  // Standard initialization
  // If you are not using these features and wish to reduce // the size of
your final executable, you should remove from // the following the
specific initialization routines you do // not need.

#ifdef _AFXDLL
  Enable3dControls();       // Call this when using MFC in a shared DLL
#else
  Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif
```

```
  // Change the registry key under which our settings are
// stored.TODO: You should modify this string to be
// something appropriate such as the name of your company or //
organization.

SetRegistryKey(_T("Local AppWizard-Generated Applications"));

  LoadStdProfileSettings(0);  // Load standard INI file options (including
MRU)

  // Register the application's document templates.  Document // templates
serve as the connection between documents,
// frame windows and views.

  CSingleDocTemplate* pDocTemplate;
  pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CRelojDoc),
        RUNTIME_CLASS(CMainFrame),  // main SDI frame window
        RUNTIME_CLASS(CRelojView));
  AddDocTemplate(pDocTemplate);

  // Parse command line for standard shell commands, DDE,
// file open

CCommandLineInfo cmdInfo;
  ParseCommandLine(cmdInfo);

  // Dispatch commands specified on the command line
  if (!ProcessShellCommand(cmdInfo))
        return FALSE;

  // The one and only window has been initialized, so show
 // and update it.
  m_pMainWnd->ShowWindow(SW_SHOW);
  m_pMainWnd->UpdateWindow();

  return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
  CAboutDlg();

// Dialog Data
  //{{AFX_DATA(CAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CAboutDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);
 // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  //{{AFX_MSG(CAboutDlg)
        // No message handlers
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
```

```
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CAboutDlg)
  //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CRelojApp::OnAppAbout()
{
  CAboutDlg aboutDlg;
  aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
////
// CRelojApp message handlers
```

// Archivo: RelojDoc.h

```
// RelojDoc.h : interface of the CRelojDoc class
//
/////////////////////////////////////////////////////////////////////////////
////

#if
!defined(AFX_RELOJDOC_H__071B50DB_6076_4CF1_BC28_A414169F435C__INCLUDED_)
#define AFX_RELOJDOC_H__071B50DB_6076_4CF1_BC28_A414169F435C__INCLUDED_

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


class CRelojDoc : public CDocument
{
protected: // create from serialization only
  CRelojDoc();
  DECLARE_DYNCREATE(CRelojDoc)

// Attributes
public:
  CObArray *m_pArrayZonas;
// Operations
public:
  bool m_bArrayModificado;

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CRelojDoc)
  public:
```

```
  virtual BOOL OnNewDocument();
  virtual void Serialize(CArchive& ar);
  virtual void DeleteContents();
  //}}AFX_VIRTUAL

// Implementation
public:
  virtual ~CRelojDoc();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
  //{{AFX_MSG(CRelojDoc)
        // NOTE - the ClassWizard will add and remove member functions
here.
        //    DO NOT EDIT what you see in these blocks of generated code
!
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RELOJDOC_H__071B50DB_6076_4CF1_BC28_A414169F435C__INCLUDED_)
```

// Archivo: RelojDoc.cpp
// implementation of the CRelojDoc class

```
#include "stdafx.h"
#include "Reloj.h"

#include "RelojDoc.h"

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

/////////////////////////////////////////////////////////////////////////////
////
// CRelojDoc

IMPLEMENT_DYNCREATE(CRelojDoc, CDocument)

BEGIN_MESSAGE_MAP(CRelojDoc, CDocument)
  //{{AFX_MSG_MAP(CRelojDoc)
  // 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()
```

```
/////////////////////////////////////////////////////////////////////////////
////
// CRelojDoc construction/destruction

CRelojDoc::CRelojDoc()
{
  // TODO: add one-time construction code here
  this->m_bArrayModificado = false;
  this->m_pArrayZonas = new CObArray[4];
}

CRelojDoc::~CRelojDoc()
{
  delete [] this->m_pArrayZonas;
}

BOOL CRelojDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
        return FALSE;

  CFile fiDatosReloj; // Si el archivo existe y se abre...

  if (fiDatosReloj.Open("reloj.dat", CFile::modeRead))
  {
  CArchive arDatosReloj(&fiDatosReloj, CArchive::load);

  arDatosReloj >> m_pArrayZonas;          // Seriar array (cargar)

  arDatosReloj.Close();                          // Cerrar
  fiDatosReloj.Close();
  }

  return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
////
// CRelojDoc serialization

void CRelojDoc::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
        // TODO: add storing code here
  }
  else
  {
        // TODO: add loading code here
  }
}

/////////////////////////////////////////////////////////////////////////////
////
// CRelojDoc diagnostics

#ifdef _DEBUG
void CRelojDoc::AssertValid() const
{
  CDocument::AssertValid();
}

void CRelojDoc::Dump(CDumpContext& dc) const
{
  CDocument::Dump(dc);
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
////
// CRelojDoc commands

void CRelojDoc::DeleteContents()
{
  // Guardar datos

  if (m_bArrayModificado)
        {
        CFile fiDatosReloj;       // Abrir

        fiDatosReloj.Open("reloj.dat",
                          CFile::modeCreate|CFile::modeWrite);
        CArchive arDatosReloj(&fiDatosReloj,
                                              CArchive::store);

        // Seriar array (guardar)
        arDatosReloj << m_pArrayZonas;

        arDatosReloj.Close();                 // Cerrar
        fiDatosReloj.Close();
        }

  // Borrar datos

  int i, nZonas = m_pArrayZonas->GetSize();

  for (i=0; i<nZonas; i++)
        delete m_pArrayZonas->GetAt(i);

  m_pArrayZonas->RemoveAll();

  CDocument::DeleteContents();
}
```

// Archivo: RelojView.h
// interface of the CRelojView class

```
#include "DlgRelojAnalogico.h"
#include "EditHora.h"

#if
!defined(AFX_RELOJVIEW_H__87469F91_2FCC_4A37_9499_2E34670EEF0C__INCLUDED_
)
#define AFX_RELOJVIEW_H__87469F91_2FCC_4A37_9499_2E34670EEF0C__INCLUDED_

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


class CRelojView : public CFormView
{
  CTimeSpan m_DesfaseHorario;
  CDlgRelojAnalogico *m_pDlgRelojAnalogico;
  UINT m_nIdTimer;
  CFont m_FuenteHora;
  CBrush m_BrochaFondoHora;
  COLORREF m_rgbTextoHora;
  COLORREF m_rgbFondoHora;
  bool m_bDespertadorActivado;
  int m_nZonas;
  int m_nZonaSeleccionada;
```

```
protected: // create from serialization only
  CRelojView();
  int UnidadesLogicas(int);
  DECLARE_DYNCREATE(CRelojView)

public:
  //{{AFX_DATA(CRelojView)
  enum { IDD = IDD_RELOJ_FORM };
  CButton      m_ctrlDespertadorActivado;
  CEditHora    m_ctrlHoraAlarma;
  CEdit  m_ctrlHora;
  //}}AFX_DATA

// Attributes
public:
  CRelojDoc* GetDocument();
  COLORREF m_ColorDeFondo;

// Operations
public:
  void CambiarHora(int,int,int);
// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CRelojView)
  public:
  virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);
// DDX/DDV support
  virtual void OnInitialUpdate();
// called first time after construct
  //}}AFX_VIRTUAL

// Implementation
public:
  virtual ~CRelojView();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:
void MostrarHoraActual();
afx_msg bool OnZona(UINT nID);

// Generated message map functions
protected:
  //{{AFX_MSG(CRelojView)
  afx_msg void OnDestroy();
  afx_msg void OnTimer(UINT nIDEvent);
  afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd,
                                                    UINT nCtlColor);
  afx_msg BOOL OnEraseBkgnd(CDC* pDC);
  afx_msg void OnOpcionesColordefondo();
  afx_msg void OnOpcionesFuente();
  afx_msg void OnUpdateOpcionesDespertadorActivar(
                                                    CCmdUI* pCmdUI);
  afx_msg void OnOpcionesDespertadorActivar();
  afx_msg void OnUpdateOpcionesDespertadorDesactivar(
                                                    CCmdUI* pCmdUI);
  afx_msg void OnOpcionesDespertadorDesactivar();
  afx_msg void OnOpcionesMostrarrelojanalgico();
  afx_msg void OnZonaAadir();
  afx_msg void OnZonaEliminar();
  afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
  afx_msg void OnUpdateOpcionesMostrarrelojanalgico(
                                                    CCmdUI* pCmdUI);
```

```
  afx_msg void OnDespertadorActivado();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in RelojView.cpp
inline CRelojDoc* CRelojView::GetDocument()
   { return (CRelojDoc*)m_pDocument; }
#endif

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RELOJVIEW_H__87469F91_2FCC_4A37_9499_2E34670EEF0C__INCLUDED_
)
```

// Archivo: RelojView.cpp
// implementation of the CRelojView class

```
#include "stdafx.h"
#include "Reloj.h"
#include "DlgDatosZonaHoraria.h"
#include "DlgEliminarZona.h"
#include "ZonaHoraria.h"
#include "RelojDoc.h"
#include "RelojView.h"

#define ID_BASE_ZONA (ID_ZONA_ELIMINAR+100)
#define NUM_MAX_ZONAS 4

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

/////////////////////////////////////////////////////////////////////////
////
// CRelojView

IMPLEMENT_DYNCREATE(CRelojView, CFormView)

BEGIN_MESSAGE_MAP(CRelojView, CFormView)
  //{{AFX_MSG_MAP(CRelojView)
  ON_WM_DESTROY()
  ON_WM_TIMER()
  ON_WM_CTLCOLOR()
  ON_WM_ERASEBKGND()
  ON_COMMAND(ID_OPCIONES_COLORDEFONDO, OnOpcionesColordefondo)
  ON_COMMAND(ID_OPCIONES_FUENTE, OnOpcionesFuente)
  ON_UPDATE_COMMAND_UI(ID_OPCIONES_DESPERTADOR_ACTIVAR,
OnUpdateOpcionesDespertadorActivar)
  ON_COMMAND(ID_OPCIONES_DESPERTADOR_ACTIVAR,
OnOpcionesDespertadorActivar)
  ON_UPDATE_COMMAND_UI(ID_OPCIONES_DESPERTADOR_DESACTIVAR,
OnUpdateOpcionesDespertadorDesactivar)
  ON_COMMAND(ID_OPCIONES_DESPERTADOR_DESACTIVAR,
OnOpcionesDespertadorDesactivar)
  ON_COMMAND(ID_OPCIONES_MOSTRARRELOJANALGICO,
OnOpcionesMostrarrelojanalgico)
```

```
  ON_COMMAND(ID_ZONA_AADIR, OnZonaAadir)
  ON_COMMAND(ID_ZONA_ELIMINAR, OnZonaEliminar)
  ON_WM_RBUTTONDOWN()
  ON_UPDATE_COMMAND_UI(ID_OPCIONES_MOSTRARRELOJANALGICO,
OnUpdateOpcionesMostrarrelojanalgico)
  ON_BN_CLICKED(IDC_DESPERTADOR_ACTIVADO, OnDespertadorActivado)
  //}}AFX_MSG_MAP
  ON_COMMAND_EX_RANGE(ID_BASE_ZONA, ID_BASE_ZONA + NUM_MAX_ZONAS - 1,
OnZona)
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
////
// CRelojView construction/destruction

CRelojView::CRelojView()
  : CFormView(CRelojView::IDD)
{
  m_DesfaseHorario = CTimeSpan(0,0,0,0);
  m_pDlgRelojAnalogico = NULL;
  m_rgbTextoHora = RGB(0,0,255);          //AZUL
  m_rgbFondoHora = RGB(255,255,0); //AMARILLO
  m_bDespertadorActivado = false;
  m_nZonas = 0;
  m_nZonaSeleccionada = -1;
  //{{AFX_DATA_INIT(CRelojView)
        // NOTE: the ClassWizard will add member initialization here
  //}}AFX_DATA_INIT
  // TODO: add construction code here

}

CRelojView::~CRelojView()
{
  delete m_pDlgRelojAnalogico;
}

void CRelojView::DoDataExchange(CDataExchange* pDX)
{
  CFormView::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CRelojView)
  DDX_Control(pDX, IDC_DESPERTADOR_ACTIVADO, m_ctrlDespertadorActivado);
  DDX_Control(pDX, IDC_HORA_ALARMA, m_ctrlHoraAlarma);
  DDX_Control(pDX, IDC_HORA, m_ctrlHora);
  //}}AFX_DATA_MAP
}

BOOL CRelojView::PreCreateWindow(CREATESTRUCT& cs)
{
  // TODO: Modify the Window class or styles here by modifying
  //  the CREATESTRUCT cs

  return CFormView::PreCreateWindow(cs);
}

void CRelojView::OnInitialUpdate()
{
  m_BrochaFondoHora.CreateSolidBrush(RGB(255,255,0));
  m_ColorDeFondo = ::GetSysColor(COLOR_3DFACE);

  CFormView::OnInitialUpdate();
  GetParentFrame()->RecalcLayout();
  ResizeParentToFit();
  m_pDlgRelojAnalogico = new CDlgRelojAnalogico(this);
  MostrarHoraActual();
  m_nIdTimer = SetTimer(ID_1SEGUNDO, 1000, NULL);

  LOGFONT lf;
```

```
 memset(&lf, 0, sizeof(LOGFONT));
 strncpy(lf.lfFaceName, "Times New Roman", LF_FACESIZE);
 lf.lfFaceName[LF_FACESIZE-1] = '\0';
 lf.lfHeight = -UnidadesLogicas(16);
 m_FuenteHora.CreateFontIndirect(&lf);
 m_ctrlHora.SetFont(&m_FuenteHora);

 CMenu *pMenu = GetParent()->GetMenu();
 pMenu = pMenu->GetSubMenu(1);

 if (m_bDespertadorActivado == 1)
    m_ctrlDespertadorActivado.SetCheck(1);
 else
    m_ctrlDespertadorActivado.SetCheck(0);

 CRelojDoc *pDoc = GetDocument();
 int i, nZonas = pDoc->m_pArrayZonas->GetSize();

 if(nZonas == NUM_MAX_ZONAS)
       pMenu->RemoveMenu(ID_ZONA_AADIR, MF_BYCOMMAND);

 if (nZonas > 0)
 {
 pMenu->AppendMenu(MF_SEPARATOR);

 CZonaHoraria *pZona;

 for (i=0; i<nZonas; i++)
       {
       pZona = (CZonaHoraria *) pDoc->m_pArrayZonas->GetAt(i);

       pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                                   ID_BASE_ZONA + i,
                                   pZona->m_sNombre);
       }
 }
 else
       pMenu->RemoveMenu(ID_ZONA_ELIMINAR, MF_BYCOMMAND);

 this->m_ctrlDespertadorActivado.SetCheck(m_bDespertadorActivado?1:0);
}

/////////////////////////////////////////////////////////////////////////////
////
// CRelojView diagnostics

#ifdef _DEBUG
void CRelojView::AssertValid() const
{
 CFormView::AssertValid();
}

void CRelojView::Dump(CDumpContext& dc) const
{
 CFormView::Dump(dc);
}

CRelojDoc* CRelojView::GetDocument()
// non-debug version is inline
{
 ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CRelojDoc)));
 return (CRelojDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
////
// CRelojView message handlers
```

```
void CRelojView::MostrarHoraActual()
{
  CTime Hora = CTime::GetCurrentTime() + m_DesfaseHorario;
  CString HoraTxt = Hora.Format("%H:%M:%S");
  SetDlgItemText(IDC_HORA, HoraTxt);

  // Mostrar la hora actual de la hora seleccionada

  if (m_nZonaSeleccionada != -1)
        {
        CZonaHoraria *pZona;
        pZona = (CZonaHoraria *)GetDocument()->
                    m_pArrayZonas->GetAt(m_nZonaSeleccionada);

        if (pZona->m_bPositivo)
              Hora += CTimeSpan(0, pZona->m_nHoras,
                                          pZona->m_nMinutos,
                                          pZona->m_nSegundos);
        else
              Hora -= CTimeSpan(0, pZona->m_nHoras,
                                          pZona->m_nMinutos,
                                          pZona->m_nSegundos);

        HoraTxt = Hora.Format("%H:%M:%S");
        SetDlgItemText(IDC_HORA_ZONA, HoraTxt);
        HoraTxt = "Hora en " + pZona->m_sNombre + ":";
        SetDlgItemText(IDC_STATIC_HORA_ZONA, HoraTxt);
        }
  else
        {
        SetDlgItemText(IDC_HORA_ZONA, "");
        SetDlgItemText(IDC_STATIC_HORA_ZONA, "Hora en...");
        }

  m_pDlgRelojAnalogico->SetHora(Hora);
  m_pDlgRelojAnalogico->Invalidate();
}

void CRelojView::OnDestroy()
{
  if (m_nIdTimer)
        KillTimer(ID_1SEGUNDO);
  CFormView::OnDestroy();
}

void CRelojView::OnTimer(UINT nIDEvent)
{
  if (IsWindow(m_hWnd) && nIDEvent == ID_1SEGUNDO)
        {
        MostrarHoraActual();
        int nHorasAlarma, nMinutosAlarma, nSegundosAlarma,nSegundos;

        if (m_ctrlHoraAlarma.GetHora(nHorasAlarma,
                          nMinutosAlarma, nSegundosAlarma))
              {
              nSegundosAlarma += nMinutosAlarma * 60 +
                                          nHorasAlarma * 3600;
              CTime Hora = CTime::GetCurrentTime() +
                                          m_DesfaseHorario;
              nSegundos = Hora.GetSecond() + Hora.GetMinute()
                                * 60 + Hora.GetHour() * 3600;
              if (m_bDespertadorActivado)
                    {
                    if (nSegundos < nSegundosAlarma)
                          {
                          if (nSegundosAlarma - nSegundos >
                                                86400 - 300)
```

```
                                    MessageBeep(0xFFFFFFFF);
                              }
                        else
                              {
                              if (nSegundos-nSegundosAlarma < 300)
                                    MessageBeep(0xFFFFFFFF);
                              }
                        }
                  }
            }

  CFormView::OnTimer(nIDEvent);
}

void CRelojView::CambiarHora(int nHora,int nMinuto,int nSegundo)
{
  CTime Hora = CTime::GetCurrentTime();
  CTime HoraNueva(Hora.GetYear(), Hora.GetMonth(),
                            Hora.GetDay(), nHora, nMinuto,
                                                nSegundo);
  m_DesfaseHorario = HoraNueva - Hora;
  MostrarHoraActual();
}

int CRelojView::UnidadesLogicas(int nPuntosAltFuente)
{
CClientDC dc(this);
POINT pt;
pt.x = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSX) / 72;
pt.y = nPuntosAltFuente * dc.GetDeviceCaps(LOGPIXELSY) / 72;
dc.DPtoLP(&pt);
return pt.y;
}

HBRUSH CRelojView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
  HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);
  if (pWnd->GetDlgCtrlID() == IDC_HORA)
        {
        pDC->SetBkColor(m_rgbFondoHora);
        pDC->SetTextColor(m_rgbTextoHora);
        return (HBRUSH) m_BrochaFondoHora.GetSafeHandle();
        }
  return hbr;
}

BOOL CRelojView::OnEraseBkgnd(CDC* pDC)
{
  CRect rect;
  CBrush brColorDeFondo (m_ColorDeFondo);
  CBrush *pPincelViejo = pDC->SelectObject(&brColorDeFondo);
  pDC->GetClipBox(&rect);
  pDC->PatBlt(rect.left, rect.top, rect.Width(),
                      rect.Height(), PATCOPY);
  pDC->SelectObject(pPincelViejo);

  return true;
  //return CFormView::OnEraseBkgnd(pDC);
}

void CRelojView::OnOpcionesColordefondo()
{
CColorDialog dlgColor(m_rgbFondoHora, CC_RGBINIT);

if (dlgColor.DoModal() == IDOK)
  {
  m_BrochaFondoHora.DeleteObject();
  m_rgbFondoHora = dlgColor.GetColor();
```

```
  m_BrochaFondoHora.CreateSolidBrush(m_rgbFondoHora);
  Invalidate();//llama a OnEraseBkgnd y a OnPaint
  }
}

void CRelojView::OnOpcionesFuente()
{
LOGFONT lf;
CFontDialog dlgFuente;
dlgFuente.m_cf.lpLogFont = &lf;

if (dlgFuente.DoModal() == IDOK)
  {
  m_FuenteHora.DeleteObject();
  m_FuenteHora.CreateFontIndirect(&lf);
  m_rgbTextoHora = dlgFuente.GetColor();
  GetDlgItem(IDC_HORA)->SetFont(&m_FuenteHora);
  Invalidate();
  }
}

void CRelojView::OnUpdateOpcionesDespertadorActivar(CCmdUI* pCmdUI)
{
  if (m_bDespertadorActivado == true)
        pCmdUI->Enable(false);
  else
        pCmdUI->Enable(true);
}

void CRelojView::OnOpcionesDespertadorActivar()
{
  m_bDespertadorActivado = !m_bDespertadorActivado;
  if (m_bDespertadorActivado == 1)
      m_ctrlDespertadorActivado.SetCheck(1);
  else
        m_ctrlDespertadorActivado.SetCheck(0);
}

void CRelojView::OnUpdateOpcionesDespertadorDesactivar(CCmdUI* pCmdUI)
{
  if (m_bDespertadorActivado == true)
        pCmdUI->Enable(true);
  else
        pCmdUI->Enable(false);
}

void CRelojView::OnOpcionesDespertadorDesactivar()
{
  m_bDespertadorActivado = !m_bDespertadorActivado;
  if (m_bDespertadorActivado == 1)
      m_ctrlDespertadorActivado.SetCheck(1);
  else
     m_ctrlDespertadorActivado.SetCheck(0);
}

void CRelojView::OnOpcionesMostrarrelojanalgico()
{
  if (m_pDlgRelojAnalogico->IsWindowVisible())
        m_pDlgRelojAnalogico->ShowWindow(SW_HIDE);
  else
        m_pDlgRelojAnalogico->ShowWindow(SW_SHOW);
}

void CRelojView::OnZonaAadir()
{
  CDlgDatosZonaHoraria dlg;

  if (dlg.DoModal() != IDOK)
```

```
        return;

int i, nZonas = GetDocument()->m_pArrayZonas->GetSize();

CZonaHoraria *pZona;

for (i=0; i<nZonas; i++)  // No puede haber duplicados
{
        pZona = (CZonaHoraria *) GetDocument()->
                                        m_pArrayZonas->GetAt(i);

        if (pZona->m_sNombre == dlg.m_sNombreZona)
        {
        AfxMessageBox("Ya existe una zona con ese nombre",
                                        MB_OK|MB_ICONEXCLAMATION);
        return;
        }
}

CMenu *pMenu = GetParent()->GetMenu();
//Primer submenú "Zona"

pMenu = pMenu->GetSubMenu(1);

// Si la que se añade es la primera zona, hay
// que añadir al submenú "Zona" la opción "Eliminar" y
// un separador. ...

if (nZonas == 0)
        {
        pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                    ID_ZONA_ELIMINAR, "&Eliminar");
        pMenu->AppendMenu(MF_SEPARATOR);
        }

pZona = new CZonaHoraria;

        // Almacenar los datos en el objeto

pZona->m_sNombre = dlg.m_sNombreZona;
pZona->m_nHoras = dlg.m_Horas;
pZona->m_nMinutos = dlg.m_Minutos;
pZona->m_nSegundos = dlg.m_Segundos;

if (dlg.m_nSigno == 0)
    pZona->m_bPositivo = true;
else
        pZona->m_bPositivo = false;

GetDocument()->m_pArrayZonas->Add(pZona);

// Añadimos la nueva zona al menú

pMenu->AppendMenu(MF_ENABLED|MF_STRING,
                            ID_BASE_ZONA + nZonas, pZona->
                            m_sNombre);

// Numero de la ultima zona añadida

m_nZonaSeleccionada = nZonas;

// Incrementamos el número de zonas

nZonas++;
GetDocument()->m_pArrayZonas->SetSize(nZonas);

// Si se alcanza el número máximo de zonas, eliminar del
// submenú "Zona" la opción "Añadir"
```

```
 if (nZonas == NUM_MAX_ZONAS)
       pMenu->RemoveMenu(ID_ZONA_AADIR, MF_BYCOMMAND);

 GetDocument()->m_bArrayModificado = true;
}

void CRelojView::OnZonaEliminar()
{
 // Almacena el nombre de un elemento del array
 CZonaHoraria *pZona;

 CMenu *pMenu = GetParent()->GetMenu();
 // Menú Principal

 pMenu = pMenu->GetSubMenu(1);
 // Primer submenú "Zona"

 // Obtener numero de zonas

 int i, k, nZonas = GetDocument()->m_pArrayZonas->GetSize();

 // Llamar a DoModal para mostrar el objeto dlg

 CDlgEliminarZona dlg(this);

 if (dlg.DoModal() != IDOK)
       return;

 // Averiguar cual es la zona seleccionada por el usuario
 // y comprobar que existe.

 for (i=0; i < nZonas; i++)
 {
       pZona = (CZonaHoraria *)GetDocument()->m_pArrayZonas->GetAt(i);
       if (dlg.m_sZonaSeleccionada == pZona->m_sNombre)
             break;
 }

 if ( i == nZonas )
 {
       AfxMessageBox("La zona no existe", MB_ICONINFORMATION);
       return;
 }

 // Si el numero de zonas es el maximo, al quitar una, hay
 // que añadir al submenu "Zona" la orden "Añadir"

 if (nZonas == NUM_MAX_ZONAS)
       pMenu->InsertMenu(0,
                   MF_BYPOSITION|MF_ENABLED|MF_STRING,
                        ID_ZONA_AADIR, "&Añadir");

 // Eliminar del submenu "Zona" todas las opciones desde
 // la correspondiente a la zona que hay que eliminar.
 // Despues hay que volver a añadirlas (todas execepto la
 // que estamos eliminando) con sus nuevos identificadores.
 // Hay que hacer lo mismo con el array de zonas del doc.

 GetDocument()->m_pArrayZonas->RemoveAt(i);

 for (k = i; k < nZonas; k++)
       pMenu->RemoveMenu(ID_BASE_ZONA + k, MF_BYCOMMAND);

 for (k = i; k < nZonas - 1; k++)
       pMenu->AppendMenu(MF_STRING, (UINT) ID_BASE_ZONA
             + k, ((CZonaHoraria *)GetDocument()->
                   m_pArrayZonas->GetAt(k))->m_sNombre);
```

```
  // Decrementar el numero de zonas

  nZonas--;

  // Actualizar el numero de la ultima zona seleccionada por el usuario

  if (m_nZonaSeleccionada >= nZonas)
        m_nZonaSeleccionada = nZonas - 1;

  GetDocument()->m_pArrayZonas->SetSize(nZonas);

  // Si ya no quedan zonas, hay que quitar del submenu
  // "Zonas" el separador y la opcion "Eliminar"

  if (nZonas == 0)
        {
        pMenu->RemoveMenu(2, MF_BYPOSITION);
        pMenu->RemoveMenu(ID_ZONA_ELIMINAR, MF_BYCOMMAND);
        }

  GetDocument()->m_bArrayModificado = true;
}

bool CRelojView::OnZona(UINT nID)
{
  CMenu *pMenu = GetParent()->GetMenu();
  pMenu = pMenu->GetSubMenu(1);

  CString sNombre;

  pMenu->GetMenuString(nID, sNombre, MF_BYCOMMAND);

  AfxMessageBox(sNombre, MB_OK | MB_ICONINFORMATION);

  m_nZonaSeleccionada = nID - ID_BASE_ZONA;

  return true;
}

void CRelojView::OnRButtonDown(UINT nFlags, CPoint point)
{
  CMenu *pMenu = GetParent()->GetMenu();

  if (nFlags & MK_CONTROL )
  {
        pMenu = pMenu->GetSubMenu(2);
        pMenu = pMenu->GetSubMenu(5);

        if (m_bDespertadorActivado)
        {
        pMenu->EnableMenuItem(
              ID_OPCIONES_DESPERTADOR_ACTIVAR, MF_GRAYED);


        pMenu->EnableMenuItem(
              ID_OPCIONES_DESPERTADOR_DESACTIVAR, MF_ENABLED);
        }
        else
        {
        pMenu->EnableMenuItem(
              ID_OPCIONES_DESPERTADOR_DESACTIVAR, MF_GRAYED);
        pMenu->EnableMenuItem(
              ID_OPCIONES_DESPERTADOR_ACTIVAR, MF_ENABLED);
        }
  }
  else
        pMenu = pMenu->GetSubMenu(1);
```

```
  ClientToScreen(&point);
  pMenu->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,
                            point.x, point.y, this, NULL);

  CFormView::OnRButtonDown(nFlags, point);

}

void CRelojView::OnUpdateOpcionesMostrarrelojanalgico(CCmdUI* pCmdUI)
{
  if (m_pDlgRelojAnalogico->IsWindowVisible())
        pCmdUI->SetText("Ocultar Reloj Analógico");
  else
        pCmdUI->SetText("Mostrar Reloj Analógico");
}

void CRelojView::OnDespertadorActivado()
{
  if (m_ctrlDespertadorActivado.GetCheck())
        m_bDespertadorActivado = !m_bDespertadorActivado;
  else
        m_bDespertadorActivado = !m_bDespertadorActivado;
}
```

// Archivo: Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Reloj.rc
//
#define IDOK2                           3
#define IDCANCEL2                       4
#define IDD_ABOUTBOX                    100
#define IDD_RELOJ_FORM                  101
#define ID_1SEGUNDO                     101
#define IDR_MAINFRAME                   128
#define IDR_RELOJTYPE                   129
#define IDD_RELOJ_ANALOGICO             130
#define IDD_DATOS_ZONA_HORARIA          131
#define IDD_ELIMINAR_ZONA_HORARIA       133
#define IDC_HORA                        1000
#define IDC_HORA_ALARMA                 1006
#define IDC_NOMBRE_ZONA                 1007
#define IDC_DIFERENCIA_HORARIA          1008
#define IDC_RADIO_POSITIVO              1009
#define IDC_RADIO_NEGATIVO              1010
#define IDC_PROBAR                      1011
#define IDC_DESPERTADOR_ACTIVADO        1012
#define IDC_LISTA_ZONAS                 1013
#define IDC_EDIT1                       1014
#define IDC_SIGNO                       1014
#define IDC_HORA_ZONA                   1014
#define IDC_STATIC_DIFERENCIA_HORARIA   1015
#define IDC_STATIC_SIGNO                1016
#define IDC_ABRIR_ELIMINAR              1017
#define IDC_STATIC_MARCO1               1018
#define IDC_STATIC_MARCO2               1019
#define IDC_STATIC_HORA_ZONA            1020
#define IDC_STATIC_HORA_ZONA2           1021
#define ID_OPCIONES_COLORDEFONDO        32776
#define ID_OPCIONES_FUENTE              32777
#define ID_OPCIONES_DESPERTADOR_ACTIVAR 32778
#define ID_OPCIONES_DESPERTADOR_DESACTIVAR 32779
#define ID_OPCIONES_MOSTRARRELOJANALGICO 32780
```

```
#define ID_ZONA_AADIR                   32781
#define ID_ZONA_ELIMINAR                32782

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        134
#define _APS_NEXT_COMMAND_VALUE         32783
#define _APS_NEXT_CONTROL_VALUE         1021
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

// Archivo: StdAfx.h

```
// stdafx.h : include file for standard system include
// files, or project specific include files that are used
// frequently, but are changed infrequently

#if
!defined(AFX_STDAFX_H__BCBC8FFE_26E5_464D_BF63_4162FA250E4D__INCLUDED_)
#define AFX_STDAFX_H__BCBC8FFE_26E5_464D_BF63_4162FA250E4D__INCLUDED_

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

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common
Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__BCBC8FFE_26E5_464D_BF63_4162FA250E4D__INCLUDED_)
```

// Archivo: StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard // includes
// Reloj.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

// Archivo: ZonaHoraria.h

```
#if
!defined(AFX_ZONAHORARIA_H__E3C7C08F_12E2_484B_ABBD_6CE1F5926D6D__INCLUDE
D_)
#define
AFX_ZONAHORARIA_H__E3C7C08F_12E2_484B_ABBD_6CE1F5926D6D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ZonaHoraria.h : header file
//

/////////////////////////////////////////////////////////////////////////
////
// CZonaHoraria document

class CZonaHoraria : public CDocument
{
protected:
  DECLARE_SERIAL(CZonaHoraria)

// Attributes
public:
  int m_nHoras;
  int m_nMinutos;
  int m_nSegundos;
  bool m_bPositivo;
  CString m_sNombre;
  CZonaHoraria();
// Operations
public:

// Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CZonaHoraria)
  public:
  // overridden for document i/o
  virtual void Serialize(CArchive& ar);
  protected:
  virtual BOOL OnNewDocument();
  //}}AFX_VIRTUAL

// Implementation
public:
  virtual ~CZonaHoraria();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

  // Generated message map functions
protected:
  //{{AFX_MSG(CZonaHoraria)
        // NOTE - the ClassWizard will add and remove member functions
here.
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_ZONAHORARIA_H__E3C7C08F_12E2_484B_ABBD_6CE1F5926D6D__INCLUDE
D_)
```

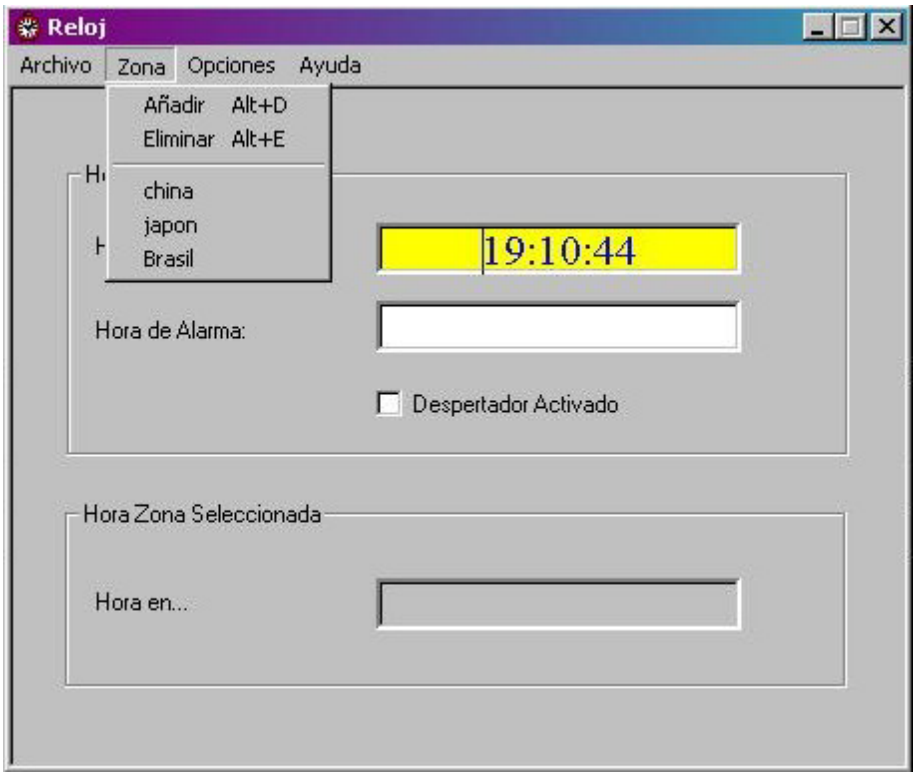
// Archivo: ZonaHoraria.cpp
// implementation file

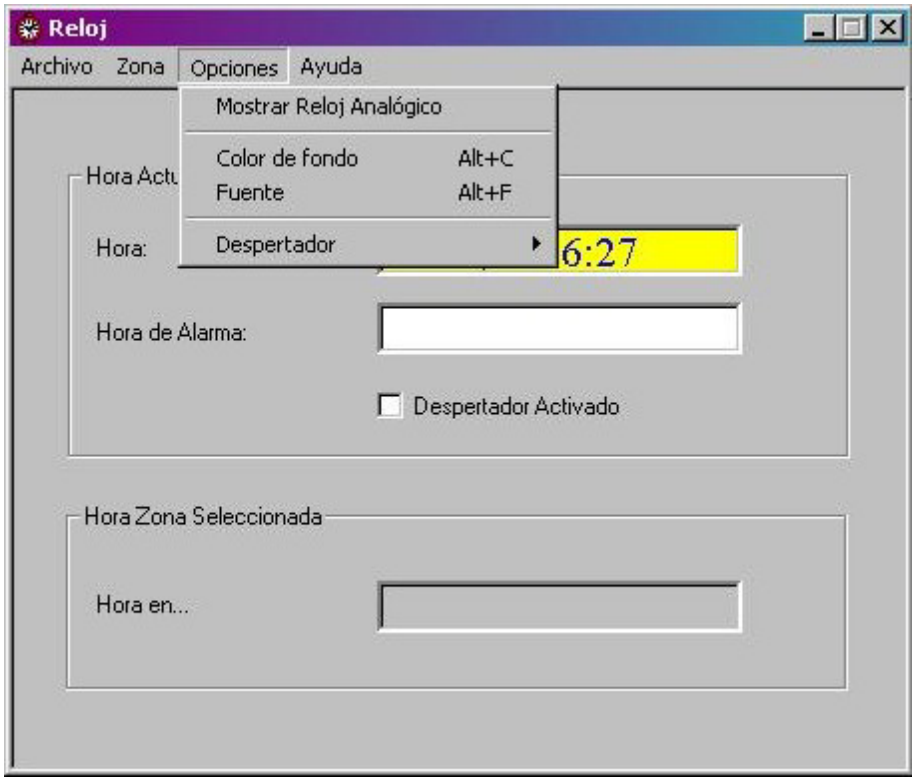
```
#include "stdafx.h"
#include "Reloj.h"
#include "ZonaHoraria.h"

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

/////////////////////////////////////////////////////////////////////////
////
// CZonaHoraria

IMPLEMENT_SERIAL(CZonaHoraria, CDocument,1)

CZonaHoraria::CZonaHoraria()
{
}

BOOL CZonaHoraria::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
        return FALSE;
  return TRUE;
}

CZonaHoraria::~CZonaHoraria()
{
}

BEGIN_MESSAGE_MAP(CZonaHoraria, CDocument)
  //{{AFX_MSG_MAP(CZonaHoraria)
        // NOTE - the ClassWizard will add and remove mapping macros
here.
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
////
// CZonaHoraria diagnostics

#ifdef _DEBUG
void CZonaHoraria::AssertValid() const
{
  CDocument::AssertValid();
}

void CZonaHoraria::Dump(CDumpContext& dc) const
{
  CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
////
// CZonaHoraria serialization
```

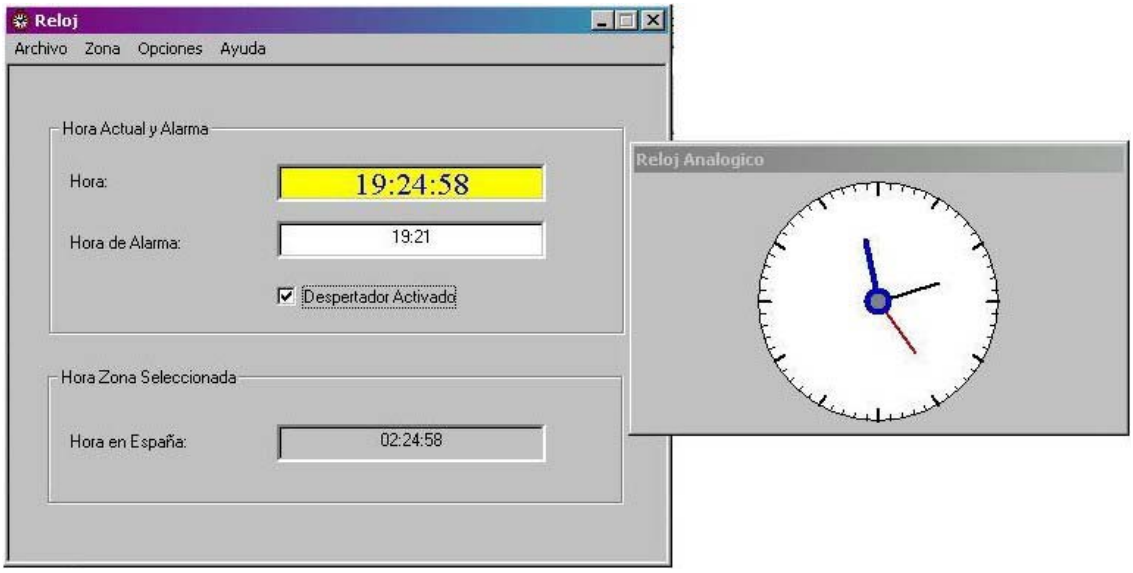
```
void CZonaHoraria::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
  ar << m_sNombre
     << int(m_bPositivo)
     << m_nHoras <<m_nMinutos << m_nSegundos;
  }
  else
  {
  int nPositivo;

  ar >> m_sNombre
     >> nPositivo
     >> m_nHoras >> m_nMinutos >> m_nSegundos;

  this->m_bPositivo = nPositivo ? true : false;
  }
}

/////////////////////////////////////////////////////////////////////////////
////
// CZonaHoraria commands
```

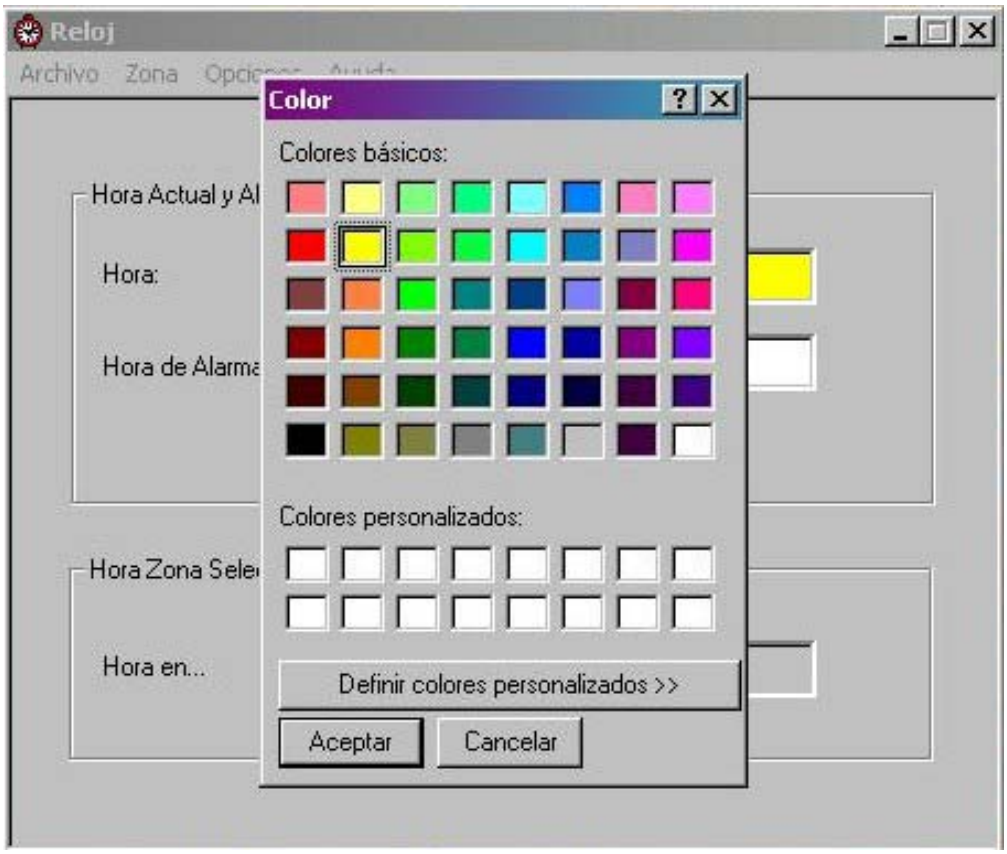
4.8.2 Interfaz gráfica de la aplicación

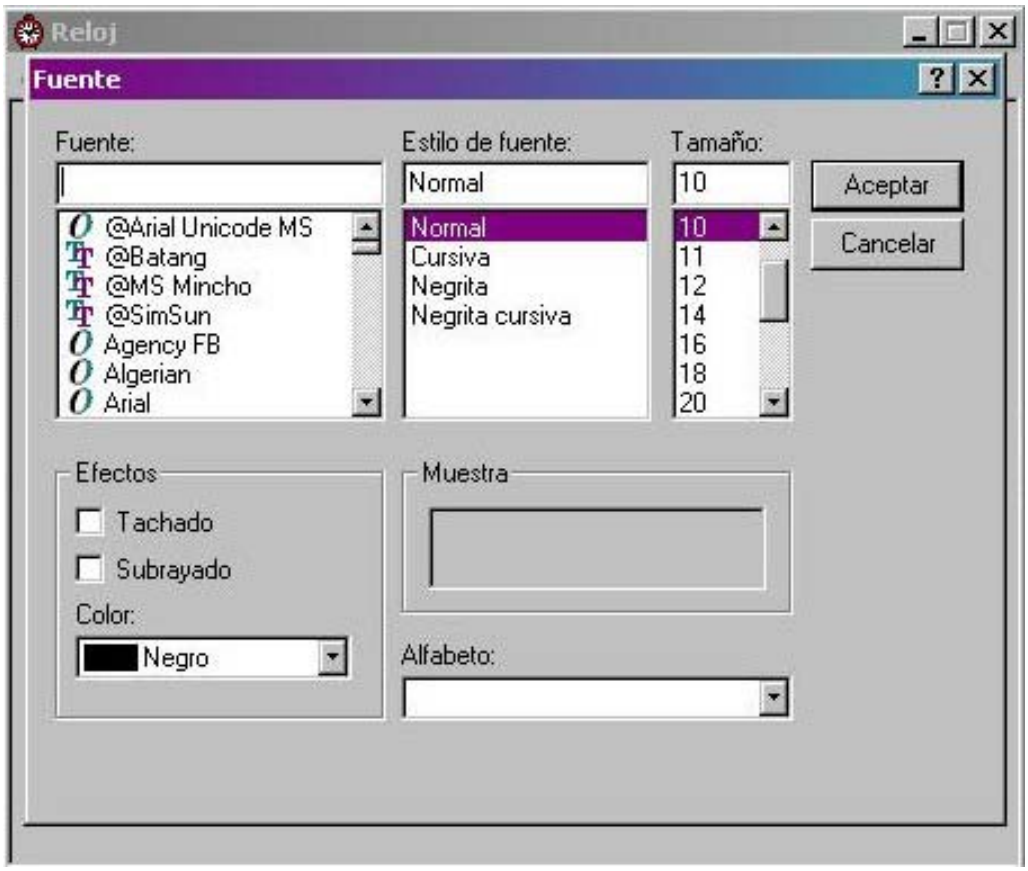
A continuación se presentan una serie de imágenes de la aplicación del ***"Reloj"***:


Reloj
Archivo
Zona
Opciones
Ayuda
Añadir Alt+D
Eliminar Alt+E
china
japon
Brasil
19:10:44
Hora de Alarma:
Despertador Activado
Hora Zona Seleccionada
Hora en...


Reloj
Archivo
Zona
Opciones
Ayuda
Mostrar Reloj Analógico
Color de fondo Alt+C
Fuente Alt+F
Despertador
Hora:
6:27
Hora de Alarma:
Despertador Activado
Hora Zona Seleccionada
Hora en...


Reloj
Archivo Zona Opciones Ayuda
Hora Actual y Alarma
Hora:
19:24:58
Hora de Alarma:
19:21
Despertador Activado
Hora Zona Seleccionada
Hora en España:
02:24:58
Reloj Analogico


Reloj
Archivo Zona
Color
Colores básicos:
Colores personalizados:
Definir colores personalizados >>
Aceptar
Cancelar
Hora Actual y Al
Hora:
Hora de Alarma
Hora Zona Sele
Hora en...


Reloj
Fuente
Fuente:
Estilo de fuente:
Tamaño:
Normal
10
Aceptar
Cancelar
@Arial Unicode MS
@Batang
@MS Mincho
@SimSun
Agency FB
Algerian
Arial
Normal
Cursiva
Negrita
Negrita cursiva
10
11
12
14
16
18
20
Efectos
Tachado
Subrayado
Color:
Negro
Muestra
Alfabeto:

ANEXO: EJERCICIOS

ANEXO - EJERCICIOS

En la siguiente sección se proponen una serie de ejercicios para que los estudiantes los realicen una vez que se ha desarrollado un determinado tema del contenido de **"Programación Visual"**. Tienen como objetivo afianzar los conocimientos teóricos adquiridos en clase en cada uno de los temas de esta asignatura. Estos ejercicios han sido recopilados por el profesor de la asignatura para motivar a los estudiantes a la resolución de problemas por parte de los alumnos.

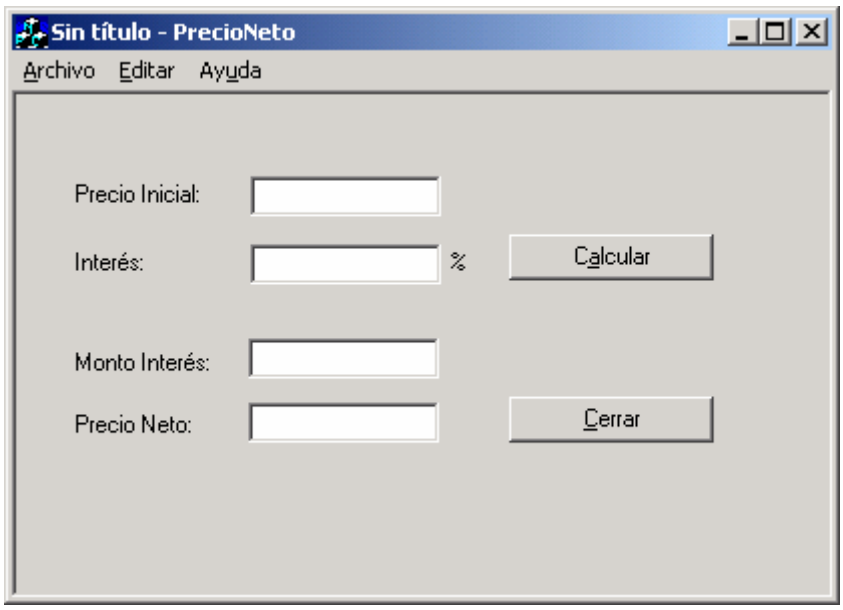
La solución de los mismos podrá ser discutida en clase con el profesor, de tal forma que las dudas que surjan puedan ser aclaradas; además se pretende con esto hacer una clase más dinámica y participativa por parte de los alumnos donde expongan sus dudas y/o puntos de vista.

CÁLCULO DEL PRECIO NETO

En este ejercicio se pretende introducir al alumno en la creación de interfaces gráficas y el manejo de eventos en Visual C++.

Introducción

En este ejercicio se desea crear una pequeña aplicación para realizar el cálculo del precio neto a partir de un precio inicial y de una tasa de interés (dada en %). La interfaz gráfica debe tener el aspecto siguiente:



El usuario debe proporcionar el precio inicial y el interés (en porcentaje); la tasa de interés, el monto y el precio neto se calcularán con la fórmulas siguientes:

```
TasaInteres = Tasa / 100;

MontoInteres = PrecioInicial * TasaInteres;

PrecioNeto = PrecioInicial + MontoInteres;
```

Estos cálculos deben hacerse al pulsar el botón “Calcular” y deben aparecer los resultados en las correspondientes cajas de texto del monto y el precio neto.

Iniciando la aplicación

1. Inicie Microsoft Visual C++

2. En el menú principal, clic en File -> New

3. Clic en **MFC AppWizard (exe)**

4. En la caja de texto del nombre del proyecto ponga “PrecioNeto”.

5. En las opciones siguientes, seleccione las opciones por defecto. No olvide derivar la clase CPrecioNetoView de CformView.

Diseñando la aplicación

1. Dibuje los controles que forman la interfaz gráfica.

2. Ponga los siguientes identificadores a las cajas de texto:

1	IDC_PRECIO_INICIAL
2	IDC_TASA_INTERES
3	IDC_MONTO_INTERES
4	IDC_PRECIO_NETO

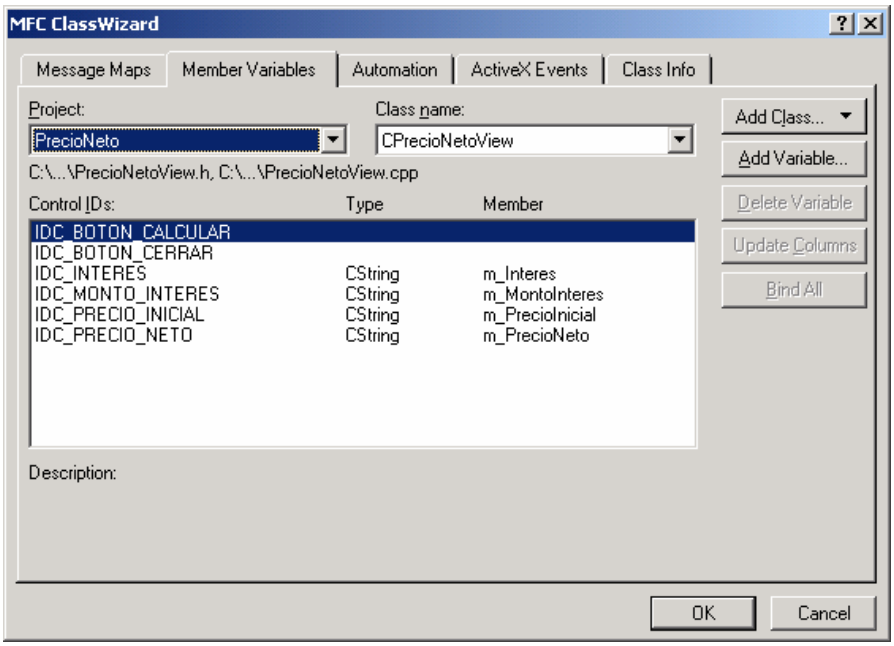
3. Salve la aplicación.

Programando la aplicación

1. Primero debemos asociar cada una de las cajas de texto con variables miembro de tipo CString.

IDC_PRECIO_INICIAL	m_PrecioInicial
IDC_TASA_INTERES	m_Interes
IDC_MONTO_INTERES	m_MontoInteres
IDC_PRECIO_NETO	m_PrecioNeto

La siguiente figura muestra el paso anterior:



2. A continuación cree el evento BN_CLICKED para el botón “Calcular” e implemente el método correspondiente.

3. Repita el paso anterior pero ahora para el botón “Cerrar”.

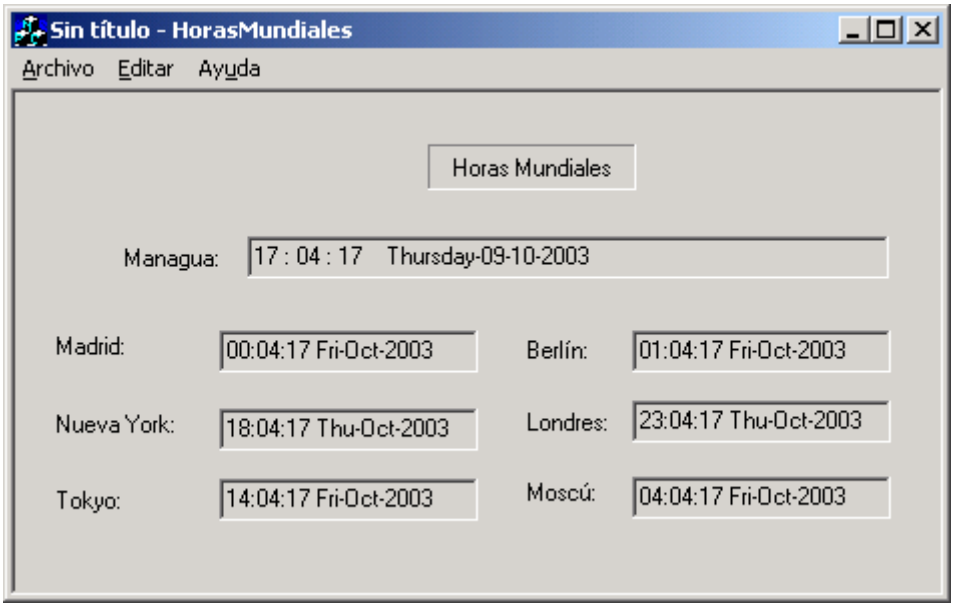
4. Compilar y ejecutar para comprobar los resultados.

HORA EN VARIAS CIUDADES DEL MUNDO

En este ejercicio se pretende que el alumno aprenda a utilizar las clases para manipular la hora y fecha: **CTime**.

Introducción

Se desea crear una pequeña aplicación para visualizar la hora correspondiente a varias ciudades del mundo. La interfaz gráfica debe tener el aspecto siguiente:



Como puede observar en la figura anterior, la interfaz gráfica está compuesta por ocho etiquetas y siete cajas de texto. Las cajas de texto tienen activada la propiedad “Read Only” con lo cual el usuario no podrá modificar el contenido de las mismas; el objetivo es mostrar en ellas la hora correspondiente a ese país.

Para hacer el cálculo de las horas tomaremos como referencia por supuesto la hora de Managua. La tabla siguiente muestra la diferencia de horas entre Managua y las otras ciudades:

Ciudad	*Diferencia (en horas)*
Managua	-
Madrid	+7
Nueva York	+1
Tokyo	+21
Berlín	+8
Londres	+6
Moscú	+11

Iniciando la aplicación

1. Inicie Microsoft Visual C++
2. En el menú principal, clic en File -> **New**
3. Clic en **MFC AppWizard (exe)**
4. En la caja de texto del nombre del proyecto ponga “HorasMundiales”
5. Elija la carpeta donde desee guardar los archivos que forman parte del proyecto
6. En el **“paso 1 de 6”** seleccione la opción “Single Document”, esto significa que se trabajará con una aplicación SDI (tendrá solamente una vista)
7. En el **“paso 2 de 6”** se le pregunta si su aplicación tendrá soporte a bases de datos. Por lo tanto, seleccionaremos la opción “None”
8. En el **“paso 3 de 6”** escoja la opción por defecto
9. En el **“paso 4 de 6”** seleccione solamente la opción 3D Controls
10. En el **“paso 5 de 6”** escoja las opciones por defecto
11. En el **“paso 6 de 6”** no olvide escoger como clase base a la clase **CFormView** esto me permitirá dibujar controles en la vista

Diseñando la aplicación

1. Dibuje los controles que forman la interfaz gráfica
2. Ponga los siguientes identificadores a las cajas de texto:

1	IDC_HORA_MANAGUA
2	IDC_HORA_MADRID
3	IDC_HORA_NUEVA_YORK
4	IDC_HORA_TOKIO
5	IDC_HORA_BERLIN
6	IDC_HORA_LONDRES
7	IDC_HORA_MOSCU

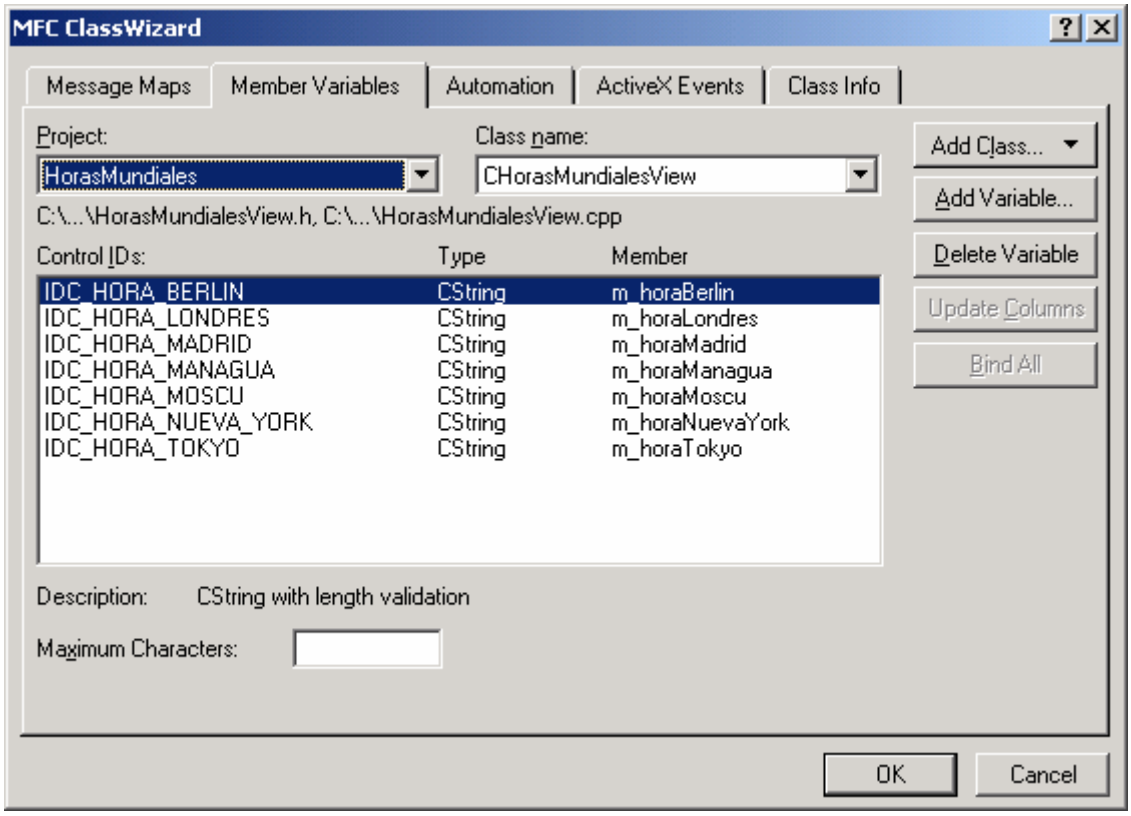
4. Salve la aplicación

Programando la aplicación

1. Primero debemos asociar cada una de las cajas de texto con variables miembro de tipo CString.

IDC_HORA_MANAGUA	m_nhoraManagua
IDC_HORA_MADRID	m_nhoraMadrid
IDC_HORA_NUEVA_YORK	m_nhoraNuevaYork
IDC_HORA_TOKIO	m_nhoraTokyo
IDC_HORA_BERLIN	m_nhoraBerlin
IDC_HORA_LONDRES	m_nhoraLondres
IDC_HORA_MOSCU	m_nhoraMoscu

La siguiente figura muestra el paso anterior:



2. A continuación deberemos incluir los identificadores que serán asociados a las variables que funcionarán como temporizadores. Para ejecutar este paso, vaya al menú ***View***, luego haga clic en ***Resource Symbols*** y con el botón ***New*** añada los identificadores siguientes:

ID_TIMER_MANAGUA
ID_TIMER_MADRID
ID_TIMER_NY
ID_TIMER_TOKIO
ID_TIMER_BERLIN
ID_TIMER_LONDRES
ID_TIMER_MOSCU

3. Edite la clase ***CHorasMundialesView*** y añada en la declaración de la clase las variables miembro privadas siguientes que se usarán para hacer referencia a cada temporizador utilizado:

UINT m_nTimerManagua, m_nTimerMadrid, m_nTimerNY, m_nTimerTokyo, m_nTimerBerlin, m_nTimerLondres, m_nTimerMoscu;

4. Ahora debemos editar la función miembro ***OnInitialUpdate()*** recuerde que esta función se llama al arrancar la vista. Al final de la función añadir la línea siguiente:

this->m_nTimerManagua=SetTimer(ID_TIMER_MANAGUA, 1000, NULL);

Con esta línea de código hemos asociado el identificador con la variable miembro privado. El valor 1000 significa 1 segundo. Se debe hacer lo mismo para los otros casos!

5. A continuación debemos añadir a la clase ***CHorasMundialesView*** una función llamada ***MostrarHoraActual()*** la cual se encargará de mostrar en la correspondiente caja de texto la hora de Managua (que será la hora actual del sistema). Edite el código de esta función; observe cuidadosamente en la primera imagen el formato que debe tener la impresión de la hora y la fecha

6. Una vez editado el código de la función anterior, mándela a llamar desde la función ***OnInitialUpdate()*** al final del código de la función para que la hora sea mostrada al iniciar la vista

7. Como siguiente paso, añadiremos a la clase ***CHorasMundialesView*** una función miembro llamada ***MostrarHoraCiudad(int diferencia)*** la que se encargará de mostrar la hora de otra ciudad, dependiendo del valor que tenga la variable ***diferencia***. Observe que a esta función se le pasa como parámetro un entero que indica la diferencia horaria entre Managua y la ciudad de la cual se quiere visualizar su hora. Edite el código de esta función; en este código se debe consultar el valor de la variable diferencia y dependiendo del valor que tenga se mandará a imprimir en la caja de texto correspondiente la hora de esa ciudad. Para calcular la hora de la ciudad debe sumar a la hora actual del sistema el número de horas de diferencia usando un objeto de la clase ***CTimeSpan*** (→ ver ayuda de Visual Studio MSDN)

8. A continuación debemos añadir el método donde se mandará a llamar la función ***MostrarHoraCiudad(int diferencia).*** Para ello, invoque a ***ClassWizard*** presionando las teclas ***Ctrl+W***; en ***Object ID's*** seleccione ***CHorasMundialesView*** y añada la función miembro ***OnTimer()*** para que maneje el mensaje ***WM_TIMER.*** Esta función es la que se encargará de llamar a la función ***MostrarHoraCiudad(int diferencia)***. Luego edite el código de esta función; a manera de ejemplo se muestra a continuación cómo se debe llamar a la función para que muestre la hora de Madrid:

if(IsWindow(this->m_hWnd) && nIDEvent==m_nTimerMadrid)
this->MostrarHoraCiudad(7);

Observe que se le pasa como parámetro el valor 7 porque se trata de la hora de Madrid. Repita esto mismo para los otros casos

9. Finalmente, invoque a ***ClassWizard,*** y añada la función miembro **OnDestroy()** a la clase ***CHorasMundialesView*** para que maneje el mensaje ***WM_DESTROY.*** Edite el código y añada al final las siguientes líneas:

```
if (this->m_nTimerManagua)
    this->KillTimer(ID_TIMER_MANAGUA);
```

   Repita este mismo paso para los otros temporizadores

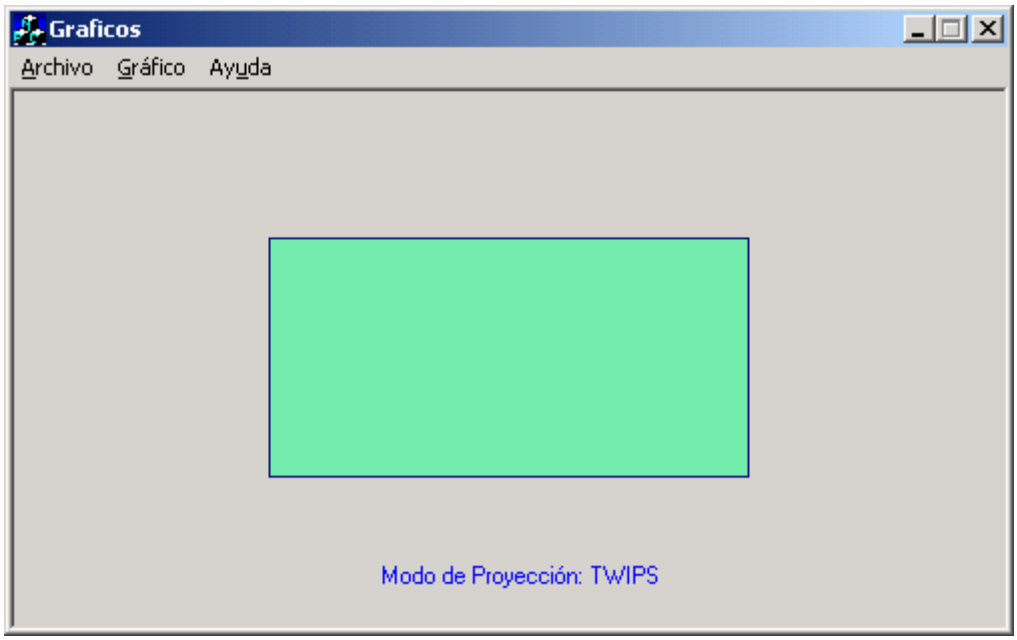
10. Compile y ejecute para ver los resultados!!!

CREACIÓN DE GRÁFICOS EN VISUAL C++

En este ejercicio se pretende que el alumno aprenda a utilizar las clases para dibujar elipses y rectángulos, cambiar el tipo de fuente asociado a un control, manejo de temporizadores y modos de proyección.

Introducción

En esta práctica se desea crear una aplicación que tenga el aspecto que se muestra en la figura siguiente:



Esta figura cambiará de forma aleatoria tanto el color del fondo como el del borde cada segundo. Al dar clic en el menú ***Gráfico*** se desplegará un submenú que contiene las opciones: ***Tipo de Figura, Cambiar Fuente y Cambiar modo de proyección***.

Al dar clic en ***Tipo de Figura*** se visualizará un submenú con las opciones: ***Elipse, Cuadrado y Rectángulo redondeado.*** Al dar clic en cualquiera de estas opciones, se visualizará la figura correspondiente en la vista. Al dar clic en la opción ***Cambiar Fuente*** se abrirá un diálogo de tipo fuente que permitirá cambiar el texto de la etiqueta. Al dar clic en ***Cambiar modo de proyección*** se visualizará un submenú con las opciones: ***Text, Twips, Lometric, Himetric, Loenglish, Hienglish, Isotropic y Anisotropic.*** Estas opciones me permitirán cambiar al correspondiente modo de proyección.

Iniciando la aplicación

1. Inicie Microsoft Visual C++.

2. En el menú principal, clic en File -> **New**

3. Clic en **MFC AppWizard (exe)**

4. En la caja de texto del nombre del proyecto ponga **"Graficos"**

5. Elija la carpeta donde desee guardar los archivos que forman parte del proyecto.

6. En el **"paso 1 de 6"** seleccione la opción "Single Document", esto significa que se trabajará con una aplicación SDI (tendrá solamente una vista).

7. En el **"paso 2 de 6"** se le pregunta si su aplicación tendrá soporte a bases de datos. Por lo tanto, seleccionaremos la opción "None".

8. En el **"paso 3 de 6"** escoja la opción por defecto.

9. En el **"paso 4 de 6"** seleccione solamente la opción 3D Controls.

10. En el **"paso 5 de 6"** escoja las opciones por defecto

11. En el **"paso 6 de 6"** no olvide escoger como clase base a la clase **CFormView.**

Diseñando la aplicación

En este caso solamente se dibujará una etiqueta con el texto "Modo de proyección: "

Programando la aplicación

1. Como primer paso, edite la clase CGraficosView e incluya las declaraciones de las siguientes variables privadas de esta clase:

   CFont m_Fuente;
   COLORREF m_rgbColorFuente;
   int m_modoProyeccion;
   CString m_tipoFigura;
   bool m_bRepintarTodo;

2. Con ayuda de ClassWizard, asocie la etiqueta con la variable miembro de tipo String *IDC_ETIQUETA*

3. En el editor de menús, construya el menú tal como se muestra a continuación:

Archivo
    Salir
Grafico
    Tipo Figura

Elipse
Cuadrado
Rectángulo redondeado
Cambiar fuente
Cambiar modo de proyección
Text
Twips
Lometric
Himetric
Loenglish
Hienglish
Isotropic
Anisotropic
Ayuda
Acerca de Gráficos

4. A continuación edite la clase CGraficosView e incluya la siguiente función miembro pública (esta función es la que se encargará de dibujar):

void Dibujar();

5. Ahora escriba el código de esta función. Sólo deberá completar el código que falte:

```
void CGraficosView::Dibujar()
{
   // Dibujando...

   CClientDC dc(this);

   this->RedrawWindow();

   // Usar la función srand

   // Obtener el tamaño de la ventana: usar un objeto de
   tipo //CRect Ver págs. 505-506 del libro

   // Establecer el origen de coordenadas en el centro (Ver
   // Pág. 518)

   // Establecer las unidades del sistema de coordenadas
   // (use la  variable miembro m_modoProyeccion)

   int x1,y1,x2,y2;  // coordenadas para pintar la figura

   // Convertir unidades de dispositivo a unidades lógicas
   // usando las coordenadas 120,60,120,60. Use las
   // variables x1,y1,x2,y2. Ver pág. 525 del libro

   // Declare un objeto de la clase CPen y un objeto de la
   // clase CBrush. Los colores se deben generar de forma
```

```
    // aleatoria usando la función rand()

    // Seleccionar los objetos anteriores para dibujar

    // Dibujar la figura correspondiente. Para saber qué tipo
    // de figura se dibujará debemos consultar el valor de la
    // variable m_tipoFigura. Esta variable será actualizada
    // desde el menú. Para dibujar la figura se usarán las
    // coordenadas x1,y1,x2,y2

    // Eliminar el lápiz y la brocha

    this->m_bRepintarTodo = false;
    this->InvalidateRect(NULL, false);
}
```

6. En la función ***OnInitialUpdate()*** escriba las siguientes inicializaciones de variables:

```
this->m_bRepintarTodo = true;
this->m_rgbColorFuente = RGB(0,0,255);          //AZUL
this->m_modoProyeccion = MM_TEXT;
this->m_tipoFigura = "Elipse";
this->SetTimer(1, 1000, NULL);
this->GetParentFrame()->CenterWindow();
this->m_textoEtiqueta = "Programación con Gráficos en
Visual C++";
this->UpdateData(false);
```

7. Invoque a ClassWizard para añadir la función OnTimer() a CGraficosView para manejar el mensaje WM_TIMER. Edite el código de tal forma que quede así:

```
this->Dibujar();
this->m_bRepintarTodo = false;
this->InvalidateRect(NULL, false);
CFormView::OnTimer(nIDEvent);
```

8. Invoque a ClassWizard para añadir la función OnDestroy() a CGraficosView para manejar el mensaje WM_DESTROY. Edite el código de tal forma que quede así:

```
CFormView::OnDestroy();
this->KillTimer(1);
```

9. Invoque a ClassWizard para añadir la función OnCtlColor() a CGraficosView para manejar el mensaje WM_CTLCOLOR. Edite el código de tal forma que quede así:

```
HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

if (pWnd->GetDlgCtrlID() == IDC_ETIQUETA)
```

```
        pDC->SetTextColor(this->m_rgbColorFuente);

return hbr;
```

10. En el editor de menús, haga clic en la opción ***Cambiar Fuente***. A continuación invoque a ClassWizard para añadir la función ***OnGrficoCambiarfuente*** que será llamada cuando se dé clic sobre esta opción del menú. Edite el código de esta función:

```
void CGraficosView::OnGrficoCambiarfuente()
{
   //Cambiando el tipo de letra de la etiqueta...

   CFontDialog DlgFuente;

   LOGFONT lf;

  DlgFuente.m_cf.lpLogFont = &lf;

  if (DlgFuente.DoModal() == IDOK)
  {
    this->m_Fuente.DeleteObject();
    this->m_Fuente.CreateFontIndirect(&lf);
    this->m_rgbColorFuente = DlgFuente.GetColor();
    this->GetDlgItem(IDC_ETIQUETA)->SetFont(&m_Fuente);
  }
}
```

11. En el editor de menús, haga clic en la opción ***Elipse***. A continuación invoque a ClassWizard para añadir la función ***OnGrficoTipodefiguraElipse*** que será llamada cuando se dé clic sobre esta opción del menú. Edite el código de esta función:

```
void CGraficosView::OnGrficoTipodefiguraElipse()
{
// Asignar a la variable m_tipoFigura la cadena "Elipse";
}
```

12. Repita el paso anterior para las opciones ***Cuadrado*** y ***Rectángulo redondeado***

13. En el editor de menús, haga clic en la opción ***Text***. A continuación invoque a ClassWizard para añadir la función ***OnGrficoCambiarmododeproyeccinText*** que será llamada cuando se dé clic sobre esta opción del menú. Edite el código de esta función:

```
void CGraficosView::OnGrficoCambiarmododeproyeccinText()
{

   // Asignar a la variable m_modoProyeccion el valor
   MM_TEXT
```

```
    // Asignar a la variable m_textoEtiqueta la cadena "Modo
    de // proyección: TEXT";

    // Llamar a UpdateData para actualizar la etiqueta
}
```

14. Repita el paso anterior para las opciones ***Twips, Lometric, Himetric, Loenglish, Hienglish, Isotropic y Anisotropic*** haciendo los cambios oportunos.

15. Edite la función ***OnDraw()*** y añada el código correspondiente a la función ***Dibujar().***