Índice General

1 Introducción **3**

2 Memoria **5**

2.1 Especificaciones . . . 5
2.2 Requisitos previos . . . 7
2.2.1 Software . . . 7
2.2.2 Hardware . . . 9
2.3 Guía del usuario . . . 10
2.3.1 Análisis gráfico . . . 10
2.3.2 Filtrado . . . 13
2.3.3 Grabar a disco . . . 14
2.3.4 Opciones de captura o muestreo . . . 15
2.3.5 Selección de dispositivo . . . 15
2.3.6 Informacion del hardware . . . 16
2.4 Guía de programación . . . 18
2.4.1 Clase CDialog_Muestreo . . . 19
2.4.2 Clase CDialog_Graficos . . . 20
2.4.3 Clase CDialog_Filtrado . . . 20
2.4.4 Clase CDialog_Personal . . . 22
2.4.5 Clase CDialog_CapsCaptura . . . 22
2.4.6 Clase CDialog_CapsReproducción . . . 23
2.4.7 Clase CMainFrame y CEcualizadorDlg . . . 23
2.4.8 Clase CDialog_Error . . . 28
2.4.9 Clase CDirectDraw . . . 28
2.4.10 Clase CDirectS . . . 31
2.4.11 Clase CFFT . . . 35
2.4.12 Clase CFiltradoView . . . 36
2.4.13 Clase CAnalisisView . . . 38
2.4.14 Clase CFiltro . . . 39
2.4.15 Clase Genrerar_RR . . . 40
2.4.16 Estructura Opc_Muestreo . . . 40
2.4.17 Opc_Graficos . . . 40

3 Código del programa **41**

3.1 Análisis . . . 41
3.1.1 AnalisisView . . . 41
3.1.2 Dialog_Error . . . 47

3.1.3 DirectDraw . . . 50
3.1.4 DirectS . . . 81
3.1.5 Estructuras.h . . . 104
3.1.6 Fft . . . 105
3.1.7 Generar_RR . . . 110
3.1.8 MainFrm . . . 112
3.2 Ecualizador . . . 123
3.2.1 Dialog_CapsCaptura . . . 123
3.2.2 Dialog_CapsReproduccion . . . 126
3.2.3 Dialog_Dispositivo . . . 129
3.2.4 Dialog_Error . . . 133
3.2.5 Dialog_Filtrado . . . 136
3.2.6 Dialog_Graficos . . . 148
3.2.7 Dialog_Muestreo . . . 155
3.2.8 Dialog_Personal . . . 160
3.2.9 DirectS . . . 163
3.2.10 EcualizadorDlg . . . 184
3.2.11 Estructuras.h . . . 203
3.3 Filtrado . . . 203
3.3.1 Dialog_Error . . . 203
3.3.2 Dialog_Filtrado . . . 206
3.3.3 DirectS . . . 210
3.3.4 Fft . . . 229
3.3.5 FiltradoView . . . 235
3.3.6 Filtro . . . 256
3.3.7 MainFrm . . . 263

4 Presupuesto **279**

Capítulo 1

Introducción

El proyecto pretende la implementación de una aplicación que corra bajo Windows, ya que es el sistema operativo más extendido, y que emule las dos herramientas principales que se utilizan en la ecualización de salas. En primer lugar un analizador de frecuencias, para el espectro audible por el hombre, haciendo caso de la norma Hi-Fi, está comprendido entre los 20Hz y los 20.000 Hz, que se complementa con un generador de ruido rosa. Además se emula también un ecualizador para poder simular el resultado final.

Por lo tanto el proyecto quiere sustituir a los analizadores tradicionales, construidos mediante filtros y con las columnas de led's fijas, que normalmente traían ya acoplados el micrófono, que realiza la transformación de energía acústica a eléctrica, superponiendo, como es natural, su propia respuesta en frecuencia, así como su ancho de banda. También es conveniente ayudarse de un generador de ruido rosa que normalmente es otro dispositivo independiente, para realizar el análisis completo. El problema de estos aparatos, sobre todo el analizador de frecuencias es su precio, puesto que la realización de un buen analizador de frecuencias requiere circutería cara y si queremos que nos proporcione algún tipo de flexibilidad o configuración, los precios se pueden subir hasta las 500.000 ptas. Por el contrario este proyecto permite realizar este proceso del análisis con un ordenador normal, en cuanto a precio se refiere, permite una total flexibilidad en cuanto a la visualización del análisis y la seleción del tipo de micrófono que se quiera, según el tipo de sonido que se vaya a analizar, por lo tanto se reduce el precio de este servicio a cotas razonables. En cuanto a la otra funcionalidad, es decir, la emulación de un ecualizador, como ya se sabe son todos lo suficientemente caros como para tener un stock y poder intercamabiarlos hasta que se encuentre el sonido deseado, con la aplicación se puede cambiar la configuración tantas veces como se quiera pudiéndose emular ecualizadores reales que una vez simulados y escuchados se procedería a comprar con un riesgo mínimo, puesto que ya se tiene una idea de como van a sonar en el conjunto del equipo.

Por lo tanto con la aplicación aquí descrita se podrá realizar un análisis y llegar a una posible solución para la sonorización de una sala cualquiera sin más que desplazar a la localización en particular un portátil y un micrófono patrón. Así se simplifica la tarea de la optimización de la acústica de una sala, con el primer paso del análisis se realiza una medida de las componentes espectrales de la señal audible, con lo que se proporciona un planteamiento del problema. Una vez que se conoce éste, se puede buscar una solución, que con el Modo Filtrado se puede simular y tantear hasta que

el usuario quede totalmente conforme con la solución o el sonido resultante, para luego proceder a colocar los equipos que quedarán fijos realizando las tareas definidas a través de la simulación y el análisis previo.

Capítulo 2

Memoria

2.1 Especificaciones

Para poder ejcutar con éxito la aplicación y que funcione correctamente se procederá a describir un montaje estándar y todo lo necesario para la conexión de los distintos equipos para que puedan interactuar correctamente. En primer lugar se describe la máquina que realizará los cálculos y donde correrá la aplicación.

Se aconseja un PentiumIII o superior con 64 Mbytes de RAM o más, en particular se ha probado en un portátil PentiumIII a 450MHz con resultados muy satisfactorios, para equipos de sobremesa esta configuración es igualmente válida puesto que las tarjetas de sonido de éstos son mejores que las de los portátiles, en estos últimos parte de sus funcionalidades han de ser emulados por el microprocesador con la merma de procesamiento correspondiente, se comenta aquí puesto que se han mencionado los portátiles, que actualmente no incluyen entrada de línea en las tarjetas de sonido que traen incorporadas, por lo tanto para realizar el filtrado será necesario conectarle una tarjeta de sonido con ella, o PCMCIA o bien USB externa . Por supuesto con tarjeta de sonido y que todo el hardware sea compatible con DirectX y Windows, siendo necesario que estén instalados ambos, para DirectX, la versión 6.0 o superior y para el sistema operativo Windows 98 o W98SE, o, NT 4.0 o bien Windows 2000. Se han detectado problemas con Windows-ME, así que se aconseja no utilizar la aplicación bajo este sistema operativo, si puede funcionar, pero es posible que no lo haga correctamente. Como se dice al comienzo es necesario disponer de una tarjeta de sonido con capacidad de manipulación de audio digital con calidad CD, es decir, que soporte un muestreo a frecuencia constante de al menos 44.100 Hz, no es necesario que la tarjeta dispònga de dos relojes independientes para captura y reproducción, que cada muestra se pueda cuantificar con 16 bits y que soporte modo stereo, esto es, 2 canales de audio a la vez. Ejemplo de tarjetas de sonido, pueden valer todas las de creative por encima de la 16bits, y por supuesto todas las mejores que se puedan en contrar en el mercado, no hay que olvidar que cuanta más calidad haya en la recogida de datos mejor será el resultado, puesto que los errores serán menores.

Con todo instalado y funcionando en el ordenador se está en condiciones de lanzar al aplicación, haciendo doble click en *Ecualizador.exe*. Existen dos modos de funcionamiento básicos, por lo que se harán todas las descripciones distinguiendo ambas, estos son Modo Análisis y Modo Filtrado, por lo que se comenzará por el primero de ellos.

- Modo Análsis: como su nombre indica se relizará un análisis espectral de la señal que se determine como fuente. Se explicará esto detenidamente, para ello se comentará como interactuará con el sistema que se desea analizar. Supongamos un equipo de audio de los que suele existir en un bar, es decir, existirán unos reproductores de cualquier tipo de soporte, CD, cintas de audio, vídeos, vinilos, etc..., que están todos unidos a una mesa de mezclas y ésta, a su vez atacará a la etapa de potencia, que moverá los conos de los altavoces que estarán repartidos por la sala. Se conectará la salida de la tarjeta de sonido a una entrada de la mesa de mezclas, y la señal que se medirá se introducirá al ordenador mediante un micrófono patrón, valdrá cualquier otro pero se debe conocer su respuesta en frecuencia, para evitar errores graves y sorpresas desagrdables en el resultado final. Por lo tanto se pincha el micrófono en la entrada correspondiente de la tarjeta de sonido, dándole paso, utlizando el mezclador del ordenador para que se pueda grabar o capturar la señal que haya en ella. Con este montaje solo queda seleccionar a través de los diálogos de la aplicación si se desea analizar la señal resultante del ruido rosa atravesando todo el equipo de sonido, puesto que pasará por la mesa, la etapa y los altavoces, imprimiendo cada uno su respuesta en frecuencia a la señal, o bien analizar otro

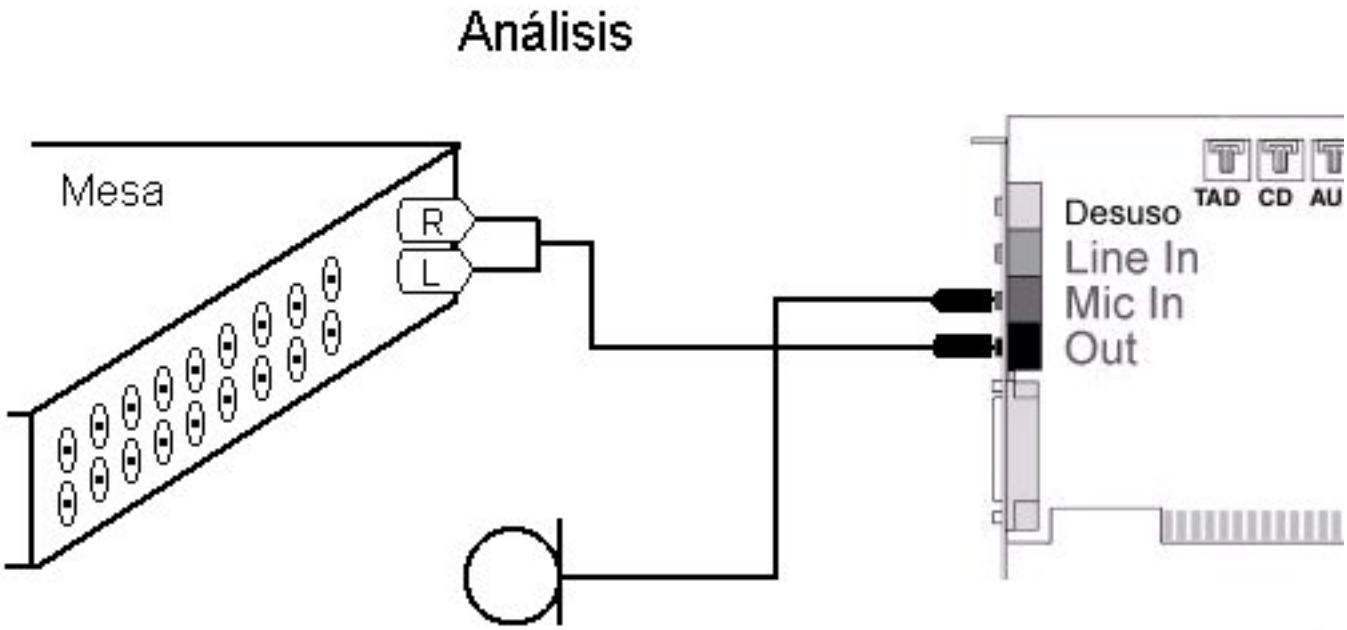

Figura 2.1: Esquema típico de conexión

- Modo Filtrado: en este caso se conectará el ordenador entre la mesa y la etapa, o bien, si la mesa está preparada para la conexsión de procesador o ecualizador lo pincharemos allí, así se obliga a la señal a pasar a través del ordenador, en este caso es obligatorio usar una fuente externa de audio, aunque se puede usar el CD-ROM del ordenador. Ahora se usará la salida de la tarjeta de sonido para atacar la entrada de la etapa, o bien, a los RCA's marcados como salida del procesador, mientras que la entrada de línea de la tarjeta de sonido recibirá o la salida de la mesa o la entrada del procesador, fijándose en la serigrafía de la mesa.

Por lo tanto será necesario disponer de al menos dos latiguillos Stereo de minijack

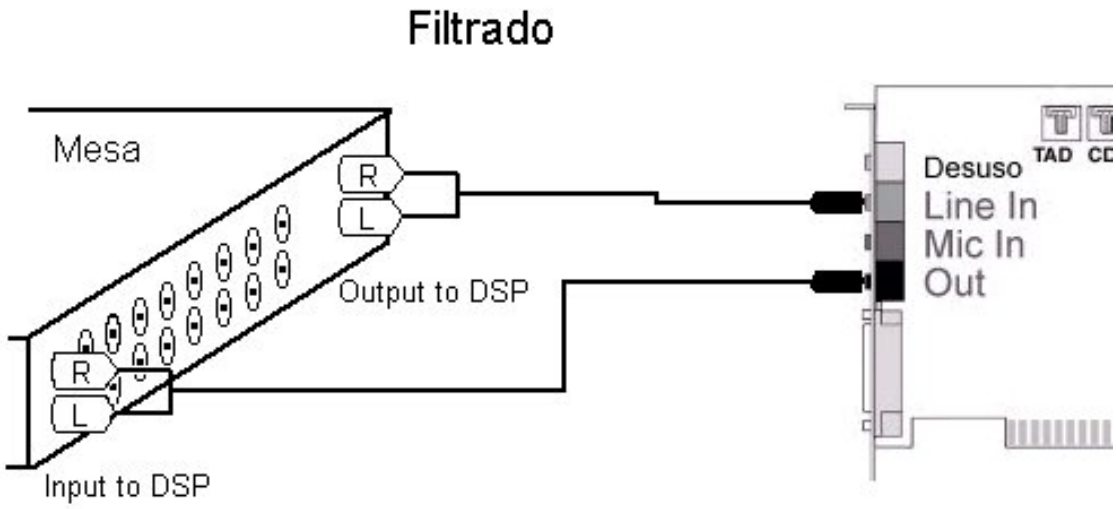

Figura 2.2: Esquema típico de conexión

(interfaces de la tarjeta de sonido) a RCA's interfaces del resto de equipos de audio (suele ser lo normal). También es posible que sea necesario un adaptador de jack a mini-jack, para poder conectar el micrófono a la tarjeta de sonido.

2.2 Requisitos previos

En este apartado se pretende dar cuenta del material necesario para el desarrollo de la aplicación, se explicará por separado hardware y software. Se mostrarán los elementos y herramientas que se han empleado en la fase de diseño y posterior implementación.

2.2.1 Software

La aplicación está construida con Visual C++ con lo que ha sido necesario instalar en el ordenador un compilador para este lenguaje, en particular, se ha utilizado el Microsoft Visual C++ 5.0, se ha tomado esta elección por compatibilidad y evitar problemas entre el sistema operativo y el compilador, no hay que olviadar toda la ayuda on-line que posee el entorno de programación y que es de gran utilidad [1]. Quizás sea necesario comentar aquí que se han compilado todas las partes de la aplicación con optimización del código para Pentium, con lo que todos los microprocesadores inferiores a esta arquitectura podrán dar errores si se pretende ejecutar la aplicación en máquinas basadas en ellos. Acompañan en la instalación del compilador algunas utilidades más, como son el SPY++ que es muy útil para saber que procesos o threads están actualmente en ejecución en el sistema, permitiendo conocer tambien las ventanas marco padre, puesto que la aplicación hace uso de threads.

Ha sido necesario también la SDK de DirectX 6.0 con todo lo que contiene, es decir, desde el ejecutable que sirve la ayuda convenientemente ordenada [2], a los ejemplos que

se proporcionan para que se compruebe la dinámica de funcionamiento de los distintos componentes que forman DirectX, es posible que sea necesario en algún momento trabajar con las versiones Debug de las DLL, solo hay que seleccionarlo en la instalación, esto facilita ciertas tareas en la depuración de la aplicación puesto que proporciona un mejor interfaz frente a los errores que se puedan producir. También señalar la posibilidad de conectar en red otro ordenador para poder observar la ventana de depuración mientras se hace lo propio con zonas del programa donde interviene DirectDraw, controlando la pantalla completa, agiliza en gran manera el ciclo de prueba y recompilación. También se ha hecho uso de la SDK de DirectX 5.0, siendo de gran utilidad los tutoriales y sobre todo los ejemplos de programas, que facilitan en entendimiento y la interoperatibilidad de todas las funciones.

El corazón de la aplicación lo forma el cálculo de la transformada de Fourier en ambos sentidos, es decir, del tiempo al dominio de la frecuencia y el camino inverso. El algorítmo que se emplea para estos cálculos se ha obtenido de la página web *www.fftw.org*, esta organización proporciona, según ellos mismos, la transformada de Fourier más rápida del oeste. Todos los ficheros empleados están escritos en lenguaje C, por lo que se han podido compilar usando el mismo compilador que el resto de la aplicación. En la misma página web se puede bajar toda la docuementación necesaria para la compilación y ejecución en la aplicación que el usuario deseé beneficiarse de la transformada de Fourier [3]. Se compilan todos los archivos bajados de el site de fftw, como una librería que posteriormente se utilizará en la aplicación. Se utilizan datos tipo double, para proporcionar más exactitud en los cálculos. Se proporcionan programas para el test de la librería creada y para medir el tiempo que tarda en el procesamiento.

Se ha empleado un programa de tratamiento de imagenes adicional, en particular se ha usado el Paint Shop Pro 5.0 pero cualquiera con prestaciones similares puede valer, no hay que olvidar el aspecto que ofrecerá la aplicación. Esto involucra el diseño y la realización de todo tipo de dibujos para completar los cuadros de diálogos. Se ha creado con él, el recurso que porporcionará las piezas necesarias para la composisción de las imágenes en el caso del Modo Análisis puesto que al ser un archivo de imagen .bmp ofrece gran versatilidad en su tratamiento, también el diseño de los iconos que identificarán los distintos programas generados, que aunque se pueden crear con el entorno integrado del compilador de Microsoft, se realizan más fácilmente con otros programas como el que se ha utilizado, Axialis AX-Icons 4.5.

Se ha empleado para el diseño y desarrollo de la ayuda el Microsoft Help WorkShop, que aunque venía una versión anterior con la instalación de Microsoft Visual C++, se ha empleado otra más nueva, la 4.03 que además de traer facilidades añadidas es totalmente compatible con el entrono integrado del compilador, puesto que es capaz de compilarlo todo a la vez, cuando se le indica que se compilen los archivos del programa. Señalar aquí que para generar los documentos base para la ayuda se ha empleado el Microsoft Word, de las office 97, necesario para crear el archivo en formato .rtf, la base sobre la que se crea la ayuda. Se encontraron problemas de compatibilidad con el formato .rtf que generaban las distintas versiones de Word, por eso se hubo de pasar a la version 4.03 de Help WorkShop que fuera capaz de procesar las marcas del Word 97.

La instación de la aplicación con todos los archivos necesarios, se ha llevado a cabo

con la ayuda del Microsoft InstallShiled 5.0 free Edition, que permite la creación de programas de instalación, gestionando la copia de los archivos implicados, la actualización del registro de Windows y la creación de los accesos directos en el menú de Inicio y la carpeta que el usuario deseé dentro de Programas. El hecho de ser free Edition hace que ciertas funcionalidades avanzadas no estén disponibles por eso no se ha realizado un proceso de instalación complejo.

Para la escritura de la docuemtación del proyecto, como puede observar el que esté leyendo esto, se ha empleado LaTeX, con lo cual se ha empleado todo el software necesario para utilizarlo TeXv3.3.1.

2.2.2 Hardware

En primer lugar es necesario un ordenador, pero no tan potente como se dice en los requerimientos, esto es así puesto que se ha usado un Pentium MMX 233 con 64MB para el desarrollo, se pueden ir probando las distintas funciones de la aplicación con calidades de captura menores, con lo que los resultados obtenidos serán extrapolables para configuraciones de más calidad y por tanto más exigentes en cuanto a capacidad de procesamiento, no hay que olvidar que se trata de procesamiento en tiempo real. También es conveniente aclarar aquí que la mayor exigencia de potencia de cálculo se hace notoria en el Modo Filtrado, puesto que en el Modo Análisis con la máquina antes descita es suficiente para todo tipo de configraciones, eso si, le ayuda la tarjeta gráfica con algo de aceleración puesto que buena parte del peso del procesamiento recae en la puesta en pantalla.

Otro elemento imprescincible es tener una tarjeta de sonido, capaz de procesar audio digital, al menos que sea posible capturar y reproducir, es decir, que permita modo Full-Dúplex, audio a calidad CD, es decir, frecuencia de muestreo 44.100 Hz, a 16 bits y Stereo. Actualmente todas las tarjetas del mercado soportan esto y seguramente más. En la fase de desarrollo se a contado con la Sb128 Pci de Creative, aún así se recomienda cualquiera de Creative superior a la 16 bits, puesto que se han encontrado incompatibilidades con DirectX en cuanto al interface de captura, y por supuesto todo tipo de tarjetas compatibles o mejores que éstas, pero sin perder compatibilidad con DirectX, no hay que olvidar que DirectX proporciona el soporte para el acceso al hardware a la aplicación.

De acuerdo con lo que se acaba de explicar, es igualmente válido para la tarjeta gráfica, aunque a día de hoy casi todas las tarjetas gráficas poseen esta compatibilidad por esta razón no se hace tanto incapié en ello.

Por último, aunque no por eso menos importante, sobre todo en el Modo Análisis, el micrófono que se usará como transductor, se recomientda un micrófono patrón que está preparado para análisis espectrales de todo el rango audible, aunque si se usa otro captador será totalmente válido, salvo que hay que tener presente en todo momento su respuesta en frecuencia para no llegar a resultados o conclusiones erróneas, puesto que el micrófono modelará la señal con su respuesta en frecuencia, lo normal en este caso son dispositivos que no abarquen todo el espectro, por lo que luego podrán a parecer cosas que en principio no aparecen en la fase de análisis, conduciendo inevitablemente a malas

conclusiones y posteriormente a erróneas planarizaciones.

2.3 Guía del usuario

En esta sección se procede a describir la interfaz de la aplicación con el usuario, por lo tanto se dará una descripción de las funcionalidades que proporciona y como llegar hasta ellas.

2.3.1 Análisis gráfico

Se pretende emular los analizadores de frecuencias gráficos que muestran los resultados en columnas de led's que se activan hasta una altura proporcional a la amplitud de la componente en frecuencia o banda de frecuencias que representa dicha columna. Como ejemplo de esta funcionalidad cualquiera de las minicadenas que se venden actualmente, poseen un display donde se muestra algo parecido a lo que se pretende obtener. A diferencia de éstas o de cualquier implemetación de este tipo de dispositivos mediante hardware, la aplicación puede configurarse con un grado de libertad bastante amplio en cuanto a la puesta de los datos por pantalla. Esto es posible puesto que se están emulando las columnas de Led's antes descritas, con lo que se pueden cambiar y colocar como mejor convenga, esta es una de las principales ventajas que proporciona el disponer de un elemento de visulización versátil, como es el que forman la VGA y el monitor de cualquier ordenador o PC. Estas opciones se encuentran en el dialogo de opciones gráficas que se accede a él, bien pulsando su botón correspondiente en la aplicación *Ecualizador.exe*, o al iniciar la secuencia de lanzamiento del modo Análisis por primera vez.
Antes de proceder a la descripción del diálogo de opciones gráficas, que es lo más importante para este punto, como notará el usuario que haya lanzado directamente el Análisis antes de llegar a este diálogo hay que tomar decisiones sobre otros dos, estos son el diálogo de Selección de dispositivo y Opciones de captura, que se describirán más adelante dado su carácter general. Una vez que se tiene el diálogo en pantalla se puede configurar prácticamente todo, es decir:

- Eje de ordenadas. Este eje alberga la amplitud de las componenetes en frecuencia, por lo que se pueden configurar los extremos **cota superior**, **cota inferior** y el **N^o de escalones**. Con ellos se especifica el rango que se mostrará por pantalla y el espaciado del mismo. Los límites se han estalecido entre 0 y -99 dB, y como máxima resolución 1 dB. Esto no presenta nigún incoveniente, puesto que el hardware encargado de la captura, como máximo usa una cuantificación de 16 bits, lo que implica un máximo de 65536 escalones, que en rangos dinámicos son alrededor de 96.32 dB, por lo tanto no hay problema. Aún así por debajo de -70 dB, el ruido de cuantificación en los codificadores se puede hacer muy molesto y menguar la calidad de la captura. El mínimo de escalones permitido es 3, tan buena elección como cualquier otra.

- Eje de abcisas. En este caso se representan las frecuencias, con un funcionamiento completamente similar al anterior, permitiéndose definir los límites con **frecuencia inferior** y con **frecuencia superior**, mientras que el espaciado se gestiona a partir del **N^o de cortes**. En este caso se realiza un reparto a lo largo del eje de forma logarítmica, por la naturaleza de la señal capturada, pudiéndose alterar

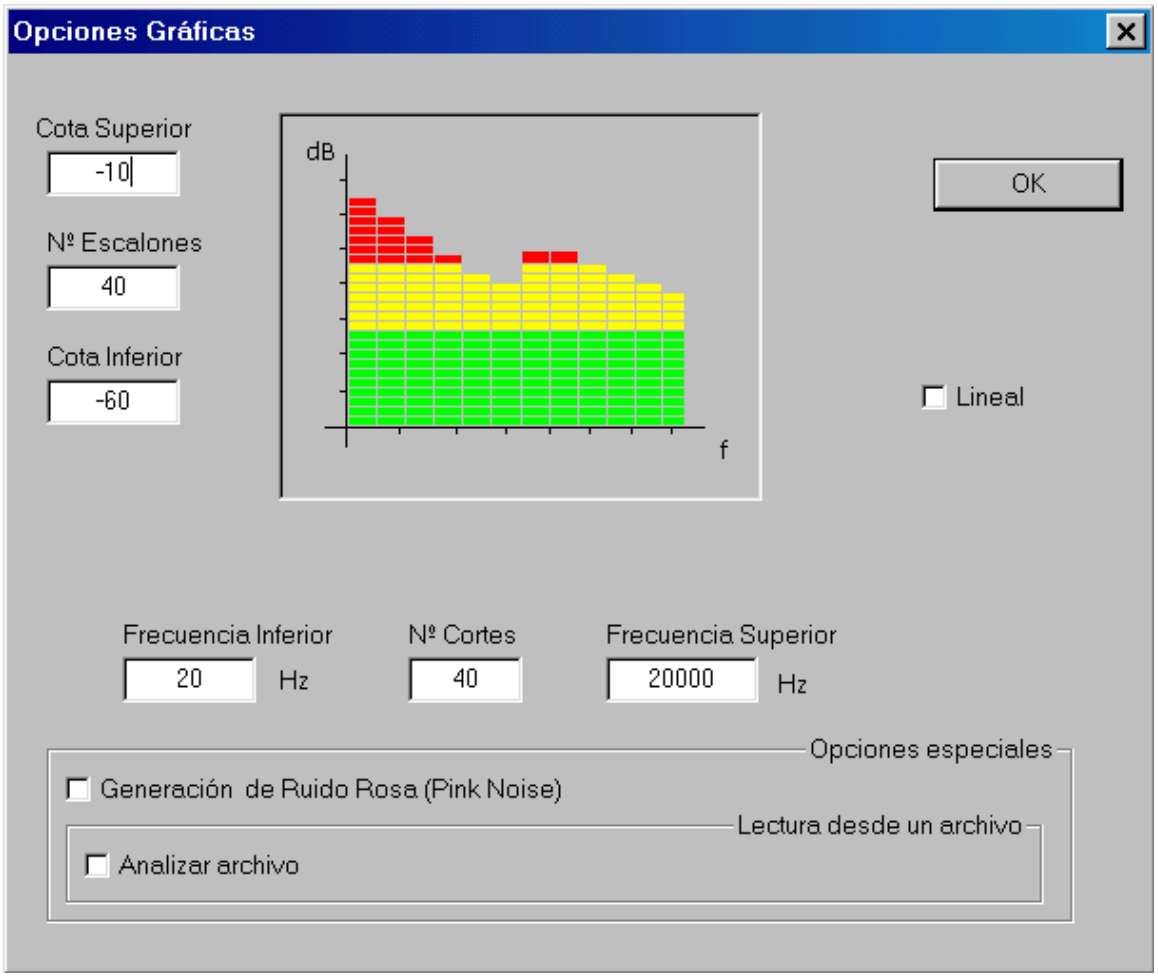

Figura 2.3: Diálogo de opciones gráficas

esta característica activando la casilla de **lineal** que existe a tal efecto. Los límites impuestos son los de la norma Hi-Fi, es decir, solo frecuencias comprendidas entre 20 y 20.000 Hz, pudiendo pedir todo el espectro entero o partes de él, también hay que tener encuenta el teorema de Nyquist que relaciona el ancho de banda de la señal que se está capturando o muestreando, con la frecuencia de muestreo que se define en el diálogo de captura. No hay problema en caso de errores con este respecto, la aplicación se dará cuenta de ello y lo corregirá, predominando la frecuencia de muestreo sobre cualquier otro parámetro definido por el usuario, por lo tanto se ajustarán los límites a ésta automáticamente. Otra de las limitaciones es el espaciado mínimo que está definido por la frecuencia de muestreo y el número de puntos que utiliza la FFT, pero en caso de conflicto, al igual que antes, la aplicación es capaz de reconocerlo y reajustar los parámetros al espaciado mínimo. Por último el mínimo de cortes que se permite representar por pantalla son 3, una elección como otra cualquiera.

Hasta aquí son las funciones básicas para lanzar el modo análisis correctamente, se pasará pues a describir las opciones avanzadas, siendo lo hasta ahora descrito válido también para estos, puesto que las capacidades que se van a explicar son aditivas al

funcionamiento básico.

- Generación de Ruido Rosa. En primer lugar se explica como se gestiona la prioridad en caso de activarse las dos al mismo tiempo, predominará la activación del ruido rosa, por lo que a todos los efectos para la aplicación será como si solo se hubiera activado ésta.
  El resultado es la obtención de dicho ruido por la salida de la tarjeta, haciendo por lo tanto un uso en Full-Duplex de la tarjeta, puesto que sigue capturando con ella para mostrar el análisis por pantalla. Esta opción esta diseñada para una primera estimación de la acústica de una sala. Por lo tanto es ideal su uso junto con un micrófono patrón, que sea éste el que haga la captación del sonido, a la vez que obligamos al Ruido Rosa generado, atacar al equipo completo. Se describirá un montaje estándar para facilitar la compresión. Se conecta la salida de la tarjeta de sonido a una entrada de la mesa que sea de línea, es decir, la impedancia que muestra al exterior sea de esta característica, para ello se usará uno de los latiguilllos que se mencionó de mini-jack a RCA, con lo que si damos paso a la señal con el potenciómetro correspondiente en la mesa se conducirá la señal a la etapa y de esta llegará a los altavoces, todo el sistema completo. Como tenemos un micrófono en el medio del campo acústico que crearán los altavoces, tendremos la señal resultante en forma eléctrica para introducirla de nuevo al ordenador a través de la entrada que a tal efecto poseen todas las tarjetas de sonido. En caso que fuera necesario, es posible que haya que adaptar, como se comentó anteriormente, los conectores del micrófono que suele ser jack a mini-jack, que son los conectores que suelen las tarjetas de sonido. Con este montaje se permite comprobar como deforma la señal original del Ruido Rosa el sistema entero de audio.

- Analizar archivo, esta funcionalidad es la complementaria a la de grabación de archivos wav de esta misma aplicación aunque, reconoce archivos wav capturados en formato PCM sin compresión desde cualquier software. Esto permite la captura de la acústica de una sala y la posibilidad del su análisis posterior. Simplemente se toma como funte para el análisis, en vez de una señal que esta presente en cualquiera de las entradas de la tarjeta de sonido, un fichero previamente grabado.

Antes de terminar se comentará aquí la relación existente entre la frecuencia de muestreo y la FFT o DFT. Si se sobremuestrea con mucho margen, el resultado es que entre la frecuencia más alta de la señal que se pretende digitalizar y la frecuencia de Nyquist o freucuencia más alta que es posible muestrear correctamente quedará mucho hueco en el que no habrá componentes espectrales con información válida. Como se ha mencionado solo se obtienen en el dominio de la frecuencia la mitad de los puntos que se han proporcionado en el del tiempo, pero si además gran parte de estos puntos corresponden a la zona que se encuentra entre la frecuencia más alta de la señal a muestrear y la frecuencia de Nyquist, se encontraran muchas componentes a 0 o desperdiciadas, por eso se ha colocado como techo 44100 Hz, se sobremuestrea pero dejando solo 2500 Hz de margen. Las tarjetas actuales permiten, casi todas, frecuencias de muestreo de hasta 100 kHz, pero por la razón antes descrita se ha optado por una cota más conservadora. Este dato se puede ver en el diálogo de información de las características de reproducción, descrito más de talladamente en siguientes secciones.

2.3.2 Filtrado

En esta sección se pretende emular un ecualizador analógico en los que cada potenciómetro controla la ganancia de la banda que representa. Así se muestra por pantalla un interface gráfico que intenta simular en la medida de lo posible a estos dispositivos. Se coloca la información correspondiente a cada banda sobre, la frecuencia central de la banda, la ganacia y atenueción máxima para el desplazamiento máximo del cursor y la amplificación actual que se está aplicando en cada momento para esa porción del espectro de frecuencias que se maneja con él. Un deslizador que imita al cursor de los potenciómetros lineales de que suelen estar compuestos y por último a semejanza de las mesas de directo y por interés del autor, se han incluido dos botones adicionales; Mute y Solo, el primero silencia la banda en cuestión y el segundo la función inversa, es decir, se silencian todas las bandas excepto la seleccionada con el botón de Solo. Se permite al usuario que active varios botones a la vez, tanto del mismo tipo como de tipos distintos al mismo tiempo, se ha implementado la lógica correspondiente para que funcionen correctamente varios a la vez, permitiendo silenciar varias bandas en particular o escuchar solamente las seleccionadas. Una vez descritas las funcionalidades que se pretenden aportar con esta sección se pasa a describir como manejar las opciones para conseguirlas. Por lo tanto se describen los distintos elementos del diálogo de opciones de filtrado. Al igual que en el modo Análisis se puede llegar a este diálogo pulsando el botón que lo invoca o al lanzar el modo Filtrado directamente por primera vez, los dos diálogos que preceden a éste se describirán en siguientes apartados. Tambien mencionar antes de continuar que todas la colisiones teóricas que se puedan producir al seleccionar parámentros del Filtrado en particular, así como la frecuencia de muestreo o los puntos FFT, al igual que antes, la aplicación es capaz de tomar medidas y evitar posibles fallos, siempre prevalecen la frecuencia de muestreo y los puntos FFT frente a cualquier otro parámetro.

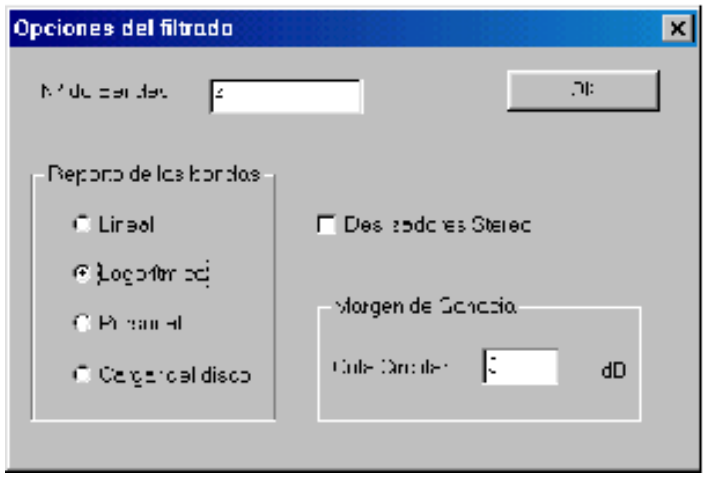

Figura 2.4: Diálogo de opciones de filtrado

- **N^o de bandas**, con este campo se le indica a la aplicación cuantos potenciómetros queremos que se muestren por pantalla, o lo que es lo mismo, en cuantas bandas se quiere dividir el espectro total disponible. Esto es así de pendiendo de la **frecuencia de muestreo** y del ya mencionado teorema de Nyquist respecto a la frecuencia

más alta disponible, y por supuesto sin que se sobrepase el espaciado mínimo de la FFT que ya se comentó anteriormente que es de 1000 y 1102, para frecuencias de muestreo de 44100 y para el resto, respectivamente.

- **Reparto de las bandas**, en este item, la opción por defecto es un reparto **logarítmico**, como corresponde a los dispositivos que se pretende emular y que así vienen configurados por la naturaleza de la señal a tratar (audio). Sin embargo se proporciona la opción de un reparto **lineal**, aunque quizás la opción más útil para la emulación de un ecualizador hardware es la posibilidad de centrar las bandas en las frecuencias que se deseé, es decir, una configuración **personal**. La aplicación es capaz de tener encuenta todas las limitaciones que puedan surgir si se vulneran requisitos de cualquier índole ya descritos para los otros casos, es decir, la frecuencia de Nyquist, los espaciados mínimos de la FFT, así como errores en la introducción de los datos. Al terminar esta fase se da al usuario la facilidad de guardar al disco la configuración para reutilizaciones posteriores, que son accesibles desde la última opción disponible **Cargar del disco**.

- **Margen de ganancia**, con esto se proporciona la posibilidad de configurar los límites tanto de la ganancia como de la atenuación de los potenciómetros emulados. Se le permite un margen de 1 y 10 a -1 y -10 dB, puesto que es un margen circular.

- **Deslizadores Stereo**, esta opción será de utilidad en caso de capturas Stereo en el que se dispone de dos canales, al activar esta opción solo se mostrará por pantalla una fila de potenciómetros que controlarán los dos canales a la vez. Por supuesto en caso de mono se ignorará su valor.

Antes de terminar con la descripción de esta sección se detallarán algunas limitaciones de la misma, cuando se pulsan sobre todo los botones de Solo, lo que se genera es un filtro muy picudo y estrecho, con lo cual se pueden llegar a escuchar ciertas discontinuidades en las uniones de las sucesivas FFT's, se ha implementado un solapamiento del 75% pero si pedimos filtros muy abruptos se notarán. No hay que olvidar que un filtro con forma perfectamente cuadrada en frecuencia es irrealizable.

2.3.3 Grabar a disco

Esta capacidad está disponible desde el lanzador principal de la aplicación, *Ecualizador.exe* y permite almacenar capturas para su posterior análisis con la opcion correspondiente del modo análisis, anteriormente descrito. Se mostrará la ubicación y el nombre del archivo que recogerá los datos y también se proporciona información del tamaño que va adquiriendo el archivo según avanza el tiempo de captura. Para esta funcionalidad solo hay que seleccionar en el diálogo de opciones de captura o muestreo el formato en que queramos que se realice ésta. Una vez llegados a este punto solo falta recordar que el aumento en la calidad de la captura incrementa el tamaño del archivo que albergará el resultado de la captura para igual tiempo de grabación. Para terminar comentar que se puede grabar a disco como máximo con calidad CD un timepo de 6.76 h que son 4GB de archivo, evidentemente cuanto menor sea la calidad más tiempo se podrá grabar.

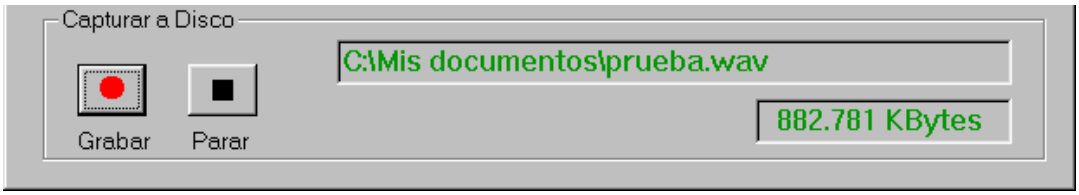

Figura 2.5: Pantalla de grabación

2.3.4 Opciones de captura o muestreo

Como se ha mencionado en apartados anteriores, este diálogo se encarga de definir el formato que tendrá la señal capturada en el mundo digital. Se ha tomado la decisión de que el formato aquí seleccionado sea también el que se utlize para la reproducción para que no se produzca ningún cambio de formato en el procesamiento de la señal, solo se ignora lo que indique este diálogo cuando se reproduce un archivo para su análisis, puesto que se toma el formato del archivo, se escuchará en el formato en que se grabó también sucede esto cuando se carga un archivo .ecu en modo filtrado. Como se ha indicado también con anterioridad, se mostrará el diálogo por pantalla, bien por que se pulsa el botón que lo invoca o por que la aplicación lo crea necesario, si necesita inicializar parámetros necesarios para lanzar algún modo.
Con él se puede seleccionar la frecuencia de muestreo, para definir cada cuanto tiempo se tomarán datos de la señal analógica, este parámetro condiciona sobre todo el ancho de banda que se procesará correctamente después, aunque también influye fuertemente en la calidad de la señal, se ha cuantificado el rango de frecuencias permitidas a 8820, 11025 22050 y 44100 Hz puesto que con éstas se observa de sobra el efecto del teorema de Nyquist y se ha colocado la superior atendiendo a los formatos de la calidad de los CD's y para no desperdiciar tiempo de proceso de los microporocesadores en el cálculo de la FFT. Los bits por muestra que definen el margen dinamico, a primera vista la calidad parece poco afectada por este parámetro pero condiciona fuertemente el modo Filtrado en cuanto a los márgenes de atenuación/ganancia con que se puede alterar la señal, se disponen de 256 y 65536 estados distintos para 8 y 16 bits respectivamente. Por último se permite definir el modo de captura o si ésta se realiza en mono o stereo, es decir, 1 ó 2 canales, esto se ha definido así puesto que e la actualidad todos los equipos de audio trabajan en stereo y la posibilidad de usar mono para máquinas poco potentes, que de esta manera procesan la mitad de datos o para configuraciones de equipos que así lo requieran.

2.3.5 Selección de dispositivo

Se ha dotado con este diálogo a la aplicación para que el usuario pueda elegir entre varios dispositivos si dispuesiera de ellos. Aunque la mayoría de los ordenadores poseen solamente una tarjeta de sonido se mostrarán dos dispositivos uno correspondiente al hardware y otro que corresponde al seleccionado en Multimedia del Panel de control y que es el que usa Windows por defecto, en ese caso da igual la selección porque se usa el mismo hardware. En caso de que existan varias tarjetas instaladas en el sistema y por supuesto, funcionen correctamente con DirectX se podrá elegir la que se dessé y beneficiarse así de las capacidades de cada una de ellas.

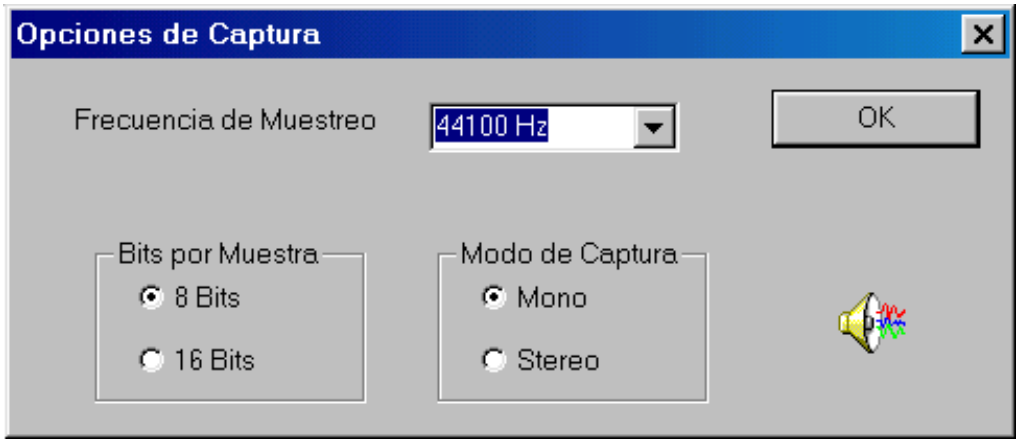

Figura 2.6: Diálogo de opciones de captura

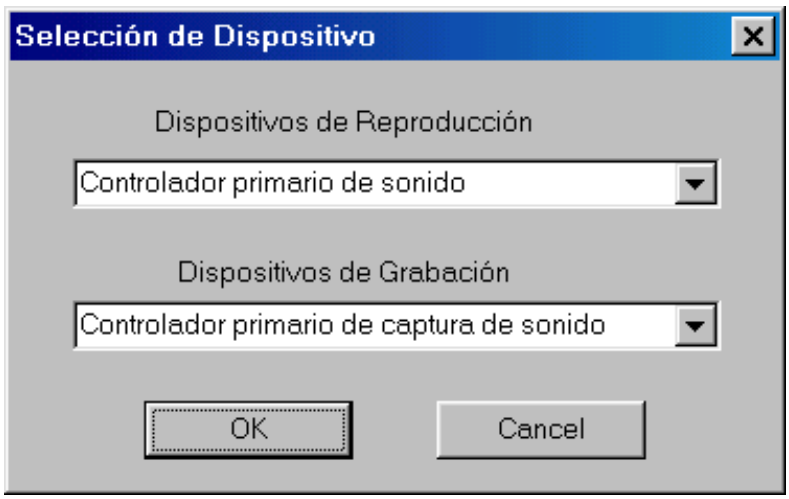

Figura 2.7: Diálogo de Selección de dispositivo

2.3.6 Informacion del hardware

Se han aprovecahdo las capacidades de DirectX para obtener información del hardware y poder mostrarla al usuario, se ha dividido en dos diálogos independientes para captura y reproducción. Estos diálogos son únicamente para mostrar datos por lo que no se permite cambiar nada de lo que en ellos se muestra. Se pasa a explicar cada uno de los apartados de que consta cada uno de ellos.

Información de reproducción

Lo primero que se da son los márgenes de la freuencia de muestreo, son las frecuencias que se pueden mantener de forma constante, las dos cotas máxima y mínima. A continuación se proporciona información sobre los tipos de buffers que se pueden procesar y de qué tipo son, es decir, los buffers hardware son los que se mezclan vía hardware, lo hace la tarjeta y no gasta ciclos de la CPU, buffers estáticos o que contienen un sondo entero, y por último buffers canalizados o que servirán para la reproducción de sonido de larga duración realizando sobre ellos un refresco de datos a medida que avanza la reproducción.

A continuación lo mismo pero para sonido 3D. La memoria Hardware disponible para la creación de buffers, aunque es solo una estimación, está bien ir testeándolo según se van creando buffers. Por último se pueden comprobar ciertas banderas de control, como si Microsoft ha probado y certificado ese controlador, si hay un rango contínuo entre las frecuencias de muestreo tope; suele ser un espaciado de 10Hz, si se está emulando; lo que quiere decir que no hay instalado ningún controlador con capacidad de funcionamiento de DirectSound, si se facilita el mezclado de buffers de 16 bits y Stereo.

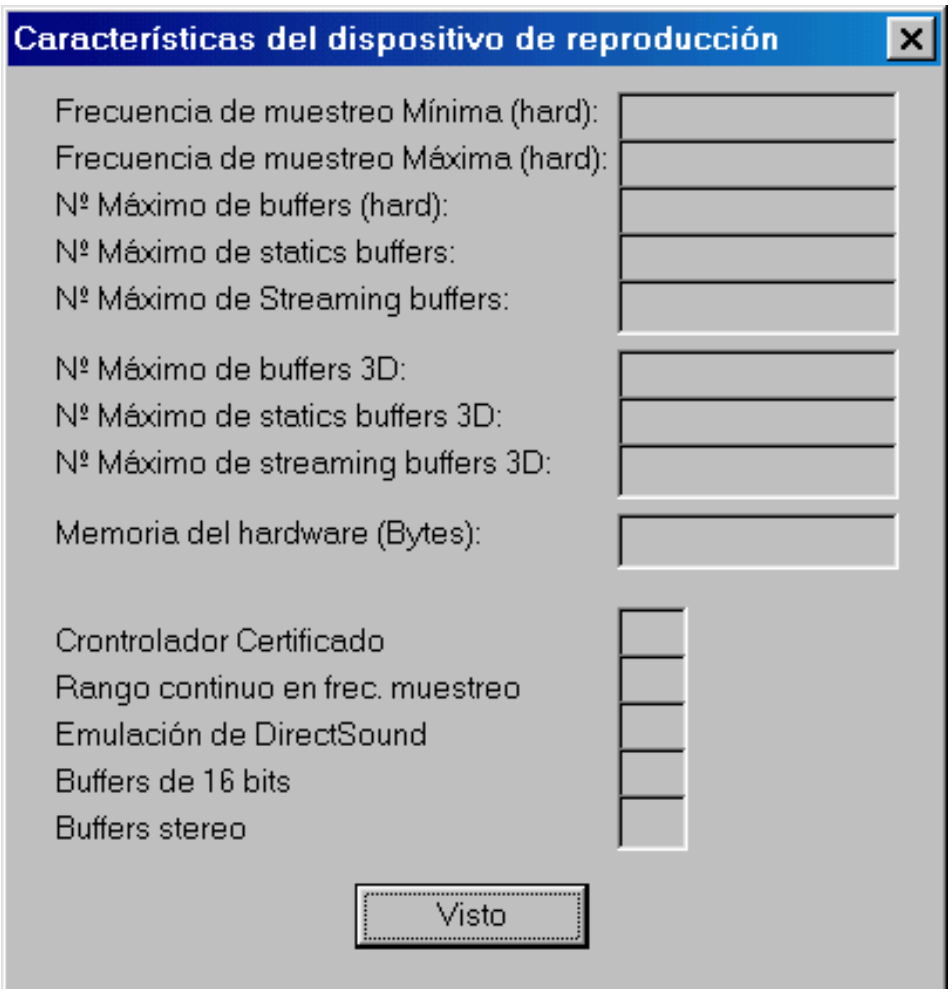

Figura 2.8: Información de reproducción

Información de captura

Esta función devuelve una estructura con únicamente 4 DWORD que se ajustan a unos flags o banderas que testean si el driver está o no certificado por Microsft y si se está emulando el driver con lo que implica, esto es, el uso de las funciones estándar de audio, las API's de Windows. La configuración actual de Windows para la captura y normalmente de reproducción. Por último el número de canales soprotados por el dispositivo, Mono o Stereo.

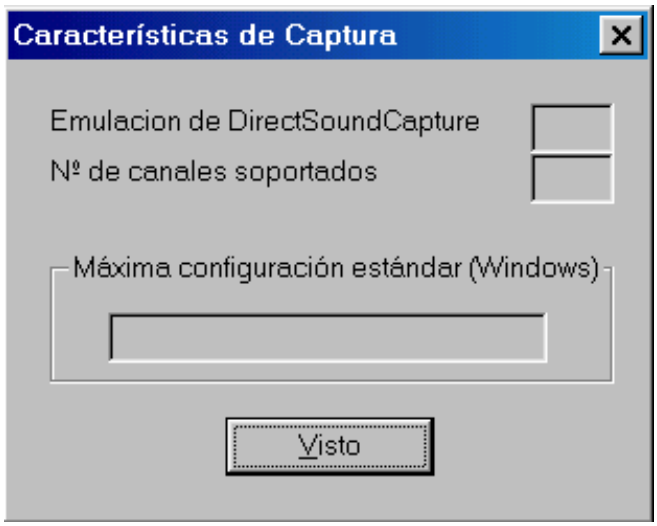

Figura 2.9: Información de captura

2.4 Guía de programación

Este apartado pretende dar unas líneas generales para la compresión del código fuente que se lista en el siguiente capítulo. Se quiere mostrar en ideas y conceptos lo que se ha implementado, pero sin llegar a bajo nivel, es decir, no se explicará a nivel más abajo de funciones y solo aquellas que tengan alguna finalidad especial. El código fuente está, a mi entender, lo suficientemente comentando para que leyendo previamente esta sección, y por lo tanto, teniendo presente que es lo que se quiere conseguir o a dónde se quiere llegar, se pueda mirar el código y entenderlo. Con la pesadez que leer el código de otro conlleva.

De esta manera, se explicará en esta sección las líneas generales, algorítmicamente hablando, de la aplicación, en particular, se explicarán una por una, primero, las clases principales, o al menos las creadas para esta aplicación. Cualquiera que haya utilizado un Visual C++, sabe que para empezar un proyecto en el que se haga uso de la MFC, o armazón de la aplicación, hay un asisitente que se encarga de generar todo eso, el armazón de la aplicación. Por lo tanto solo se describirán las clases, bien diseñadas enteramente por mi, o bien las generadas por el asistente que se hayan modificado tanto que sea necesario su comentario, en caso contrario se supone que estan ahí, pero no se dirá nada de ellas. Esto sucede con la clase de la vista de Filtrado, puesto que se realiza un uso exhaustivo de la vista, constituyendo éste el interface con el usuario, para el resto de los programas no se menciona. Una vez que se hayan descrito las clases se describirá en cada modo de ejecución de la aplicación, como ya se ha dicho en repetidas ocasiones, como se llama o lanza a cada objeto que representa a las clases ya descritas para conseguir el resultado deseado.

Tampoco se va a entrar en como se maneja el entorno integrado del compilador, para las tareas de creación de mensajes, teclas virtuales, adicción de funciones miembro en las clases proporcionadas por al armazón o tantas otras tareas necesarias para que la apicación pueda compilarse y, sobre todo, tenga el aspecto final que posee, ya que han sido necesarios la creación de recursos de todo tipo, desde simples iconos, hasta archivos

bmp's o punteros para el ratón. Sí se comentará, que en las opciones para compilar en modo release se ha empleado la optimización para Pentium con lo que implica de incompatibilidad hacia microprocesadores de arquitecturas anteriores.

2.4.1 Clase CDialog_Muestreo

Esta clase al igual que todas las que empiezan por CDialog, dan soporte a diálogos que se muestran en cierto momento para recoger datos que el usuario debe proporcionar a la aplicación. En este caso en particular, recoge los datos referidos a las opciones de muestreo o de captura. No se dirá nada acerca de las funcionalidades que proporciona el armazón al heredar cada una de ellas de CDialog de la MFC, puesto que no es el propósito de esta memoria explicar la MFC, para más información [1]. Como siempre se detallarán las funciones miembro de la clase que se consideran importantes para facilitar la compresión.

Inicializacion() Se le obliga a pasar por aquí antes de mostrar nada por pantalla, por lo que se encarga de comprobar que existe el archivo OPCIONES.CFG, para leer la configuración que en él haya, sino existe, puede ser la primera vez que se ejecute, se proporcionan unos valores por defecto que aparecerán cuando se muestre el diálogo en la pantalla, en caso de que ni siquiera se pudiera crear el archivo se mostraría el consecuente mensaje de error por pantalla. También se proporciona una inicialización de los parámetros gráficos puesto que van en el mismo archivo, OPCIONES.CFG. Antes de terminar se invoca al diálogo, para que el armazón lo ponga en la pantalla.

OnOk() Se llega a este punto cuando el usuario pulsa el botón de Ok, que aparece en todos los diálogos y que confirma los cambios, se le indica al armazón con una llamada a la clase base para que realice el intercambio de los nuevos datos con las variables miembro de la clase. Ahora se está en disposición de comenzar a manipularlos. Por lo tanto se escriben en el archivo OPCIONES.CFG, después se comprueban los posibles efectos laterales en los parámetros de filtrado para que se remedien y no haya incoherencias, para terminar forzando un refresco de los indicadores permanentes que informan sobre la configuración actual de captura.

Serialize() Esta función se hereda de las clases base que proporciona la MFC, como se comentó al principio, tiene sobre cargados los operadores de » y « para leer y escribir en el disco, solo hay que porporcionarle un objeto CArchive sobre el que relizar ambas operaciones. El parámetro Salvar, le informa si hay que escribir o leer, aunque es redundante, proporciona claridad. En OPCIONES.CFG se escriben las variables que contienen la frecuencia de muestreo, los bits por muestra y el modo, mono/stereo. Después se almacenan las estructuras que definen a los dispositivos tanto de captura como de reproducción seleccionados por el usuario y por último todos los parámetros que se permite al usuario modificar en el diálogo de opciones gráficas, junto con el nombre del archivo que desea analizar si se activó esa opción.

Grabar() Se emplea para forzar un acceso al disco para escribir los datos que haya en ese momento, preparando tados los objetos necesarios para ir a "Serializar".

2.4.2 Clase CDialog_Graficos

Esta clase soporta el diálogo que recoge las opciones proporcionadas por el usuario en cuanto a los parámetros que definirán la puesta en pantalla del análisis gráfico. Se explicarán más detalladamente las funciones miembro más importantes.

Inicialización() Identica a la del diálogo de muestreo, puesto que los datos de la configuración gráfica y de muestreo se almacenan en el mismo archivo OPCIONES.CFG. La única diferencia es que después de invocar al diálogo y haberlo cerrado se comprueba que no haya habido algún error en las preferencias introducidas por el usuario, en cuyo caso se resetea y se colocan los valores por defecto de los parámetros que pueden generar error, esta es la razón por la que siempre que se deseé ruido rosa o analizar un archivo hay que especificarlo explícitamente cada vez viniendo hasta el diálogo de opciones gráficas. Se ha optado por este reset para evitar posibles conflictos y para forzar al usuario a que utilice sólo las funciones avanzadas cuando las quiera usar, no por equivocación o por selecciones anteriores.

OnOk() Al pulsar el botón de Ok de cualquier diálogo se llega a este punto, se le comunica al armazón que realice las operaciones necesarias con los datos introducidos en el diálogo para poder usarlos en la clase. Lo primero es establecer la prioridad entre las funciones avanzadas del modo Análisis, ruido rosa o analizar un archivo; y por supuesto abortar el lanzamiento en caso de cancelar el archivo a analizar si se eleigió esta modalidad. Por último se escriben los datos en OPCIONES.CFG para que los utilice *Analisis.exe*.

DialogoAbrir() Se tapa la gestión de la puesta en pantalla del diálogo estándar de Windows que permite al usuario la selección de un archivo. Realizando todas las conversiones y tratamiento de los datos proporcionados por esta clase, así como comprobar que se puede abrir el archivo seleccionado antes de seguir, para que lo pueda utilizar el resto de la aplicación.

Serialize() Esta función es idéntica a la que poseé *CDialog_Muestreo* por la razón antes expuesta del uso del mismo archivo.

2.4.3 Clase CDialog_Filtrado

Esta clase aparte de dar soporte al diálogo en el que se definirán las opciones para realizar el filtrado, se realizan ciertas tareas de preconfiguración, es decir, se realizan cálculos que luego el programa *Filtrado.exe* leerá del archivo FILTRADO.CFG y usará directamente. Se han segmentado así estos cálculos previos para que cuando se inicie *Filtrado.exe* tenga los datos necesarios para obtener las dimensiones de la ventana, pues, hay que tenerlos antes de mostrarla en pantalla, por eso, se ha aligerado de algunos de ellos. Para verlo con más claridad se explicarán las funciones miembro con más detalle.

Inicializacion() Simplemente pone unos valores iniciales que aparezcan en el diálogo, para cuando se invoca por primera vez.

OnOK() Se provoca el intercambio de valores, y se llama a la función que realiza esos cálculos de preconfiguración.

Configurar() Lo primero es obtener información del diálogo de muestreo puesto que hay datos que están relacionados, como ya se ha explicado en varias ocasiones, através del teorema de Nyquist o de los espaciados mínimos en el cálculo de la FFT. Por lo tanto una vez que sabemos esos datos de la captura se pueden comprobar que no se sobrepasa ningún límite, tanto en espaciado como en ancho de banda. Una vez testeadas las restricciones, se puede proceder al cálculo de las frecuencias que proporcionará la FFT, para configurar acto seguido las estructuras de datos que harán funcionar correctamente el filtrado.
Se discrimina según la modalidad de reparto de la bandas que haya solicitado el usuario, se verán una por una en el mismo orden que están en el código:

- Obtención del disco. Lo primero es mostrar en pantalla el diálogo que proporciona la MFC para la selección de un archivo .ecu. Si el usuario cancela o el archivo seleccionado no se puede abrir se muestra un mensaje en pantalla y se aborta la operación. Por el contrario si todo ha ido bien sólo resta completar los arrays necesarios, puesto que como se explicará más adelante, cuando se escribe en el disco todos los datos ya han sido comprobados y son válidos. Así se crea un array que contendrá las frecuencias que indique el archivo y se genera otro array que será el encargado de saber cuantas componentes forman cada banda. Para terminar cabe la posibilidad de que no coincidan las frecuencia de muestreo, la que está definida en ese momento y la que indica el archivo, luego se comprueba y si hay incoherencias se corrigen.
- Personalizar las bandas. Es la parte complementaria al punto anterior, se van mostrando diálogos consecutivos para que el usuario introduzca las frecuencias centrales que definirán las bandas, tantos como haya especificado en el número de bandas, si cancela, se aborta mostrando el mensaje correspondiente de error. Si este paso ha concluido satisfactoriamente se procede a comprobar la validez de los datos proporcionados por el usuario, hay que comprobar varios parámetros. Primero se buscan frecuencias repetidas, después se ordenan de menor a mayor puesto que es así como trabaja la FFT con las componentes en frecuencia, se comprueba la restircción del teorema de Nyquist, y por último se coloca un mínimo, como otro cualquiera. A estas alturas es posible que el número de bandas haya disminuido, luego se tiene encuenta esta posibilidad al "pasar a limpio" las frecuencias de las bandas que utilizará luego *Filtrado.exe*. Ahora ya se está en condiciones de definir los arrays de las frecuencias centrales y el de las agrupaciones, como en casos anteriores. Antes de mostrar el ecualizador en la pantalla se ofrece la posibilidad de guardar a disco la configuración introducida, eso sí, pasadas todas las restricciones y lista para ser usada sin más. Si el usuario desea, se escribirá en un archivo .ecu y sino será descartada al terminar la ejecución.
- Cálculo automático de las bandas. Se entra en este modo siempre que se seleccione **Logarítmica** o **Lineal**. Primero se establece el límite inferior en las frecuencias en 20 Hz, se comprueba que no coincidan el número de bandas con las componentes que proporciona la FFT. Si es así se asignan directamente sin más, en caso contrario, lo más normal, hay primero que calcular el barrido por el espectro, se discrimina por la selección hecha en el diálogo, si es **Logarítmica** se implementa un barrido que satisfaga este barrido, comprobándose que no haya intervalos menores que el espaciado de

la FFT, en la escala lineal es mucho más fácil realizar el barrido, a parte, de no existir ese problema nunca. Por último se configura el array que controla las agrupaciones. Antes de salir se testea cuantas filas de potenciómetros habrán de aparecer en la pantalla, debido a la selección de **Mono/Stereo** y **Bandas Stereo**, esta última permite controlar los dos canales con un solo mando, y una vez solventado esto se puede proceder a escribir en el disco a FILTRADO.CFG liberando las estructuras de datos que ya no son útiles.

Serialize() Esta función como siempre se utiliza para el acceso a disco, pero en este caso no solo para guardar la configuración en FILTRADO.CFG, sino que también para acceder a OPCIONES.CFG, cuando sea necesario, por eso necesita otro parámetro más, pero el funcionamiento es similar al de las otras. Se almacenan como datos específicos para el filtrado el número de bandas final, despues de comprobar errores, las filas de potenciómentros que habrá que mostrar, el array de agrupación, y las banderas que indican el estado de **Bandas Stereo** y la ganancia que se aplicará a las bandas.

Solo queda por explicar la otra clase, simétrica a ésta que recoge los datos del archivo FILTRADO.CFG, en el ejecutable *Filtrado.exe*. Se ha utilizado una clase de diálogo para beneficiarse de la herencia antes comentada y así aprovechar las ventajas del acceso a disco. En este caso no se hace nada más, no hay que hacer ningún cálculo, basta con recoger los parámetros y los arrays precalculados en *Ecualizador.exe*.

2.4.4 Clase CDialog_Personal

Se implementa esta clase para mostrar un diálogo adicional que facilita la introducción de las frecuencias centrales de las bandas que tendrá el ecualizador cuando se ha solicitado la personalización de las bandas. Permite la visualización de las bandas previamente introducidas para guíar en lo posible al usuario, en caso de que sea larga la lista. Como siempre se detallan las funciones más significativas.

OnInitDialog() El armazón ejecuta esta función entre la invocación del diálogo pero antes de hacerlo visible en la pantalla, por lo tanto se aprovecha para colocar los datos que previamente se han ido introduciendo de bandas anteriores, si hay alguna, en caso contrario aparecerá vacío. Al terminar se le indica al armazón que realice el intecambio de datos, ya que este es bidireccional, es decir, ahora pasa los datos de la variables a las estructuras que los pondrán en pantalla, mientras que en las funciones de OnOk(), como se menciona en apartados anteriores, la transferencia se hace en sentido inverso.

OnOk() y OnCancel() Se usan para tener al corriente una bandera que informará de si ha cancelado el usuario la introducción de datos.

2.4.5 Clase CDialog_CapsCaptura

Esta clase como todas las de diálogos sirven de soporte a los diálogos que se muestran por pantalla, este diálogo difiere del resto en que solo pone información en la pantalla y no recoge nada. De esta manera la principal función miembro es la de inicilizar los datos

antes de hacer la transferencia entre las variables miembro del objeto y las que muestran su contenido en la pantalla, se procede a comentar su implementación.

OnInitDialog() A diferencia de otros diálogos, en este se accede directamente a los apartados del diálogo para colocar en cada uno de ellos el texto necesario, que se discrimina con las estrtucturas de selección necesarias, en primer caso se comprueba un valor boleano, para saber si se emula DirectSoundCapture, pero luego hay que comporbar que configuración se ajusta a la devuelta por la instrucción de obtención las capacidades. Una vez que se sabe se pone en pantalla.

2.4.6 Clase CDialog_CapsReproducción

Al igual que el anterior pone información por pantalla sin recibir ni tener que recoger nada de parte del usario. Muestra por pantalla algunas de las capacidades que poseé el hardware que se emplea para ello. Como el diálogo que muestra la captura solo es relevante la función que inicializa todos los datos para su posterior visualización en la pantalla.

OnInitDialog() Hay dos zonas de datos, en la primera hay que colocar valores numéricos en las casillas correspondientes, lo cual implica una conversion de int a CString, pero la MFC proporciona una función que realiza todo ello. En segundo y último lugar unos datos con valores boleanos que son los bits que forman una DWORD, que hay que comprobar y colocar la cadena correspondiente.

2.4.7 Clase CMainFrame y CEcualizadorDlg

Ambas clases, realizan la función de marco principal o general, esto es relacionan el resto del armazón con el programador. Desde aquí se gestiona y/o personaliza todo lo relacionado con lo que no es la vista. Se procede a centralizar en este apartado la explicación de las 3 clases de cada ejecutable que realizan esta función. Emplean *CMainFrame* en el caso de *Analisis* y *Filtrado*, puesto que hacen uso de la arquitectura documento-vista. Pero no es el caso de *Ecualizador*, puesto que se ha utilizado una arquitectura de diálogo para la aplicación al crearla con el asistente, con lo que estrictamente hablando no es marco, pero por la funcionalidad que proporciona se ha incluido aquí. Con esta aclaración se procede a describir una por una las clases involucradas:

Analisis Como se acaba de comentar *CMainFrame* controla todo lo que sale por pantalla y no es la vista, es decir, el tamaño de la ventana macro, los botones que aparecerán en la parte superior derecha (minimizar, restaurar y cerrar), los botones de la barra de herramientas, si hay barra de estado, el menú, la gestión de los mensajes dirigidos a la aplicación y todo lo que se pueda configurar del marco de la aplicación. Por esto se utiliza el constructor para inicializar las variables globales que controlan el estado de la aplicación. Como en otros casos se detallarán las funciones miembro más importantes para facilitar la comprensión.

PreCreateWindow() Con esta función se determinan las dimensiones de la ventana, así como lo botones que se quiere que aparezcan.

OnComenzarAnalisis() Desde aquí se lleva a cabo todas las tareas necesarias que preparen el camino para la ejecución con éxito de *Analisis.exe*. Por lo

tanto, se comprueba que no haya reentradas, no hay que olvidar que existen unas variables globales que saben en todo momento en que estado está el programa. Acto seguido se leen los parámetros que ha preparado *Ecualizador.exe* del fichero OPCIONES.CFG, que en caso de no existir abortan la ejecución. Llegados a este punto se utilizan las funciones que sirven de interfaz con DirectS para; crear los objetos necesarios, lo mismo para CDirectDraw y también para CFft. Se toman las decisiones necesarias para la inicialización de DirectS, como se comentó cuando se describió esta clase, según que proceso va a dirigir el trasiego de datos, es decir, si la captura o la reproducción de un archivo, dato que se extrae del archivo. Para terminar con DirectS se inician los buffers necesarios y se lanza el hilo trabajador que se encarga de la gestión autónoma de los eventos que se vayan generando. Ya sólo queda la parte gráfica de la puesta en pantalla de los datos, así que se inicializan los planos (surfaces) necesarios, se realizan los cálculos necesarios para poder poner las columnas de led's en la pantalla y se termina lanzando el temporizador que pedirá los refrescos de la pantalla, se ha elegido un refresco cada 30 msg, siendo el límite de los 25 Hz de barrido vertical 40 msg.

OnTimer() Esta función proporcionada por el armazón se encarga de la atención cada vez que se cumple la temporización, así, se comprueba el identificador del temporizador, puede haber varios. Resultando esta comprobación satisfactoria se fuerza el refresco de la pantalla, pero identificando la duración de éste con eventos, para poder modificar datos que se estén representando en ese momento y evitar así acceso a datos en momentos críticos. Antes de salir se testea si el usuario desea terminar para lanzar el proceso de parada desde aquí. La colocación de este test en este lugar fuerza a la sincronización de la orden de parada con el refresco de pantalla.

OnParar() Se encarga de parar toda la aplicación, así lo primero que se hace es activar el evento que controla la parte del audio, en seguida se termina con la temporización y consecuentemente el refresco de la pantalla. A diferencia de las liberaciones y desasociaciones de interfaces que se llevan a cabo en el hilo trabajador en caso de la gestión del audio, para DirectDraw se llevan a cabo ordenándolas desde aquí. Y por último se fuerza un refresco de la ventana, pero con el control restaurado a la GDI de Windows para lanzar el mensaje que fuerza la salida de la aplicación.

Serialize() Como en el resto de clases se utiliza la herencia para el acceso al disco, así que no se hace ningún comentario más.

OnClose() Se ejecuta el código que haya aquí antes de cerrar el programa, por lo tanto hay dos formas de salir, una habiendo cerrado todo previamente, solo liberar las estructuras asignadas para la Fft o si se está analizando pues forzar la parada de la gestión del audio así como del refresco de la pantalla y sin olvidarse de liberar toda la memoria que ha sido asignada dinámicamente para la correcta ejecución.

OnError() En caso de que exista un fallo o cualquier anomalía que impida el correcto funcionamiento de la aplicación, a parte de mostrar el diálogo correspondiente ha ser capaz de salir del programa sin problemas, luego se encarga de saber en que punto del lanzamiento se ha producido y por consiguiente

qué pasos dar para volver al estado inicial. Al terminar se fuerza la salida al sistema operativo.

Filtrado La aplicación también sigue la arquitectura documento-vista del anterior, por lo tanto es la misma filosofía, solo se diferencian en que ahora se usan más elementos de la ventana marco que en *Analisis.exe* por lo que hay más funciones miembro que se proceden a detallar.

LeerArchivo() Esta función se encarga de los preparativos para leer el otro archivo necesario para Filtrado, aunque solo lee la información necesaria para los constructores y las inicializaciones de la ventana, es decir cuantos potenciómetros virtuales hay que poner en la pantalla. Esto se obtiene de FILTRADO.CFG.

OnCreate() y PreCreateWindow() Entre ambas funciones se personaliza el marco de la ventana, en particular se indica que botones de los que interactúan con el sistema operativo están activos, en este caso sólo cerrar y minimizar, y configura la barra de estado diseñada para la aplicación. De la misma manera se definen las medidas de la ventana que se han obtenido de los datos leídos del fichero FILTRADO.CFG, también se definen otras submedidas que usará la vista para realizar los scrolls de la vista.

Serialize() No se dirá nada más de esta función ya explicada en ocasiones anteriores, solo comentar que se encarga del archivo OPCIONES.CFG.

OnComenzarFiltrado() Como su nombre indica realiza las tareas necesarias para inicializar correctamente los distintos objetos y que funcione el filtrado. Después de evitar reentradas, se lee OPCIONES.CFG, con el resto de información necesaria creándose los interfaces a DirectSound, solo una vez por ejecución naturalmente. Se ordena la creación de los planes necesarios para los cálculos de la FFT y se comienza la inicialización de los buffers DirectSound, se lee de nuevo el fichero FILTRADO.CFG pero esta vez entero. Con estos nuevos datos ya se está en condiciones de configurar el filtro. Se fuerza un refresco de la venta para que aparezcan los potenciómetros en la vista, para terminar, como en Análisis, se lanza el hilo trabajador que gestionará el audio independientemente para que no interfiera en la gestión de la vista y no haya sensación de una aplicación lenta, puesto que se actualizarán los elementos que están en la pantalla en cuanto sea necesario.

OnParar() En este caso simplemente hay que parar la atención a la gestión de los eventos generados por DirectSound, una vez hecho esto se procede a repintar la vista, para que los nuevos cambios tengan efecto y se actualizan las variables que controlan el estado de la aplicación.

OnUpdatexxx() Estas funciones son proporcionadas por el armazón de la aplicación para la gestión de los elementos visuales, y evitar incoherencias entre el estado de la aplicación y lo que señalen los distintos indicadores, tales como botones, barras de estado o menús. Por lo tanto su única misión es comprobar esas variables globales para saber que está haciendo en ese momento y reflejarlo correctamente. Windows refresca estos indicadores cuando está ocioso, por lo tanto se ha hecho uso de esta capacidad para forzar el refresco de la barra de estado en una de estas funciones.

OnError() Su misión es la de recuperar un estado estable en caso de error, por eso ha de saber dónde se produjo éste y obrar en consecuencia deshaciendo lo hecho hasta ese punto y poder salir de la aplicación al sistema operativo de una forma elegante.

OnClose() Como ya se ha mencionado se pasa por aquí antes de abandonar la aplicación, por lo tanto se comprueba si se está filtrando, el usuario pulsó Alt+F4 sin parar el Filtrado, para detener el proceso y liberar la memoria pedida al sistema. De todos modos siempre se libera aquí la memoria solicitada por la FFT.

OnSalir() Simplemente llama a OnClose(), se han duplicado estas funciones por los nombres que genera el armazón en la atención de los mensajes.

Ecualizador Como se ha comentado al principio del apartado, en este caso, la aplicación no tiene la arquitectura docuemnto-vista, sino que se basa en un diálogo modal. A pesar de esto, ambas clases tienen el mismo cometido, por eso se explica en este apartado, es decir, ambas controlan la ventana macro. Por lo tanto se procede a comentar las funciones más importantes.

CEcualizadorDlg() En el constructor además de inicializar las varibles globales que como hasta ahora se encargan de saber el estado de la aplicación, se comprueba mediante una librería de enlace dinámico, que se ha instalado a tal efecto, dsetup.dll de DirectX, la versión instalada en el sistema, en caso de no tener instaladas la versión 6.0 o superior se termina la aplicación sin posibilidad de realizar ninguna tarea, eso sí, mostrándose el consecuente mensaje por pantalla.

OnInitDialog() Esta función es idéntica a la que poseen el resto de diálogos, pero se usa para añadir unos bmp a los botones más importantes.

OnPaint() Se utiliza esta función para acceder al diálogo y pintar cosas en él, pues como se sabe de los anteriores los cambios en los diálogos se canalizan a través de *OnInitDialog()* antes de mostrarlo por pantalla, y pulsando *Ok* al salir, por tanto hay que forzar el refresco para cambiar cosas sin pulsar *Ok*. De este modo siguiendo un paralelismo con la arquitectura documento-vista esta misión la desenpeña la función miembro *OnDraw()*. Por esto, se pinta los bytes que ocupa la grabación, coloca el nombre del fichero en el que se está grabando y coloca la configuración actual de captura.

OnAnalisis() Cuando se pulsa el botón del análisis se llega a esta función que realiza los preparativos necesarios para llamar a *Analisis.exe*, es decir, en caso de necesitar datos adicionales los pide, invocando los diálogos necesarios, si todo ha sido satisfactorio lanzar el ejecutable *Analisis.exe*. Antes se provoca una liberación del Objeto DirectS para evitar conflictos sobre todo con DirectSoundCapture puesto que en la reproducción DirectSound, poseé mecanismos que gestionan la compartición, varias aplicaciones mandando audio a la tarjeta, pero no ocurre así con la captura, una aplicación cada vez. Se evita esto liberándolo todo.

OnOpcionesGraficos() Se llega a este punto al pulsar sobre OpcionesGraficos, por lo tanto invoca el diálogo correspondiente. Se comprueba que no se esté grabando, no es necesaria la comprobación de estar en otro modo porque la aplicación se paraliza mientras se está ejecutando *Analisis.exe* o *Filtrado.exe*.

OnOpcionesMuestreo() Cuando el usuario lo deseé, pulsando el botón, se invoca el diálogo que pone en la pantalla las opciones de captura o muestreo.

OnFiltrado() Comprobado que no se está grabando, se chequea que todos los datos necesarios para el filtrado estén inicializados, en caso contrario se obliga al usuario a seleccionar en los distintos diálogos. Si todo ha ido bien se liberan los interfaces a DirectSound y DirectSoundCaptura para evitar conflictos con *Filtrado.exe* que también requiere exclusividad en el uso de la tarjeta de sonido. Así que si todo ha ido bien se lanza *Filtrado.exe*.

OnOpcionesFiltrado() Responde a la pulsación de este botón invocando el diálogo de las opciones de filtrado.

OnError() Cuando se produce un error o una situación que imposibilita el seguir con la ejecución normal del programa, se intenta el volver a un estado estable, simpre se intenta que sea muy parecido al inicial y se muestra el mensaje informativo por pantalla.

OnOpcionesDispositivos() Cuando el usuario así lo quiere se invoca el diálogo que permite cambiar el dispositivo que se utilizará para captura o para la reproducción. Lo primero antes de ponerlo en pantalla es comprobar si existe la lista que contiene los dispositivos disponibles. En caso de no existir se crea, utilizando la funcion a tal efecto que poseé la clase CDirectS, si ya existe simplemente se lee y coloca en el diálogo, para mostrarlo por pantalla. Una vez realizada la selección se fuerza un acceso al disco de la captura para actualizar los nuevos datos en OPCIONES.CFG.

OnCapsCaptura() El flujo del programa llega a esta función cuando se quieren ver las capacidades del dispositivo seleccionado de captura. Por lo tanto fuerza las inicializaciones que no estén hechas aún y crea los objetos DirectSound para realizar la petición y colocar los datos en pantalla. No sale sin antes liberar DirectSound que como ya se ha comentado tiene problemas de compartición de recursos.

OnCapsReproduccion() Exactamente igual que el anterior, pero para las capacidades del dispositivo de reproducción.

OnClose() Antes de salir, se comprueba si se está grabando en ese momento, con lo que se detiene el proceso en curso, para terminar liberando hasta la lista de dispositivos.

OnOK() Realiza las mismas tareas que *OnClose()* pero se llega a ésta al pulsar el botón de salir, mientras que la anterior recibe la pulsación de cerrar.

OnGrabar() Prepara la aplicación para realizar una captura de audio a disco, cuando se está realizando este proceso, la aplicación inhabillita la pulsación de cualquier botón que no sea el de parar la grabación para evitar problemas, porque sólo hay una tarjeta de audio luego solo se puede capturar una cosa a la vez.

Lo primero es mostrarle al usuario el diálogo típico de Windows para guardar archivo, para que seleccione donde quiere almacenar el archivo que se cree, si cancela se aborta el proceso. Acto seguido se comprueba si falta por especificar alguna de las opciones necesarias para la grabación, si es así se invocan los diálogos necesarios para que el usario haga lo propio. Si todo ha ido bien se crean los objetos DirectSound y DirectSoundCapture, se crean e inicializan

los buffers que se necesiten con las funciones que a tal efecto tiene la clase CDirectS, para terminar lanzando el hilo trabajador que se encarga de atender los eventos que genera el llenado de los buffers.

OnParar() Solo tiene, como siempre, activar el evento encargado de la parada y que se comprueba en el bucle de eventos para que finalice el proceso.

OnHelp() Enlaza la pulsación del botón que así lo indica con la aplicación que ejecuta el archivo .hlp.

2.4.8 Clase CDialog_Error

En caso de que algún tipo de operación no se pueda ejecutar, se canalizan estos errores para poder informar al usuario de lo que pasa. Por lo tanto esta clase soporta el diálogo que aparecerá por pantalla mostrando el mensaje correspondiente apoyándose en una variable global que contiene el código del error en particular.

Todas las funciones miembros las proporciona el armazón de la aplicación asi que se han acondicionado para que cumplan el cometido deseado, la única que se ha retocado es:

OnInitDialog() La MFC ejecuta esto justo antes de mostrar el diálogo por pantalla con lo que es el lugar idóneo para colocar los datos a mostrar que dependen del error ocurrido, con lo cual se ha colocado una estructura de decisión con los mensajes correspondientes.

2.4.9 Clase CDirectDraw

Envuelve completamente con una clase toda la gestión de los DirectX en lo concerniente al acceso a la VGA para gestión de gráficos 2D. Como siempre se explicarán las funciones miembro más significativas y que porporcionen una ayuda en el entendimiento de la clase.

Crear() Crea y asocia los punteros al objeto DirectDraw através de las interfaces correspondientes que representa al dispositivo de representación gráfica, VGA y su monitor. No hay que olvidar que DirectX utiliza el modelo COM.

IniciarSurfaces() Primero se establece el nivel de cooperatividad que se quiere con el sistema operativo, en este caso en particular, pantalla completa y acceso exclusivo. Seguidamente se configuran las estructuras que definirán los planos de que se quiere disponer (en terminología inglesa "surfaces"), hay que tener simpre el plano principal, que es lo que se ve por pantalla en cada momento (primary surface), un plano adicional para realizar refrescos de pantallas sin parpadeos (back surface) y por último un plano que contendrá las distintas figuras que se pondrán por pantalla, para componer cada frame, según sea necesario. Para esta última operación se usa una función que proporciona la SDK de DirectX que crea un plano (surface) que ubica una copia del bmp en memoria de video, tal cual es, ajustando el plano a las dimensiones y paleta del bmp. Con él en memoria, se realizan copias al plano de atrás (back surface) para componer allí el siguiente frame que se desea mostrar. Estas funciones se detallarán más adelante.

Restaurar() Se encarga de volver la aplicación al estado inicial, es decir, desmantela todos los privilegios que se le pidieron al sistema de panatalla completa y exclusividad

en el uso y por último se le devuelve el control del plano principal (primary surface) a la GDI para que repinte la pantalla que estaba antes y continúe normalmente.

Refrescar() Corazón de la gestión de la pantalla, será llamada cada vez que sea necesario poner nueva información en la pantalla, por lo tanto será un temporizador de unos aproximadamente 30 ms, para que de sensación de fluidez. Básicamente es un bucle en el que se comprueban varios parámetros antes de ordenar la preparación de la nueva pantalla y su posterior intercambio por la actual. Primero se preparan los datos numéricos para que sean fácilmente puestos de forma visual, acto seguido se compone la pantalla en el plano de atrás, y para terminar se intercambia con el plano que actulamente es visible, para mostrarla. Veamos que comprobaciones hay que hacer. Lo primero es comprobar que no se han perdido los privilegios pedidos con anteridad, si es así, se espera y se restaura la memoria de video, si es posible, los planos que son necesarios. En caso de otro tipo de error o que otra aplicación haya cambiado la configuración de la pantalla se sale informando del error. Sino, se procede a realizar el intercambio entre el plano principal (primary surface) y el plano de atrás (back surface), esta operación se conoce como "Flip", solo puede ser realizada si se tienen privilegios de exclusividad y pantalla completa, de ahí que se comprueben ambos antes de intentar ejecutarla, por último este intercambio "Flip" se sincroniza con el comienzo del barrido vertical. Se ha elegido este método de primero componer la nueva pantalla en el frame de atrás, para luego intercambiarla por la que se está visualizando para evitar los desagradables parpadeos si se compusieran los frame en el que se está visualizando. El proceso de "Flip" solo intercambia punteros, por eso es tan rápido consiguiéndose el efecto de fluidez deseado.

PintarPantalla() Se encarga de componer la pantalla que se va a mostrar en el siguiente intercambio de planos. Se distinguen dos zonas, una en la que se pinta el fondo, esto es, el color de fondo propiamente dicho y todo tipo de objetos que no variarán durante la visualización, y luego la colocación de las partes que varirán continuamente. Para el fondo se rellena todo de negro y seguidamente se colocan todos los objetos que acompañan a los "led's", identificadores de los ejes, escalas numéricas, y la ayuda en caso de que se solicite, se ejecuta 2 veces para pintar en los 2 planos que se usan de intercambio y sino hay orden de Scroll no se volverán a pintar, no perdiendo tiempo en ello en los siguientes refrescos. Se usa un bucle de comprobación de parámetros antes de la orden de relleno, parecida a la del "Flip", por si se han perdido los planos y sincronizándose con otras posibles tareas de "Blt".

En segundo lugar se disponen los led's en sus columnas correspondientes, empleándose una estrategia de envejecimiento, para lograr que se vayan bajando despacio, al pasar de un valor alto a uno inferior. Una vez hecho esto el array resultante, que posee tantos elementos como columnas o bandas se van a representar, se compara con los escalones en la escala de las ordenadas, de este modo se sabe cuantos han de ir encendidos, de este para abajo, y cuántos apagados, el resto hasta arriba. De esta manera con 2 bucles anidados uno bariendo las columnas de arriba abajo y el exterior recorriendo todas las bandas, pero teniendo encuenta que puede haber scroll en caso de que no quepan todas en la pantalla, por lo tanto hay que saber que trozo se está representando en cada plano o fotograma.

Configurar() En esta función se hacen todos los cálculos preliminares antes de la puesta en pantalla, tanto de los datos numéricos a representar como de la configuración de la pantalla y de la colocación de los distintos elementos que se colocarán en ella. Se hace una peticion a DirectX para saber la configuración de pantalla que existe en ese momento, y con los datos obtenidos se realizan los cálculos pertinentes para ocupar como mucho 7/10 partes de ella, en cuanto al eje de ordenadas, puesto que el eje de abcisas puede implementar scroll, así, si el usuario se excedió, la aplicación se limita a igualar con este tope. Después se comprueban los espaciados pedidos, es decir, la precisión máxima, si se rebasa, el programa se en carga de hacerla coincidir con el mínimo, 1dB para eje ordenadas y 20, 10, 5, 4 Hz para el eje de ordenadas, según Fs; 44100, 22050, 11025, 8820 Hz respectivamente. Una vez que se ha ajustado a los límites de la aplicación se preparan las estructuras que contendrán los datos. Se procura simpre no mostrar frecuencias más altas de la frecuencia de Nyquist y en el caso de 44100 Hz no se muestran las frecuencias por encima de 20000 Hz, lo que exige la norma Hi-Fi. En la mayoria de los casos no se piden tantas bandas que sea necesario el espaciado mínimo, con lo que hay que agrupar las componentes que se obtendrán de la FFT en bandas que se corresponderán con las columnas de led's, también se realizan esos cálculos aquí para aligerar todo lo posible los cálculos que haya que realizar en el proceso en tiempo real, no hay que perder de vista que los cálculos aquí realizados solo se ejecutan una vez, al contrario que en *Refrescar()* o *PintarPantalla()*. Se inicializan los valores que sean necesarios inicializar y para terminar se determina la escala o el tamaño de los led's según la cantidad de ellos que se pretenda poner en pantalla. De esta forma se establecen unos índices que indicarán a *PintarPantalla()* cual de todos los indicadores que componen el bmp hay que poner en pantalla cada vez, a la vez que se selecciona cuántos serán rojos, cuántos amarillos y verdes el resto.

PrepararDatos() Aquí se gestiona desde el acondicionamiento de los datos que devuelve la FFT, es decir, se calcula el módulo al cuadrado, o lo que es lo mismo, la potencia al cuadrado, no olvidar que la FFT devuelve componentes complejas. Se recuerda aquí que esta clase sirve solo para el modo Análisis, por lo tanto se llama al cálculo de la FFT para obtener las componentes en frecuencia de la señal en el tiempo capturada por la tarjeta de sonido, de esta forma en el dominio de la frecuencia tenemos datos complejos, en esta aplicación se ignora la fase por lo que solo interesa el módulo o potencia de la señal. También se asocian las componentes en las bandas que serán representadas luego por pantalla, realizando la conversion a una escala de dB. Se han proporcionado secciones críticas comunes con el cálculo de la FFT puesto que, acceden a los arrays el cálculo de la FFT y cada vez que lo indica el temporizador hay que mostrar por pantalla el siguiente frame, que implica también el acceso a los datos, por lo que se puede llegar a conflictos si intentan manipular los dos a la vez, para evitar esto se han usado una clase proporcionada por la MFC que bloquea y desbloquea las secciones críticas.

RestaurarMemVid() Se llamará a esta función en caso de que se pierdan los planos asignados de la memoria de la tarjeta de video o del sistema, por cambios a otra aplicación o por cualquier otro motivo, su misión es la de recolocar en memoria de video, o sino, en la del sistema los planos desalojados para que continue normalmente la ejecución.

Destruir() Se encarga de la liberación de todos los planos creados (surfaces) y si se indica de la desasociacion del interface al objeto DirectDraw. También se encarga de dejar el sistema tal cual estaba al principio puesto que todos los datos se albergan en memoria asignada dinámicamente que hay que liberar antes de abandonar el programa.

Por último las funciones importadas de la SDK de DirectX que facilitan la manipulación de bmp's con DirectDraw, se utilizan sólo 3 de todas ellas, no se dice aquí nada de ellas puesto que vienen suficientemente comentadas. Mirar en el capítulo del código fuente para ver que es lo que hacen.

2.4.10 Clase CDirectS

Se crea esta clase para envolver a la tarjeta de sonido junto con las funciones principales que proporciona DirectX para su gestión. De esta manera se facilita una capa de abstracción más alta para gestionarla desde las funciones que proporciona el armazón y poder así relacionar la aplicación con el hardware. En los siguientes puntos se procede a explicar las funciones miembro, sobre todo las públicas que sirven de interfaz con el exterior.

Configurar() En esta función se cogen los datos del diálogo de Captura o muestreo y se procesan los datos para darselos en la forma correcta y poder ser manipulados por el objeto, en particular se pasan los datos de CString a int, para la frecuencia de muestreo, y de bool a int tanto para los canales como para los bits usados por muestra

CrearObjetos() Se crean los objetos DirectSound y DirectSoundCapture que representan a la tarjeta de sonido para la reproducción y la captura respectivamente. En el caso particular de *Ecualizador.exe* se ha prorcionado la capacidad de saber si han sido creados o no para no provocar conflictos al lanzar *Analisis.exe* o *Filtrado.exe*, sobre todo con DirectSoundCapture puesto que DiretSound si puede convivir con otras aplicaciones en el uso del hardware si se establece su prioridad con las funciones a tal efecto. Así se asocian los punteros a las interfaces proporcionadas por la arquitectura COM a los objetos antes mencionados.

InicializarBuffers() Hace exactamente lo que su nombre indica, aunque tiene encuenta varios parámetros que se pasan a detallar. La estructura de gestión de DirectSound y DirectSoundCapture se basa en buffers hacia o desde los cuales van los datos que se capturan o que se reproducirán. Se crea primeramente para DirectSound, un buffer primario, en el que están los datos que se van a reproducir, es el sonido ya mezclado de todos los buffers secundarios que existan y se estén reproduciendo en ese momento. Se mantiene simpre en reproducción para evitar chasquidos en los arranques y paradas por las consecuentes operaciones en las transferencias del DMA [4] sobre todo en los PC's antiguos, puesto que ahora las tarjetas PCI no lo utilizan. En cuanto a la captura con DirectSoundCapture solo existe un buffer en el que se van colocando los datos que representan al sonido que se está capturando en ese momento. Sin perder esto de vista se procede a detallar la implementación. Se configura una estrucura para definir el tipo de datos que se va a utilizar, en particular PCM, la frecuencia de muestreo, los canales y los bits por muestra que

se obtienen de *Configurar()* y un par de parámetros más derivados de éstos, como la tasa de bytes por segundo y el alineamiento por bloque. Esta estructura vale para todos los buffers que se van a crear. El primero en crearse es el buffer Primario, se configura la estructura que define al buffer tal como se quiere que sea y luego se configura con el formato definido anteriormente. Antes de continuar hay que definir cuanto se pretende almacenar en los buffers, es decir, su tamaño, se intenta que alberguen aproximadamente 2 sg de audio, independientemente de su formato por lo que hay que calcularlo según los datos del usuario, también se tiene la precaución de hacer ese tamaño multiplo de 4, para faciliatar las transferencias en el peor caso, DWORD y por supuesto las menores simpre se adaptarán bien, WORD y BYTE. Con esto, solo falta configurar la estructura que definirá las capacidades del buffer secundario, y crearlo. Este buffer junto con el de captura son los 2 que se manipulan, el primario como ya se comentó solo es el recptor del resultado de la mezcla vía hardware si es posible, si existieran varios secundarios reproduciéndose. Antes de seguir y para evitar picos indeseados en la reproducción se inicializa el buffer secundario con silencio, según el tipo de datos que vaya a almacenar.
Llegados a este punto, hay que saber quien marca las interrupciones para atender a los buffers según se van llenando, por lo tanto hay que discernir que proceso es el predominante; en modo Análisis, con o sin ruido rosa y el modo Filtrado, es la captura, porque no se puede permitir perder datos, aunque en cada caso se hará algo distinto con ellos, bien se calcula su FFT y se muestran los resultados por pantalla o bien, se procesan y se devuleven a la tarjeta de sonido através del buffer secundario. Por el contrario en modo Análisis desde un archivo se leen los datos del disco y se colocan en el buffer secundario, luego en este caso, es la reproducción la que marca cuando le faltan datos que reproducir. Una vez aclarado este punto, se configuran los eventos que pedirán atención a medida que se llena el buffer en el de captura o según avanza el de reproducción. Se segmentan los buffer en 16 subsecciones para que se pidan interrupciones al pasar por ellas y asi realizar las tareas de atención progresivamente según avanza el tiempo. En el caso de grabar a disco en *Ecualizador.exe* también se inicializa la cabecera y se crea el archivo .WAV que albergará la captura de sonido. Antes de terminar se gestionan los indices que controlan el intercambio de datos entre los buffers y la FFT, reseteándose solo cuando sea necesario.

LanzarCaptura() En el caso de que sea la captura la que dirija la petición de los eventos, se arrancan los buffers desde aquí, también hay que saber que es lo que se va a realizar, puesto que Análisis con ruido rosa necesita reproducir el ruido rosa a la vez que está capturando, en cambio, Análisis solo, únicamente debe capturar, en Filtrado simpre full-duplex, luego esta función da las órdenes correspondientes a los buffers necesarios, en caso de grabación se gestiona lo necesario para la preparación del archivo receptor de los datos. Por último se encarga de quedarse en un bucle infinito para atender a los distintos eventos que se sucedan, según se va llenando el buffer de captura. Terminando cuando el usuario lo indique, saliendo correctamente del bucle.

LanzarReproduccion() Esta se llama cuando sea la reproducción la que pida atención através de los eventos, es decir, en el caso de Análisis de archivo. Igual que su homóloga, primero arranca los buffers necesarios y por último se queda en un bucle infinito para ir sacando datos del disco y llevarlos al buffer secundario hasta

que se termine el archivo o el usuario decida que ya ha sido suficiente.

Parar() Como su nombre indica hace exactamente eso, teniéndo encuenta eso sí, el estado de la aplicación para detener únicamente los buffers que estén en marcha, secundario, primario y/o el de captura.

Liberar() Esta función se encarga de desasociar los punteros que apuntan a los buffers y si se va a salir de la aplicación tambien de los interfaces a los objetos DirectSound y DirectSoundCapture. En el caso de *Ecualizador.exe* siempre se libera todo por el problema ya comentado de la imposibilidad de compartir entre dos programas DirectSoundCapture.

ProcesarEntrada() Para la captura cada vez que se pide atención por sobrepasar la marca asociada a un evento se procesa en el bucle y se llama a esta función con la posición del evento que corresponde con las subdivisiones hechas en los buffers y así conocer el lugar de donde hay que quitar los datos recien capturados. Una vez delimitado el bloque que es seguro procesar, se bloquea ese trozo de buffer de captura; ahora hay que saber en que modo se está para realizar lo que el usuario pide, si se está en Análisis, con o sin ruido rosa o si es Filtrado, se llamarán a las funciones que realizarán las tareas necesarias y que se explicarán más adelante. En caso de Análisis con ruido rosa se aprovechan las peticiones de eventos por parte de la captura para llevar la cuenta y bloquear lo suficientemente por delante e ir incluyendo el ruido rosa generado que se va reproduciendo. Para terminar se actualizan los contadores que llevan la cuenta de las posiciones de los buffers. Se desbloquea la parte del buffer bloqueada del buffer de captura y lo mismo con el secundario si se bloqueo para escribir el ruido rosa o el filtrado con el resultado del procesamiento. En caso de grabación se llama a una función proporcionada por DirectX para la gestión de archivos multimedia.

ProcesarReproduccion() Es la homóloga de la anterior solo que los eventos son pedidos por la reproducción que se alimenta de un archivo WAV, se usa otra función proporcionada por DirectX para tal efecto y se gestionan los datos en este caso para el Análisis, es el único modo que puede manipular archivos WAV. Se llaman a las funciones que se encargan de eso y se desbloquea al terminar el buffer secundario que es el único necesario en este caso.

TransferenciaBuffers() Aquí se llega en modo Filtrado, se usa la tarjeta de sonido en modo full-duplex con lo que hay que recoger los datos capturados procesarlos y volverlos a reproducir. Esta función se encarga de todo esto, recibe sendos punteros con el origen y destino dónde deben ir los datos y la cantidad de ellos, simpre en formato BYTE. Lo primero que se hace al entrar en ella es discernir el tipo de transferencia que realizará según el tipo de captura que se haga, será BYTE si 8 bits y mono, WORD si 16 bits y mono o 8 bits y Stereo o DWORD si 16 y stereo. Esto facilita el procesamiento de los datos puesto que en caso de Stereo intercala los canales derecho e izquierdo y así poder procesarlos por separado, también hay que tener siempre presente la diferencia en el tipo de datos si se eligió 8 bits, se tiene 1 BYTE con el cero en el valor 128, mientras que para 16 bits se usan 2 BYTES en modo Little Endian, como corresponde a máquinas Intel, y en codificación de complemento a 2, no olvidar que las señales capturadas son alternas. Por lo tanto hay que recogerlos de esta manera y devolver los en el mismo formato. A parte

de la colocación física de los datos en los buffers, se implementa un solapamiento del 75% en las FFT que también hay que gestionar de forma correcta para evitar los chasquidos en lo posible debido a las discontinuidades en la FFT. Esto es, se calculan 4 FFT's distintas en independientes solapadas un 75%, que una vez se tienen de nuevo los datos en el dominio del tiempo, es decir, se calcula la FFT, se procesan los datos en frecuencia y se calcula de nuevo la FFT^{-1}, se promedian y este resultado es lo que se deposita en el buffer secundario para que se reproduzca y sea audible el resultado.

TransferenciaFFT() Está función es el corazón del Análisis en cualquiera de sus vertientes, a diferencia de la anterior ésta solo gestiona la entrada, así, su cometido es la adaptación de los datos del buffer de captura a la FFT, se recuerda que en 8 bits la cuantificación coloca el 0 en la mitad del rango (128) y el margen dinámico está comprendido entre [0,255], mientras que con 16 bits los datos están en complemento a 2 con lo que no hay nivel de continua, en caso de stereo se intercalan los datos de uno y otro canal, primero va el izquierdo y luego el derecho, aunque en el código se a supuesto que el primero es el derecho, realmente da igual, puesto que no se utliza ninguna función 3D. Para mejorar la eficacia de la FFT se enmarca las capturas de 16 bits en una ventana tipo Hanning y así reducir errores de spanning si las frecuencias no encajan bien en el espaciado de la FFT. En 8 bits no se implementa por el nivel de DC, puesto que mete un armónico de fs/2205 totalmente ficticio, con lo que se optó por no incluirla.

GenerarRR() Solo para Análisis y cuando se active el cuadro correspondiente. Se emplea el algoritmo Voss, en caso de stereo no están en fase ambos canales. Se basa en ir añadiendo ruido blanco en las frecuencias bajas para así simular el ruido rosa, si se entiende como un riodo blanco filtrado por un pasa bajos. Como la generación se realiza en el dominio del tiempo basta con saber la dirección inicial dentro del buffer secundario y cuantos datos se quieren escribir.

SeleccionDispositivo() Para crear la lista de dispositivos que existen en el sistema para gestión de audio. Se utilizan unas funciones especiales de DirectX a tal efecto que reciben como argumento un puntero a otra función *CALLBACK* con esto se consigue que se llame a esa función una vez por dispositivo, permitiendo así al programador ir almacenando los distintos datos de cada dispositivo, tanto para reproducción como para captura. Una vez que se tienen las estructuras que definen a cada dispositivo se porcede a ordenarlas en una lista para que el usuario las pueda ver y seleccionar en el diálogo correspondiente.

Por último las funciones que acompañan la SDK de DirectX para el tratamiento de archivos WAV, al igual que antes no se dirá nada de ellas pues van lo suficientemente comentadas en el código fuente.

Tanto *LanzarCaptura()* como *LanzarReproduccion()* se ejecutan simpre desde una funcion global para poder así satisfacer los requisitos de la funcion que brinda la MFC para lanzar hilos, y así la gestion de los eventos generados por el audio no interfieren en el hilo principal de la aplicación que se podrá dedicar a otras cosas.
En el código fuente, en las funciones que no se usan de esta clase, según el programa en el que se esté, hay un trozo de código en el que se realiza la copia del buffer de captura al

buffer secundario BYTE a BYTE, permitiendo comprobar los efectos de realizar copias incompletas.

2.4.11 Clase CFFT

Esta clase gestiona todo lo relacionado con el cálculo y operaciones adicionales que hay que realizar con la FFT. Se crea esta clase para ocultar las llamadas a las funciones que han sido previamemte compiladas de la version 2.1.3. de RFFT (Real Fast Fourier Transform) de www.fftw.org , creando una librería que se asocia en el lincado. Se usa la RFFT puesto que las señales de las que se parte son reales, lo que captura una tarjeta de sonido. Se procede a describir las distintas funciones miembro.

CrearPlan() Gestiona todas las operaciones previas que hay que realizar, además de asignar a las estructuras de datos las zonas de memoria necesarias para que contengan los valores numéricos que habrá que procesar. Se usa un espaciado fijo o número de puntos fijo, es decir, siempre 2205 puntos, de tal manera que las frecuencias mínimas que se obtendrán dependerán en cada caso de la frecuencia de muestreo (fs/N^o puntos). Las rutinas elegidas que realizarán el cálculo de la FFT requieren que se cree antes de usarlas un "plan" que se genera en este momento, básicamente es una estimación de la máquina, sobre todo de la fiabilidad del reloj [3]. Se crean unas estructuras que luego se utilizarán cada vez que se realice el cálculo de la FFT, por lo tanto, se deben crear 2 planes, uno para la transformada del tiempo a frecuencia y otro para la transformada inversa. Acto seguido se pide memoria al sistema para los arrays que contendrán los datos que hay que suministrar y los que almacenarán los resultados. En caso de *Filtrado.exe* se usan 12 arrays más que en Analisis porque se implementa un solapamiento del 75%, ya comentado, son 4 FFT's, asi que, se necesita más memoria. Los arrays que contendrán el sonido ya procesado, es decir, que su contenido va a ir volcado al buffer secundario de la tarjeta de sonido, se inicializa con los valores necesarios para producir silencio y así evitar chasquidos y otros picos indeseados al comenzar a escribir los primeros segmentos de datos.

CalcularTF() Llama a la funcion de la librería que hace exactamente eso. Se ha añadido la capacidad de gestión de secciones críticas para estos datos puesto que pueden ser manipulandos en cualquier otra función u objeto en ese momento, así se evitan sorpresas desagradables. En este caso se ha definido la transformada para que se conserven los datos de entrada, con lo cual al salir de la función se poseen los dos arrays con el audio en el tiempo y el recien calculado con sus componentes espectrales, se recomienda mirar la documentación adjunta sobre la fftw [3].

CalcularFT() Realiza la operación complementaria a la anterior, realiza la transformada inversa del array que está denominados como In y deja el resultado en Out, aunque en este caso se destruye el contenido de In, por razones de la librería de la fftw [3]. Tambien se ha dotado de protección a los datos con secciones críticas por si alguien está manipulando los datos al mismo tiempo.

DestruirPlan() Como su nombre indica elimina y desasocia todo lo relacionado con los planes, también libera todos los arrays que se han usado para albergar los datos. Solo se viene a esta función antes de salir de la aplicación.

2.4.12 Clase CFiltradoView

Se explica esta clase proporcionada por la MFC debido a que se ha personalizado en gran manera debido a que hay que hacer un uso de la vista mucho más importante que en los otros casos, puesto que sirve de entrada y salida con el usuario. Hay que tener en cuenta que toda la interacción se realiza a través de unos gráficos que simulan potenciómetros reales, así como los botones auxiliares de que sea dotado a cada banda. Por esto se dará una descripción general de cómo se coloca y gestiona su puesta en pantalla e interacción con el ratón.

En principio, se dispone de 4 bmp's diferentes que componen todas las formas posibles que se pueden generar en la vista. Primero, el fondo de los deslizadores con los botones inactivos; segundo, el cursor de los potenciómetros; y por último cómo quedan los botones cuando están pulsados. Por lo tanto, cada uno de ellos es un recurso que irá compilado con el ejecutable. Una vez que se tiene claro con qué se cuenta, sólo resta prepararlo para que lo pueda utilizar el objeto encargado de la vista; se describe un poco más en detalle el proceso.

Constructor/Destructor Se encarga de la signación dinámica de la memoria que albergará los recursos bmp para poder usarlos. Primero hay que cargar el bmp en memoria para luego crear un contexto de dispositivo compatible y poder utlizarlo donde se quiera. Por supuesto el destructor forzará la desasociación de todo lo creado en el constructor.

OnInitialUpdate y PreCreateWindow Entre ambas definen el color de fondo de la vista, en este caso particular negro, y se define tambien el tamño que tendrá la ventana, que únicamente permitirá scroll si el número de bandas pedidas sobrepase la configuración de pantalla. Todo esto se calcula antes de que aparezca la ventana por pantalla. Esto es posible porque se leen del archivo FILTRADO.CFG este tipo de parámetros, con lo que se sabe el tamaño de la vista antes de que se pinte nada en la pantalla.

Un vez comentadas las inicializaciones previas se pasa comentar el funcionamiento, primero se verán las funciones que proporciona la MFC en esta clase vista, algunas siempre están y otras hay que pedirlas. La función *OnDraw()* esta siempre presente, puesto que el flujo del programa viene aquí cuando hay que pintar algo en la vista. Y opcionales que reciben el flujo de ejecución como atención a ciertos sucesos como son el pulsar el botón izquierdo del ratón *OnLButtonDown()*, el soltarlo *OnLButtonUp()* y cuando se mueve el ratón *OnMouseMove()*. Con estos 3 sucesos es posible la gestión de la interfaz con los elementos de la pantalla. Se procede a detallar un poco más cada una de ellas.

OnDraw() Lo primero que se hace es comprobar si se está en disposición de acceder a la pantalla, o si por el contrario hay que mantenerla en negro, esto se decide si se está filtrando o si se está parado. Como es fácil de suponer todas las bandas tienen los mismo elementos y se diferencian en pocas cosas, por lo que, se pintan todas ellas en un bucle que lleva la cuenta de cúal se está pintando y obra en consecuencia para colocarle a cada una de ellas sus elementos diferenciadores. El fondo de los deslizadores simpre se repinta igual, no así los cursores de los potenciómetros que pueden permanecer estáticos, con lo cual se obtiene su posición de las estructuras

encargadas de almacenar estos valores y el otro modo que el usuario los esté moviendo con lo cual hay que proceder a repintarlos con las coordenadas correctas según el deseo del usario, es decir, seguir el arrastre del ratón. Se ha añadido un potenciómentro más que se encarga de controlar el volumen general para evitar posibles saturaciones debidas a la naturaleza digital del procesamiento. Una vez que se ha puesto lo básico en la pantalla, se procede a diferenciar cada banda, así, se serigrafía la parte inferior de cada banda con el valor de la frecuencia central, los primeros que controlan el volúmen general se les identifica como corresponde, se le corrige la serigrafía inicial del fondo que identificaba las funciones de los botones que por estar en ellos se les asigna una nueva funcionalidad, Reset y Direct. Se sigue con la puesta en el final del recorrido de los cursores los margenes de ganancia y atenuación. También se ha dotado de una ventanita donde queda reflejado el estado actual de ganancia de cada potenciómetro. Por último se comprueba el estado de los botones y en caso de estar activos se coloca el bmp correspondiente para que de el efecto de pulsado según corresponda.

OnLButtonDown() Esta función realiza una tarea de rastreo a lo largo de toda la vista hasta que detecta una pulsación del botón izquierdo del ratón dentro de los límites de algún elemento susceptible de interactuar con el usuario. Primero se realiza un rastreo a lo largo de todos los cursores de los potenciómentros. En el momento que se detecta que se ha pinchado dentro de la región que forma uno de los cursores se realiza la captura para que el cursor siga al ratón en sus movimientos verticales. En esta función solo se determina la posición del ratón y el offset respecto a la esquina superior izquierda del cursor, para minimizar los errores de redondeo con números enteros. El siguiente rastreo se realiza para los botones, es más sencillo su procesamiento puesto que estos no se pueden arrastrar como los cursores, así, al detectar su pulsación se actualizan las variables correspondientes en cuanto al estado y los arrays que están incluidos en las operaciones de filtrado. Antes de terminar hay que decirle al armazón que repinte la zona que ha cambiado, para que ponga el botón como corresponde. Según el tipo de botón pulsado se implementa la lógica asociada con él, Mute que silencia la banda en que se ha activado éste, o Solo que silencia todas las bandas excepto ella misma, tampoco hay que olvidar que esos dos botones en el deslizador dedicado al volumen tienen otra función y que se procesan por separado, implementando un bypass en caso de Direct o el de Reset, llevando al ecualizador a su estado inicial. Por supuesto si existen dos filas de potenciómentros hay que rastrear la segunda también.

OnMouseMove() Veamos como se gestiona el arrastre, que es en el único momento en que realiza alguna tarea en esta función. Se controla que se está arrastrando, es decir, que se ha capturado el ratón, el usuario lo nota porque ha cambiado el puntero. Primero se invalida la ubicación actual del cursor, se recuerda que se conoce la ubicación del ratón y el offset con la esquina superior izquierda del cursor del potenciómetro, para que se repinte. Se calcula la nueva posición hasta la que hay que llevar el cursor, esto se puede hacer porque uno de los parámetros con que llama el armazón de la aplicación a esta función es con la posición actual del ratón, que irá cambiando en respectivas llamadas. Se comprueban previamente los límites del deslizador, y una vez que se tiene la nueva posición se invalida la zona de la pantalla para que se repinte correctamente. Así se consigue el efecto del arrastre.

OnLButtonUp() Esta función recibe el control cuando se levanta el dedo del botón izquierdo del ratón, por lo tanto es aquí donde se termina de gestionar los arrastres y algunos botones. Primero se verá como gestionar el final del arrastre. Se comprueba si se estaba con el ratón capturado y si es así se procede a terminar, empezando por devolver al puntero del ratón su apariencia inicial. Seguidamente se distingue entre la fila superior de potenciómentros y la inferior si existe, este parámetro se estableció al comenzar el arrastre, como se ajustó el desplazamiento en la funcion OnMouseMove() se hace un cambio de escala entre la medida en pixels y el intervalo [0, margen de ganancia] con escala logarítmica en ambos sentidos, es decir, amplificación o atenuación, por lo tanto se trata de distinta manera si los cursores están en la parte superior, amplificación, o si están en la parte inferior, atenuación, también se mantiene la gestión en caso de que el usuario deseé potenciómentros stereo y se gestionan los dos canales con una única fila de potenciómentros en la pantalla. Se escriben los valores en los arrays que servirán para el calculo del filtro y está. Hay que recordar el tratamiento especial que reciben los primeros potenciómetros que aplicándoseles la misma conversión se obtiene la amplificación que hay que aplicarle al total de la señal. Lo último que se procesa son los botones de Reset que están en los potenciómetros del volumen, que se comportan como pulsadores y no como botones, por lo que se levantan al levantar el dedo del botón izquierdo del ratón y desencadena la tarea asociada al botón.

2.4.13 Clase CAnalisisView

Se hace mención de esta clase sólamente por que centraliza la atención a la entrada del teclado cuando se tiene el análisis en la pantalla, se usan dos métodos que ofrece el armazón, para las teclas sueltas, como F1, ESC o las flechas de cursor se utiliza la función miembro *OnKeyUp()* que captura cualquier tecla que haya sido pulsada, recibiendo el control del programa al levantar el dedo de ella, por lo tanto, primero se comprueba que la tecla en cuestión es una de las que realizarán una función y luego se implementa la funcionalidad, amplificar la señal, desplazar en una unidad el espectro en la pantalla, etc.

- ESCAPE. Se activa el evento que arranca la secuencia de parada
- Flecha derecha. Comprueba los límites del scroll, y luego incrementa la variable que lleva la cuenta de las columnas que se están representando en cada momento.
- Flecha izquierda. Se comprueban los límites de scroll y decrementa el contador correspondiente.
- Flecha arriba. Incrementa la varible que sirve de cte para todas las bandas.
- Flecha abajo. Decrementa la varible que sirve de cte para todas las bandas.
- F1. Simplemente gestiona una bandera boleana que hace aparecer o desaparecer de pantalla la ayuda.

Por el contrario las que implican la pulsación de dos teclas simultáneamente, como los scrolls rápidos se usan pulsaciones virtuales, es decir, el armazón traduce esa combinación de teclas en un mensaje normal de Windows que se puede procesar casi en cualquier lado, en particular se procesa en la clase de la vista. Así existen dos funciones miembro para cada mensaje.

OnRapidoIzq() Realiza scroll pero de pantalla completa, es decir, se despalza tantas columas a la izquierda como se muestran en la pantalla, pero comprobando previamente que se puede hacer, en caso de ser imposible se desplaza las necesarias hasta que llega a la frecuencia más baja que se muestra por pantalla. Para el acceso a los datos, que también se usan en la representación por pantalla, se sincronizan convenientemente para que se modifiquen cuando no se está refrescando.

OnRapidoDer() Realiza el scroll pero hacia frecuencias altas, que el efecto en la pantalla es hacia la derecha, la implementación es exactamente igual que el anterior solo que ahora hay que comprobar que no se sobrepasen los limites superiores.

2.4.14 Clase CFiltro

Esta clase se crea para encapsular todo lo relacionado con el tratamiento de los arrays que proporciona la FFT en el dominio de la frecuencia, los trata y se los devuelve para volver al dominio del tiempo y poder así escuchar el resultado através de la tarjeta de sonido. Como en los otros apartados se procede a explicar las funciones miembro que dan una idea de las capacidades de la clase.

Configurar() Del archivo FILTRADO.CFG se obtienen los 2 arrays calculados en *Equalizador.exe*, con las frecuencias y la agrupación sobre los puntos de la FFT. Por lo tanto se centra esta funcion en la creación y asignación de todas las estructuras dinámicas que contrendrán los datos intermedio necesarios para la correcta manipulación de los mismos. Se inicializan desde los arrays necesarios para interactuar con la FFT, a las estructuras que controlan la posición y estado de todos los elementos que aparecerán por pantalla, así como los valores numéricos que contendrán por defecto.

Reiniciar() En caso de pulsar los botones de reset en los potenciómetros que controlan el volumen general, se produce una reinicialización de todos lo valores del ecualizador, por lo tanto, recorre todos los arrays para colocarles los valores iniciales. El resultado gráfico es que todo vuelve a la situación inicial.

FiltrarR() Se encarga de implementar el filtrado de la señal en frecuencia, su función principal es la de aplicar los distintos coeficientes a cada componente o conjunto de componentes (bandas) de frecuencia de la señal. Antes de realizar cálculo alguno se hace una comprobación por si está pulsado el botón que permite hacer un bypass al ecualizador y permitir escuchar la señal sin filtrado, o si tienen efecto las correspondientes constantes que afectan a cada banda. Luego según el estado de éste se procesa teniendo en cuenta los coeficientes de cada banda o todos con la unidad. Por supuesto esta función filtra el canal derecho, R en inglés, usa todas las estructuras de datos destinadas a él. Por supuesto se ha dotado de zonas críticas el manejo de los arrays de datos que luego manipula la FFT, para evitar conflictos entre ellos. Viendo el código detenidamente las llamdas son sucesivas luego no debería haber problemas, pero ...

FiltrarL() Hace exactmanete lo mismo que la anterior pero cogiendo los datos de los arrays destinados al canal izquierdo. También en este caso se protege la manipulación de los datos con zonas críticas puesto que la FFT está también accediendo a los arrays.

Liberar() Se encarga de desasociar y liberar todas las estructuras definidas en *Configurar()* que como eran todas dinámicas antes de salir de la aplicación hay que dejarlo todo como estaba para evitar posibles pérdidas de memoria y errores en la salida al sistema operativo.

2.4.15 Clase Genrerar_RR

Con esta clase se pretende envolver todas las operaciones necesarias para generar el Ruido Rosa y dejar una interfaz para su uso desde los objetos que sea necesarios. Utiliza el algoritmo de Voss y esta implementado por Thomas Hudson en 1998.

En el constructor se define un parámetro que personaliza los rangos en los que se quiere obtener el ruido rosa, puesto que son distintos según se esté procesando el audio a 8 o 16 bits, éste debe ser distinto, por esta razón se crean 2 objetos distintos para la generación según el tipo de datos.
En cuanto a la generación propiamente dicha se basa en la implementación de ruido blanco al que se le van añadiendo progresivamente más ruido blanco pero en las frecuencias bajas, puesto que el ruido rosa se puede obtener como ruido blanco filtrado através de un pasa bajos.

2.4.16 Estructura Opc_Muestreo

Se crea esta estructura para recoger los datos del diálogo correspondiente de Ecualizador, y así conseguir cieta simetría entre los distintos programas. De esta manera, la estructura es el receptáculo en el que se colocarán los datos que se leen de OPCIONES.CFG, que ha escrito previamente el lanzador (*Ecualizador.exe*). En ella se almacena la frecuencia de muestreo, los bits que se usarán por captura y los canales que se capturan y/o reproducen (Mono o Stereo).

2.4.17 Opc_Graficos

Esta estructura recoge los datos relacionados con la puesta por pantalla del analizador gráfico, y que sirva de homóloga al diálogo que emplea Ecualizador para la recogida de configuración por parte del usuario. Por lo tanto al teminar de leer los datos de OPCIONES.CFG se almacenan aquí para luego servir en la configuración de la clase *CDirectDraw*. Almacena los datos de; número de cortes por pantalla, si la escala de las abcisas será lineal o logarítmica, el número de escalones que habrá en el eje de ordenadas, las cotas de los ejes de ordenadas y abcisas, y por último si se analiza un archivo, y el nombre del mismo o si se quiere generar ruido rosa.

Capítulo 3

Código del programa

Se procede a detallar los archivos principales, al menos, los que han sido creados para este proyecto y los que aún habiendo sido generados por el asistente han sufrido tantas modificaciones que merecen aparecer aquí. Forman cada uno de los 3 proyectos que generan los respectivos ejecutables; *Analisis.exe*, *Ecualizador.exe* y *Filtrado.exe*. Los ficheros que no han variado, es decir, los creó el asistente y han sido poco o nada modificados se han omitido, si se desean ver, están junto con éstos en la version electrónica que acompaña al proyecto. En esta sección se pretende reflejar o mostrar lo que se explicó en la Guía de programación, para el que lo deseé pueda ver la implementación real, con todas las limitaciones y parafernalia que ésta conlleva.

3.1 Análisis

3.1.1 AnalisisView

AnalisisView.h

```
// AnalisisView.h : interface of the CAnalisisView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_ANALISISVIEW_H__7FA5BF4F_4AE6_11D5_871D_8995A9A7F779__INCLUDED_)
#define AFX_ANALISISVIEW_H__7FA5BF4F_4AE6_11D5_871D_8995A9A7F779__INCLUDED_

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

class CAnalisisView : public CView
{
protected: // create from serialization only
        CAnalisisView();
        DECLARE_DYNCREATE(CAnalisisView)

// Attributes
public:
```

```
        CAnalisisDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAnalisisView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CAnalisisView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CAnalisisView)
        afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
        afx_msg void OnRapidoIzq();
        afx_msg void OnRapidoDer();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in AnalisisView.cpp
inline CAnalisisDoc* CAnalisisView::GetDocument()
   { return (CAnalisisDoc*)m_pDocument; }
#endif

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

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

#endif // !defined(AFX_ANALISISVIEW_H__7FA5BF4F_4AE6_11D5_871D_8995A9A7F779__INCLUDED_)
```

AnalisisView.cpp

```
// AnalisisView.cpp : implementation of the CAnalisisView class
//

#include "stdafx.h"
#include "Analisis.h"

#include "AnalisisDoc.h"
#include "AnalisisView.h"
#include "Global.h"

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

extern CDirectDraw Dd;


/////////////////////////////////////////////////////////////////////////////
// CAnalisisView

IMPLEMENT_DYNCREATE(CAnalisisView, CView)

BEGIN_MESSAGE_MAP(CAnalisisView, CView)
        //{{AFX_MSG_MAP(CAnalisisView)
        ON_WM_KEYUP()
        ON_COMMAND(ID_RAPIDO_IZQ, OnRapidoIzq)
        ON_COMMAND(ID_RAPIDO_DER, OnRapidoDer)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAnalisisView construction/destruction

CAnalisisView::CAnalisisView()
{
        // TODO: add construction code here
        m_reentrada=false;
}

CAnalisisView::~CAnalisisView()
{
}

BOOL CAnalisisView::PreCreateWindow(CREATESTRUCT& cs)
```

```
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        cs.lpszClass=AfxRegisterWndClass(CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW,
                NULL,::CreateSolidBrush(RGB(0,0,0)));   // Fondo Negro

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CAnalisisView drawing

void CAnalisisView::OnDraw(CDC* pDC)
{
        CRect rc;
        int x,y;

        CAnalisisDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // Para confirmar que el usuario quiere lanzar el analisis, muestra un mensaje
        // por la ventana de la vista

        GetClientRect(&rc);

        pDC->SetTextColor(RGB(0,255,255));
        pDC->SetBkColor(RGB(0,0,0));
        y=rc.Height();
        x=rc.Width();
        if (WaitForSingleObject(g_PararTimer, 0) == WAIT_OBJECT_0)      {
                pDC->TextOut((x/2)-90, (y/2)-30, "Espere unos instantes");
        }
        else    {
                pDC->TextOut((x/2)-100, (y/2)-40, "Pulsa la barra espaciadora");
                pDC->TextOut((x/2)-85, (y/2)+30, "Para Cancelar Alt+F4");
        }
}

/////////////////////////////////////////////////////////////////////////////
// CAnalisisView diagnostics

#ifdef _DEBUG
void CAnalisisView::AssertValid() const
{
        CView::AssertValid();
}
```

```
void CAnalisisView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}

CAnalisisDoc* CAnalisisView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CAnalisisDoc)));
        return (CAnalisisDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CAnalisisView message handlers

void CAnalisisView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        // Son las teclas que funcionan en el Analisis grafico y no son virtuales
        // como el resto, es decir CTRL+-> y la otra

        if (g_Analisis) {
                switch (nChar)  {
                case VK_ESCAPE:
                        // Para terminar y restaurar los valores del principio
                        TRACE("SE PULSO ESCAPE\n");
                        g_PararTimer.SetEvent();
                        break;
                case VK_RIGHT:
                        // Hara el scroll del visualizador cuando se pueda hacer
                        if (!m_reentrada)       {
                                m_reentrada=true;
                                if (Dd.m_Virt_X < Dd.m_NCortes - Dd.m_CortesPorPantalla) {
                                        WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                        g_RefrescoPantalla.SetEvent();
                                        // Avanza en 1 hacia Frecuencias Altas
                                        Dd.m_Virt_X++;
                                        Dd.m_CambiarFondo=0;    // Para que actualize todo
                                }
                                m_reentrada=false;
                        }
                        break;
                case VK_LEFT:
                        // Implementa el Scroll del visualizador cuando se pueda
                        if (!m_reentrada)       {
                                m_reentrada=true;
                                if (Dd.m_Virt_X > 0)    {
                                        WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                        g_RefrescoPantalla.SetEvent();
```

```
                                        Dd.m_Virt_X--;
                                        Dd.m_CambiarFondo=0;
                                }
                                m_reentrada=false;
                        }
                        break;
                case VK_UP:
                        // Se encarga de escalar en amplitud
                        Dd.m_Cte-=1;
                        break;
                case VK_DOWN:
                        Dd.m_Cte+=1;
                        break;
                case VK_F1:
                        // Muestra la ayuda por pantalla
                        if (Dd.m_Ayuda)
                                Dd.m_Ayuda=false;
                        else
                                Dd.m_Ayuda=true;
                        Dd.m_CambiarFondo=0;
                }
        }
        CView::OnKeyUp(nChar, nRepCnt, nFlags);
}

void CAnalisisView::OnRapidoIzq()
{
        // Avance rápido para el SCROLL si hay muchos cortes, esto ya son teclas
        // virtuales, generan un mensaje directamente manipulable por el armazón

        if (!m_reentrada)       {
                m_reentrada=true;
                if (Dd.m_Virt_X > 0)    {
                        // Esta separado del lado Izquierdo
                        if (Dd.m_Virt_X-Dd.m_CortesPorPantalla > 0)     {
                                // Se puede avanzar una pantalla entera
                                // Espera a que no se esté refrescando la pantalla
                                WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                g_RefrescoPantalla.SetEvent();
                                Dd.m_Virt_X-=Dd.m_CortesPorPantalla; // Avanza a f bajas
                        }
                        else    {
                                // No se puede una entera
                                WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                g_RefrescoPantalla.SetEvent();
                                Dd.m_Virt_X=0;          // Al principio del todo
                        }
                }
```

```
                Dd.m_CambiarFondo=0;
                m_reentrada=false;
        }
        return;
}

void CAnalisisView::OnRapidoDer()
{
        // Avance rápido para SCROLL hacia las frecuencias altas, igual que antes

        if (!m_reentrada)       {
                m_reentrada=true;
                if (Dd.m_Virt_X < Dd.m_NCortes - Dd.m_CortesPorPantalla)        {
                        if (Dd.m_Virt_X+Dd.m_CortesPorPantalla <= Dd.m_NCortes -
                                        Dd.m_CortesPorPantalla) {
                                // Se puede avanzar una pantalla entera
                                WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                g_RefrescoPantalla.SetEvent();
                                // Avanza a Frecuencias Altas
                                Dd.m_Virt_X+=Dd.m_CortesPorPantalla;
                        }
                        else    {
                                WaitForSingleObject(g_RefrescoPantalla, INFINITE);
                                g_RefrescoPantalla.SetEvent();
                                // Al final
                                Dd.m_Virt_X=(Dd.m_NCortes - Dd.m_CortesPorPantalla);
                        }
                }
                Dd.m_CambiarFondo=0;    // Para que actualize todo
                m_reentrada=false;
        }
        return;
}
```

3.1.2 Dialog_Error

Dialog_Error.h

```
#if !defined(AFX_DIALOG_ERROR_H__B05EC781_4B86_11D5_871D_AA51EC8B3E79__INCLUDED_)
#define AFX_DIALOG_ERROR_H__B05EC781_4B86_11D5_871D_AA51EC8B3E79__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Error.h : header file
//

/////////////////////////////////////////////////////////////////////////////
```

```
// CDialog_Error dialog

class CDialog_Error : public CDialog
{
// Construction
public:
        CDialog_Error(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialog_Error)
        enum { IDD = IDD_DIALOG_ERROR };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Error)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_Error)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_ERROR_H__B05EC781_4B86_11D5_871D_AA51EC8B3E79__INCLUDED_)
```

Dialog_Error.cpp

```
// Dialog_Error.cpp : implementation file
//

#include "stdafx.h"
#include "Analisis.h"
#include "Dialog_Error.h"
#include "Global.h"
```

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

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error dialog


CDialog_Error::CDialog_Error(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Error::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Error)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CDialog_Error::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Error)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Error, CDialog)
	//{{AFX_MSG_MAP(CDialog_Error)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error message handlers

BOOL CDialog_Error::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Inicializaciones adicionales

	CString str;

	switch (g_Error)	{
	case 30:
	case 31:
```

```
        case 32:
        case 33:
        case 34:
        case 35:
        case 36:
        case 37:
        case 38:
        case 39:
        case 40:
        case 41:
                str="Fallo en DirectSound";
                break;
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
                str="Fallo en DirectDraw";
                break;
        case 90:
                str="No se puede leer la configuración. Debes lanzarlo desde
                     Ecuallizador.exe";
                break;
        case 100:
                str="Tienes unas DirectX antiguas, debes actualizar a versión
                     6.0 o superior";
                break;
        default:
                str="Esto no debería salir nunca";      // No mensajes por defecto
                break;
        }

        SetDlgItemText(IDC_STATIC_ERROR, str);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

3.1.3 DirectDraw

DirectDraw.h

```
// DirectDraw.h: interface for the CDirectDraw class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DIRECTDRAW_H__A4C53E81_F763_11D4_B11D_E8F76C0C897E__INCLUDED_)
```

```
#define AFX_DIRECTDRAW_H__A4C53E81_F763_11D4_B11D_E8F76C0C897E__INCLUDED_

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

#include "Global.h"


class CDirectDraw
{
public:
        CDirectDraw();
        virtual ~CDirectDraw();
        int Restaurar(void);
        Crear();
        Configurar();
        IniciarSurfaces();
        Refrescar();
        Destruir(bool Todo);

protected:
        DDLoadBitmap(IDirectDraw4 * pdd, LPCSTR szBitmap, int dx, int dy,
                LPDIRECTDRAWSURFACE4 *Salida);
        DDReLoadBitmap(IDirectDrawSurface4 * pdds, LPCSTR szBitmap);
        DDCopyBitmap(IDirectDrawSurface4 * pdds, HBITMAP hbm, int x, int y, int dx,
                int dy);

        HRESULT RestaurarMemVid();
        int PintarPantalla();
        void PreparararDatos();

public:
        HINSTANCE                       m_hinstance;
        int m_Canales, m_Fs;
        int m_NCortes;
        int m_NEscalones;
        int m_CambiarFondo;             // Para actualizar el fondo
        int m_Virt_X;                   // Virt_Cortes
        int m_CortesPorPantalla;        // Si hay que implementar Scroll
        double m_Cte;                   // Escalado en Amplitud
        UINT m_Timer;                   // Identificador del temporizador de refresco
        bool m_Ayuda;                   // Ayuda en pantalla

protected:
        LPDIRECTDRAW4 lpDD4;
        LPDIRECTDRAWSURFACE4 lpDDSPrimary;
        LPDIRECTDRAWSURFACE4 lpDDSBack;
```

```
        LPDIRECTDRAWSURFACE4 lpDDSbmp;
        DDSURFACEDESC2 ddsd;
        DDBLTFX   ddbltfx;
        HWND m_HwndTop;                 // Puntero a la ventana marco principal
        double *m_Escala_double;
        int *m_Escala;
        DWORD *m_Indice;
        int *m_Frecuencias, *m_Agrupacion;
        int *m_PuntosFFT;               // Las f a las que se obtienen las componentes
        int m_CortesMax;
        int m_PuntofftInicio;

        DWORD OffsetY, OffsetY_L, OffsetX, totx, toty, Paso_y, Paso_x;  // Coordenadas

        double *m_OutR, *m_OutL;                // Datos para pintar las barras
        double *m_PersistenciaR, *m_PersistenciaL;  // Datos para la persistencia
        int m_Envejecer;                        // Cada cuanto se envejecen los datos

};

#endif // !defined(AFX_DIRECTDRAW_H__A4C53E81_F763_11D4_B11D_E8F76C0C897E__INCLUDED_)
```

DirectDraw.cpp

```
// DirectDraw.cpp: implementation of the CDirectDraw class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DirectDraw.h"
#include "Global.h"
#include "Estructuras.h"

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

static char strbmp[]="IDC_TODO";        // Para cargar el recurso

CEvent g_RefrescoPantalla;

extern CFft Fft;
extern CDirectS Ds;
extern Opc_Muestreo Muestreo;
extern Opc_Graficos Graficos;
```

```
#define PASO_DEL_TIEMPO 3               // Se refresca cada 30msg=>300msg envejece
#define CAIDA                   3               // dB que cae cada vez que envejece
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CDirectDraw::CDirectDraw()
{
        // Inicializacion para los punteros
        lpDD4=NULL;
        lpDDSPrimary=NULL;
        lpDDSBack=NULL;
        lpDDSbmp=NULL;
        m_CambiarFondo=0;
        m_Cte=10;
        m_Ayuda=false;

}

CDirectDraw::~CDirectDraw()
{
}


CDirectDraw::Crear()
{
        LPDIRECTDRAW  lpDD;

        if(DD_OK != DirectDrawCreate(NULL, &lpDD, NULL))
                return -1;


        if (DD_OK != lpDD->QueryInterface(IID_IDirectDraw4, (LPVOID *)&lpDD4))
                return -1;


        // Obtener un puntero a la ventana raiz
        m_HwndTop=AfxGetMainWnd()->GetSafeHwnd();

        TRACE("TODO FUE BIEN EN LA CREACION Dd\n");

        return 0;
}

int CDirectDraw::IniciarSurfaces()
{
        HRESULT hr;
```

```
DDSCAPS2      ddscaps;

if (DD_OK != lpDD4->SetCooperativeLevel(m_HwndTop, DDSCL_FULLSCREEN
                |DDSCL_EXCLUSIVE | DDSCL_ALLOWREBOOT ))
        return -1;
// Cuando se esté depurando y probando es conveniente permitir CRTL+ALT+SUP
// añadiendo la bandera | DDSCL_ALLOWREBOOT

// Creacion de una Surface Principal con otra Back (para flipper)
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);
// Le dice a DirectDraw que miembros son válidos
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
// Coloca cada miembro al valor que le corresponda
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE
                                        | DDSCAPS_FLIP
                                        | DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;

hr=lpDD4->CreateSurface(&ddsd, &lpDDSPrimary, NULL);

TRACE("lpDDSPrimary=%x\n",lpDDSPrimary);

if (DD_OK != hr)        {
        TRACE("ERROR: Creacion Primary Surface\n");
        return -1;
}
// Obtencion de un puntero a la pantalla que no se muestra, lpDDSBack

ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
hr=lpDDSPrimary->GetAttachedSurface(&ddscaps, &lpDDSBack);
if (DD_OK != hr)        {
        TRACE("ERROR: Obtencion de un puntero a Back Surface\n");
        return -1;
}
else    {
        TRACE("lpDDSBack=%x\n",lpDDSBack);
}

// Creacion de otra superficie para albergar los bmp que se emplearan
// para componer el visualizador
DDLoadBitmap(lpDD4, strbmp, 0, 0, &lpDDSbmp);
if (lpDDSbmp == NULL)   {
        TRACE("ERROR: Creacion de Surface con los bmp\n");
        return -1;
}
TRACE("lpDDSbmp=%x\n",lpDDSbmp);
```

```
        // Reset de la variable que controla el envejeciemiento de los datos
        m_Envejecer=0;

        return 0;
}

int CDirectDraw::Restaurar()
{
        // Deja todo como estaba
        // Repinta la ventana de la aplicacion

        if (DD_OK != lpDD4->SetCooperativeLevel(m_HwndTop, DDSCL_NORMAL))       {
                TRACE("ERROR: Restauracion de la prioridad\n");
        }

        if (DD_OK != lpDD4->FlipToGDISurface())         {
                TRACE("ERROR: Al volver el control a la GDI de la MFC\n");
        }

        m_CambiarFondo=0;    // La siguiente vez que se active hay que pintarlo
                             // todo
        return 0;
}


int CDirectDraw::Refrescar()
{
        // Esta funcion se encarga del refresco de la pantalla.
        // Aqui se gestiona el cambio entre las 2 surfaces
        HRESULT hr;
        int result;

        PrepararDatos();                // Realiza las transformaciones necesarias
        result=PintarPantalla();        // Compone la pantalla en el plano de atrás

        if (result != 0)                {
                TRACE("ERROR: En PintarPantalla\n");
                return -1;
        }
    while (true)        {
                hr=lpDD4->TestCooperativeLevel();
                // Esperar
                if (DDERR_NOEXCLUSIVEMODE == hr)
                        // Se ha perdido el control exclusivo de la VGA => Esperar
                        RestaurarMemVid();              // Restaura la memoria de video
                else if (hr == DDERR_INVALIDOBJECT || hr == DDERR_WRONGMODE )   {
                        g_Error=12;
                        return -1;
```

```
                }
                else if (DD_OK == hr)   {
                        // Cambiar plano que se visualiza por el de atrás
                        hr = lpDDSPrimary->Flip(NULL, DDFLIP_WAIT);
                        if (hr == DD_OK)
                                break;          // Lo hizo bien. Sale
                }
        }

        return 0;
}

int CDirectDraw::PintarPantalla()
{

        // Coloca en la pantalla los datos nuevos que se mostrarán en el siguiente
        // frame

        HRESULT hr;
        HDC lpDC;
        RECT rc;
        int i,j;
        static BOOL fase=false; // Para saber que surface se esta mostrando

        DWORD x,y;              // Apuntan las posiciones donde colocar los indicadores
        char cadena[5], *string;
        HFONT lpFont;
        bool oscilar;


        // Gestion del fondo y preparacion de las coordenadas para comenzar a
        // mostrar en pantalla según la configuracion de la pantalla

        // Realizacion del fondo de la pantalla, releno con color e indicadores
        // que estarán siempre, informacion adicional...

        if (m_CambiarFondo < 2)         {
                ddbltfx.dwSize = sizeof(ddbltfx);
            ddbltfx.dwFillColor = 0;

                while (true)    {
                        // Relleno del fondo de la pantalla de negro
                        hr=lpDDSBack->Blt(NULL, NULL, NULL, DDBLT_COLORFILL | DDBLT_WAIT,
                                        &ddbltfx);
                        if (hr == DD_OK)
                                break;
                        else if (hr == DDERR_SURFACELOST)       {
                                hr = RestaurarMemVid();
```

```
                        if (hr == DD_OK)
                                continue;
        }
            if (hr == DDERR_WASSTILLDRAWING)
                        continue;
                if (hr == DDERR_INVALIDPARAMS || hr == DDERR_GENERIC
                                || hr == DDERR_INVALIDOBJECT )  {
                        g_Error=13;
                        return -1;
                }
            }

            // Colocacion de marcas auxiliares para la visualizacion
            x=OffsetX-25;
            y=OffsetY-15;                   // A la izquierda de las barras
            string=cadena;

            lpDDSBack->GetDC(&lpDC);

            lpFont=CreateFont(14,
                     0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE,
                     ANSI_CHARSET,
                     OUT_DEFAULT_PRECIS,
                     CLIP_DEFAULT_PRECIS,
                     DEFAULT_QUALITY,
                     VARIABLE_PITCH,
                     "Arial");
            if (lpFont == NULL)             {
                    TRACE("ERROR: Creacion de la nueva fuente\n");
                    g_Error=15;
                    return -1;
            }

            // Marcas verticales de amplitud
            SelectObject(lpDC, lpFont);
            SetBkColor(lpDC, RGB(00, 00, 00));

            for (i=0; i< m_NEscalones; i=i+(Paso_y==12 ? 1 : 2))    {
                    string=itoa(m_Escala[i], string, 10);
                    SetTextColor(lpDC, RGB(0, 200, 0));        // Color
                    if (*(m_Escala+i) == 0)
                            TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)), string, 1);
                    else if (*(m_Escala+i) <= -10)
                            TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)), string, 3);
                    else
                            TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)), string, 2);
            }
            // Colocacion de dB
```

```
SetTextColor(lpDC, RGB(0, 255, 0));                    // Color
TextOut(lpDC, x+3, y-(i*(Paso_y==12 ? 12 : 6))-10, "dB", 2);

// Marcas horizontales para los cortes de frecuencia
x=OffsetX;
y=OffsetY;                 // Debajo de las barras
oscilar=false;

for (i=m_Virt_X; i<(m_Virt_X+m_CortesPorPantalla<m_NCortes ?
                m_Virt_X+m_CortesPorPantalla :  m_NCortes); i++) {
        string=itoa(m_Frecuencias[i], string, 10);
        SetTextColor(lpDC, RGB(0, 200, 0));              // Color
        rc.left=x;
        rc.top=y;
        rc.right=x+28;
        rc.bottom=y+12;
        if (m_Frecuencias[i]<=99)
                DrawText(lpDC, string, 2, &rc, DT_CENTER);
        else if (m_Frecuencias[i]<=999)
                DrawText(lpDC, string, 3, &rc, DT_CENTER);
        else if (m_Frecuencias[i]<=9999)        {
                DrawText(lpDC, string, 4, &rc, DT_CENTER);
                if (!oscilar)   {
                        // Alterna para que entren en horizontal
                        y+=12;
                        oscilar=true;
                }
                else    {
                        y-=12;
                        oscilar=false;
                }
        }
        else            {
                DrawText(lpDC, string, 5, &rc, DT_CENTER);
                if (!oscilar)   {
                        // Alterna para que entren en horizontal
                        y+=12;
                        oscilar=true;
                }
                else    {
                        y-=12;
                        oscilar=false;
                }
        }
        x+=28;
}
// Colocacion de leyenda Hz
SetTextColor(lpDC, RGB(0, 255, 0));                    // Color
```

```
TextOut(lpDC, x-15, y+(oscilar ? 15 : 15+12), "Hz", 2);

// En caso de Stereo para colocar las marcas al otro canal
if (m_Canales == 2)             {
        // Marcas verticales de amplitud
        x=OffsetX-25;
        y=OffsetY_L-15;
        SelectObject(lpDC, lpFont);
        SetBkColor(lpDC, RGB(00, 00, 00));
        for (i=0; i< m_NEscalones; i=i+(Paso_y==12 ? 1 : 2))    {
                string=itoa(m_Escala[i], string, 10);
                SetTextColor(lpDC, RGB(0, 200, 0));    // Color
                if (*(m_Escala+i) == 0)
                        TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)),
                                string, 1);
                else if (*(m_Escala+i) <= -10)
                        TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)),
                                string, 3);
                else
                        TextOut(lpDC, x, y-(i*(Paso_y==12 ? 12 : 6)),
                                string, 2);
        }
        // Colocacion de dB
        SetTextColor(lpDC, RGB(0, 255, 0));                     // Color
        TextOut(lpDC, x+3, y-(i*(Paso_y==12 ? 12 : 6))-10, "dB", 2);

        // Marcas horizontales, para el eje de frecuencias
        x=OffsetX;
        y=OffsetY_L;                    // Debajo de las barras
        oscilar=false;
        for (i=m_Virt_X; i<(m_Virt_X+m_CortesPorPantalla<m_NCortes ?
                        m_Virt_X+m_CortesPorPantalla :  m_NCortes); i++) {
                string=itoa(m_Frecuencias[i], string, 10);
                SetTextColor(lpDC, RGB(0, 200, 0));          // Color
                rc.left=x;
                rc.top=y;
                rc.right=x+28;
                rc.bottom=y+12;
                if (m_Frecuencias[i]<=99)
                        DrawText(lpDC, string, 2, &rc, DT_CENTER);
                else if (m_Frecuencias[i]<=999)
                        DrawText(lpDC, string, 3, &rc, DT_CENTER);
                else if (m_Frecuencias[i]<=9999)        {
                        DrawText(lpDC, string, 4, &rc, DT_CENTER);
                        if (!oscilar)   {
                                // Alterna para que entren en horizontal
                                y+=12;
                                oscilar=true;
```

```
                        }
                        else    {
                                y-=12;
                                oscilar=false;
                        }
                }
                else            {
                        DrawText(lpDC, string, 5, &rc, DT_CENTER);
                        if (!oscilar)   {
                                // Alterna para que entren en horizontal
                                y+=12;
                        oscilar=true;
                        }
                        else    {
                                y-=12;
                                oscilar=false;
                        }
                }
                x+=28;
        }
}
// Colocacion de leyenda Hz
SetTextColor(lpDC, RGB(0, 255, 0));                     // Color
TextOut(lpDC, x-15, y+(oscilar ? 15 : 15+12), "Hz", 2);
lpDDSBack->ReleaseDC(lpDC);

// El mensaje de ayuda...
lpFont=CreateFont(18,0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE,
                    ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS,
                    CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY,
                    VARIABLE_PITCH,
                    "Verdana");
if (lpFont == NULL)             {
        TRACE("ERROR: Creacion de la nueva fuente\n");
        g_Error=15;
        return -1;
}

lpDDSBack->GetDC(&lpDC);
SelectObject(lpDC, lpFont);
SetBkColor(lpDC, RGB(00, 00, 00));
SetTextColor(lpDC, RGB(52, 200, 255));
if (!m_Ayuda)
        TextOut(lpDC, 5, 5, "Mostrar ayuda; Pulse F1", 23);
else
        TextOut(lpDC, 5, 5, "Ocultar ayuda; Pulse F1", 23);
```

```
lpDDSBack->ReleaseDC(lpDC);

if (m_Ayuda)    {
        lpDDSBack->GetDC(&lpDC);
        // Mostrar ayuda sobre las teclas que se pueden utilizar
        SetTextColor(lpDC, RGB(30,180,230));
        SetBkColor(lpDC, RGB(00, 00, 00));
        TextOut(lpDC, 40, ddsd.dwHeight-50, "Scroll a la derecha",19);
        TextOut(lpDC, 40, ddsd.dwHeight-25, "Scroll a la Izquierda",21);
        TextOut(lpDC, 240, ddsd.dwHeight-50, "Aumenta amplitud",16);
        TextOut(lpDC, 240, ddsd.dwHeight-25, "Disminuye amplitud", 18);
        TextOut(lpDC, 490, ddsd.dwHeight-50,"Scroll rápido Izq.", 18);
        TextOut(lpDC, 490, ddsd.dwHeight-25,"Scroll rápido Der.", 18);
        TextOut(lpDC, ddsd.dwWidth-100, 20,"ESC :Salir", 10);
        lpDDSBack->ReleaseDC(lpDC);
        // Colocar los dibujillos
        rc.top=47;
        rc.left=0;
        rc.right=24;
        rc.bottom=rc.top+20;
        x=10;
        y=ddsd.dwHeight-50;
        for (i=0; i<6; i++)             {
                while (true)    {
                        // Copia desde el recurso bmp a la pantalla
                        hr=lpDDSBack->BltFast(x, y, lpDDSbmp, &rc,
                                      DDBLTFAST_NOCOLORKEY);
                        // Esperar
                        if (hr == DD_OK)
                                break;
                        else if (hr == DDERR_SURFACELOST)       {
                                hr = RestaurarMemVid();
                                if (hr == DD_OK)
                                   continue;
                        }
                    if (hr == DDERR_WASSTILLDRAWING)
                                continue;
                }
                switch (i)      {
                case 0:
                        rc.left+=25;
                        rc.right+=25;
                        y+=22;
                        break;
                case 1:
                        rc.left+=25;
                        rc.right+=25;
                        y-=22;
```

```
                                x+=200;
                                break;
                        case 2:
                                rc.left+=25;
                                rc.right+=25;
                                y+=22;
                                break;
                        case 3:
                                rc.left=0;
                                rc.right=72;
                                rc.top=68;
                                rc.bottom=68+20;
                                y-=22;
                                x+=200;
                                break;
                        case 4:
                                rc.left+=73;
                                rc.right+=73;
                                y+=22;
                                break;
                        }
                }
        }
        m_CambiarFondo++;
}

// Cuidado con GetDC y ReleaseDC porque usan internamente Lock y Unlock
// en la memoria de video con lo que Blt y demás funciones no pueden funcionar

// Puesta en pantalla de los indicadores luminosos...


x=0;                    // Va de izq a derecha
y=toty;                 // Por empezar desde arriba

// Gestion de la persistencia, es decir, se proporciona una memoria a los
// valores que se visulizarán para que la caida sea progresiva.
// Además, de esta manera las zonas críticas son menores y colisionan menos
// en el acceso a los arrays, la FFT y los cálculos graficos
m_Envejecer++;          // Cada refresco de pantalla que es cada 30msg
for(i=0; i<m_NCortes; i++)      {
        // Las subidas se notan instantáneas, no así la caída, que se envejecerán
        // los valores del array
        if (*(m_PersistenciaR+i) < *(m_OutR+i))
                *(m_PersistenciaR+i)=*(m_OutR+i);
        if (m_Envejecer == PASO_DEL_TIEMPO)             {
                *(m_PersistenciaR+i)-=CAIDA;
        }
```

```
}

for (i=m_Virt_X; i< (m_Virt_X+m_CortesPorPantalla < m_NCortes ? m_Virt_X+
                 m_CortesPorPantalla : m_NCortes); i++)        {
        // Gestion del nuemro de columnas
        j=m_NEscalones;
        do      {
                // Gestion de cada indicador dentro de cada columna
                j--;
                if (m_PersistenciaR[i] >= m_Escala_double[j])   {
                        // Se ilumina
                        do {    // Cuando se ilumina arriba, los de abajo tb
                                rc.left=27;
                                rc.top=m_Indice[j];
                                rc.right=27+24;
                                rc.bottom=m_Indice[j]+6;
                                while (true)    {
                   // Copia desde el bmp del recurso a la pantalla de atrás
                                        hr=lpDDSBack->BltFast(OffsetX+x,
                                            OffsetY-y, lpDDSbmp, &rc,
                                            DDBLTFAST_NOCOLORKEY);
                                        // Esperar
                                        if (hr == DD_OK)
                                                break;
                                        else if (hr == DDERR_SURFACELOST)       {
                                                hr = RestaurarMemVid();
                                                if (hr == DD_OK)
                                                   continue;
                                        }
                                if (hr == DDERR_WASSTILLDRAWING)
                                                continue;
                                        if (hr == DDERR_EXCEPTION || hr ==
                                            DDERR_GENERIC || hr ==
                                            DDERR_INVALIDOBJECT )  {
                                                g_Error=14;
                                                return -1;
                                        }
                                }
                                j--;
                                y-=Paso_y;
                        } while(j >= 0);
                }
                else    {               // No se activa el indicador
                        rc.left=0;
                        rc.top=m_Indice[j];
                        rc.right=24;
                        rc.bottom=m_Indice[j]+6;
```

```
                        while (true)    {
                                // Copia del led apagado a la pantalla de atrás
                                hr=lpDDSBack->BltFast(OffsetX+x ,OffsetY-y,
                                   lpDDSbmp, &rc, DDBLTFAST_NOCOLORKEY);
                                if (hr == DD_OK)
                                        break;
                                else if (hr == DDERR_SURFACELOST)       {
                                        hr = RestaurarMemVid();
                                        if (hr == DD_OK)
                                           continue;
                        }
                                if (hr == DDERR_WASSTILLDRAWING)
                        continue;
                                if (hr == DDERR_EXCEPTION || hr == DDERR_GENERIC
                                                || hr == DDERR_INVALIDOBJECT )  {
                                        g_Error=14;
                                        return -1;
                                }

                        }
                        y-=Paso_y;
                }
        } while (j > 0);
        x+=Paso_x;
        y=toty;
}

// Poner el otro bucle para los casos de stereo
if (m_Canales == 2)             {
        x=0;                    // Va de izq a derecha
        y=toty;                 // Por empezar desde arriba

        // Gestion de la persistencia al igual que en el otro canal, cuando
        // corresponda
        for(i=0; i<m_NCortes; i++)      {
                // Las subidas se notan instantáneas, no así la caída, que se
                // envejecerán los valores del array
                if (*(m_PersistenciaL+i)<*(m_OutL+i))
                        *(m_PersistenciaL+i)=*(m_OutL+i);
                if (m_Envejecer == PASO_DEL_TIEMPO)             {
                        *(m_PersistenciaL+i)-=CAIDA;
                }
        }

        for (i=m_Virt_X; i< (m_Virt_X+m_CortesPorPantalla<m_NCortes ?
                        m_Virt_X+m_CortesPorPantalla :  m_NCortes); i++)        {
                // Gestion del nuemro de columnas
                j=m_NEscalones;
```

```
do      {
        // Gestion de cada indicador dentro de cada columna
        j--;
        if (m_PersistenciaL[i] >= m_Escala_double[j])   {
                // Se iluminan
                do {  // Cuando se ilumina arriba los de abajo tb
                   rc.left=27;
                   rc.top=m_Indice[j];
                   rc.right=27+24;
                   rc.bottom=m_Indice[j]+6;
                   while (true)    {
             // Copia de led's encendidos a la pantalla de atrás
                         hr=lpDDSBack->BltFast(OffsetX+x, OffsetY_L-y,
                            lpDDSbmp, &rc, DDBLTFAST_NOCOLORKEY);
                         if (hr == DD_OK)
                               break;
                         else if (hr == DDERR_SURFACELOST)       {
                                hr = RestaurarMemVid();
                                if (hr == DD_OK)
                                    continue;
                         }
                         if (hr == DDERR_WASSTILLDRAWING)
                           continue;
                if (hr == DDERR_EXCEPTION || hr == DDERR_GENERIC
                             || hr == DDERR_INVALIDOBJECT )  {
                                    g_Error=14;
                                    return -1;
                         }
                   }
                   j--;
                   y-=Paso_y;
                } while(j >= 0);
        }
        else    {               // No se activa el indicador
                rc.left=0;
                rc.top=m_Indice[j];
                rc.right=24;
                rc.bottom=m_Indice[j]+6;
                while (true)    {
                        // Se copia el led apagado
                        hr=lpDDSBack->BltFast(OffsetX+x ,
                           OffsetY_L-y, lpDDSbmp, &rc,
                           DDBLTFAST_NOCOLORKEY);
                        if (hr == DD_OK)
                                break;
                        else if (hr == DDERR_SURFACELOST)       {
                                hr = RestaurarMemVid();
                                if (hr == DD_OK)
```

```
                                                continue;
                                        }
                                            if (hr == DDERR_WASSTILLDRAWING)
                                            continue;
                                                    if (hr == DDERR_EXCEPTION || hr ==
                                                       DDERR_GENERIC || hr ==
                                                       DDERR_INVALIDOBJECT )  {
                                                            g_Error=14;
                                                            return -1;
                                                    }
                                            }
                                            y-=Paso_y;
                                    }
                            } while (j > 0);
                            x+=Paso_x;
                            y=toty;
                    }
            }

            // Gestión de la variable que controla el envejecimiento

            if (m_Envejecer == PASO_DEL_TIEMPO)
                    m_Envejecer=0;

/*          Para poner por pantalla texto que distinga las dos surfaces o pantallas
            if (!fase)      {       // Esta mostrandose Primary Surface => pintar en Back
                    rc.top=0;
                    rc.left=0;
                    rc.right=24;
                    rc.bottom=8;
                    hr = lpDDSBack->GetDC(&lpDC);
                    SetBkColor(lpDC, RGB(0, 0, 0));
                    SetTextColor(lpDC, RGB(0, 250, 0));
               TextOut(lpDC, (ddsd.dwWidth)-134, (ddsd.dwHeight-54), "Alfonso Martín.", 15);
                    fase=true;

            }
            else    {               // Esta Back Surface en pantalla => construir Primary
                    // De momento no pone nada al estar ahi detras
                    hr = lpDDSBack->GetDC(&lpDC);
                    SetBkColor(lpDC, RGB(0, 250, 0));
                    SetTextColor(lpDC, RGB(0, 0, 0));
               TextOut(lpDC, (ddsd.dwWidth)-134, (ddsd.dwHeight-54), "Alfonso Martín.", 15);
                    fase=false;

            }
            hr =lpDDSBack->ReleaseDC(lpDC);
*/
```

```
        // Lo que hay que hacer en los dos casos puesto que al realizar la operacion
        // de  flip cambia las Surfaces pero los punteros siguen apuntando
        // Primary al que se ve y Back al que no se ve...

        return 0;

}


CDirectDraw::Configurar()
{
        // Segun las opciones seleccionadas por el usuario, se preparan los datos
        // para poder utilizar el visualizador correctamente

        int Cortes, Escalones, Frecuencia_max, Frecuencia_min;
        int Escalones_max, Escalones_min;
        bool Escala;          // Reciben datos del dialogo, debidamente tratados

        int j,i,k;            // Cuentas varias
        float tempf, fmin, fmax;
        int temp, h_min;

        char cadena[10];                        // Conversion de CString a int

        // Paso a variables locales para hacer cuentas sin peligro...

        Cortes=Graficos.m_Cortes;
        Escalones=Graficos.m_Escalones;
        Escalones_max=Graficos.m_Escalones_max;
        Escalones_min=Graficos.m_Escalones_min;
        Frecuencia_max=Graficos.m_Frecuencia_max;
        Frecuencia_min=Graficos.m_Frecuencia_min;
        Escala=Graficos.m_Escala ? true : false;

        // Estraccion de datos necesarios para Objeto Dd del dialogo de Muestreo
        Muestreo.m_Modo==0 ? m_Canales=1 : m_Canales=2;
        // Comprobacion de la entrada de datos
        if (Muestreo.m_Fs.IsEmpty())
                Muestreo.m_Fs="44100 Hz";
        for (i=0; i<Muestreo.m_Fs.GetLength() && Muestreo.m_Fs[i] !=' ' ; i++)  {
                cadena[i]=Muestreo.m_Fs[i];
        }
        cadena[i]='\0';
        m_Fs=atoi(cadena);

        // Obtencion del máximo de escalones segun los puntos en pantalla
        if (DD_OK != lpDD4->GetDisplayMode(&ddsd))              {
```

```
        TRACE("ERROR: Obtencion de la conf del display\n");
        return -1;
}

if (m_Canales == 1)     {
        // Mono
        temp=(7*ddsd.dwHeight/10)/6;
        if (Escalones > temp)
                m_NEscalones=temp;
        else
                m_NEscalones=Escalones;
}
else    {
        // Stereo
        temp=(7*ddsd.dwHeight/10)/12;
        if (Escalones > temp)
                m_NEscalones=temp;
        else
                m_NEscalones=Escalones;
}


// Para las amplitudes dejo como máximo una escala [-99, 0] dB, la tarjeta
// tiene un margen dinámico en el mejor de los casos de 90.3 dB => Suficiente

// Recibir aqui los max y min de las 2 esclas que se utilizan en el
// visualizador; es decir, en la frecuencia y la amplitud
// El usuario dice que fraccion de amplitud mostrar y cuantas divisiones

// Eje Ordenadas
j = Escalones_max-Escalones_min;        // Calcula el min espaciado (son -)
if (m_NEscalones < j)   {
        // Es correcto, el espaciado pedido es mayor que el minimo
        m_Escala= (int *) malloc(m_NEscalones * sizeof(int));
        m_Escala_double=(double *) malloc(m_NEscalones * sizeof(double));
        tempf=(float)j/(float)(m_NEscalones-1);         // El min ya esta asignado
        for (i=0; i<=m_NEscalones-1; i++)       {
                *(m_Escala+i)=(int) (Escalones_min+(i*tempf));
                *(m_Escala_double+i)=*(m_Escala+i);
        }
}
else    {
        // Si se piden más escalones que el espaciado minimo, luego se coge este
        // que es la unidad
        m_Escala= (int *) malloc(j * sizeof(int));
        m_Escala_double=(double *) malloc(j * sizeof(double));
        for (i=0; i<=Escalones; i++)    {
        *(m_Escala+i)=(int) Escalones_min+(i);
```

```
        *(m_Escala_double+i)=*(m_Escala+i);
        }
}

// Eje abcisas
if (Frecuencia_max <= Frecuencia_min)   {
        Frecuencia_min=20;
        Frecuencia_max=20000;   // Pongo lo que quiero
}

// Eje de Abcisas; Cortes de frecuencia
// Minimo espaciado 20 Hz, son 1000 Cortes para todo espectro audible
// Además es el máximo que permite la FFT con 2210 puntos

h_min=m_Fs/Fft.m_n;     // Espaciado mínimo en frecuencia, depende de
                        // Frecuencia de meustreo y puntos de la FFT
if (m_Fs == 44100)
        temp=20000;
else
        temp=m_Fs>>1;   // Banda representable en funcion de la Fs

m_CortesMax=temp/h_min;         // Todo el espectro

// Máximo Nž de cortes que se pueden representar según el espaciado mínimo

temp=(Frecuencia_max-Frecuencia_min)/Cortes;    // Espaciado mínimo pedido

if (Cortes >= m_CortesMax)
        m_NCortes=m_CortesMax;          // Se ajusta a lo máximo que se puede
else if (temp < h_min)
        m_NCortes=(Frecuencia_max-Frecuencia_min)/h_min;
        // Se ajusta a lo máximo que se puede
else
        m_NCortes=Cortes;

// Concordancia de la f max según la Fs
if (Frecuencia_max > m_Fs>>1)
        Frecuencia_max=m_Fs>>1;

// Una vez solventado los posibles errores en los parámetros de entrada se
// procede a los cálculos de los cortes en f

m_Frecuencias=(int *)malloc(m_NCortes * sizeof(int));   // Marcas en pantalla
m_Agrupacion=(int *)malloc(m_NCortes * sizeof(int));    // Para agrupar componetes
                                                        // de la FFT y representarlo en un corte
m_OutR=(double *)malloc(m_CortesMax * sizeof(double));
m_PersistenciaR=(double *)malloc(m_NCortes * sizeof(double));
```

```
m_PuntosFFT=(int *) malloc(m_CortesMax *sizeof(int));   // f de la FFT

for (i=0; i<m_CortesMax; i++)
        *(m_PuntosFFT+i)=h_min*(i+1);   // Frecuencias que obtiene la FFT

if (m_Canales == 2)     {
        m_OutL=(double *)malloc(m_CortesMax * sizeof(double));
        m_PersistenciaL=(double *)malloc(m_NCortes * sizeof(double));
}

// Ajuste de las frecuencias bajas, para comenzar siempre en 20 Hz o en la
// Frecuencia mínima que haya elegido el usuario, también hay que controlar que
// el espaciado sera menor, si Frecuencia de Muestreo es baja.
j=0;
while (h_min*(j+1) < (Frecuencia_min > 20 ? Frecuencia_min : 20))
        j++;

k=0;
if (m_NCortes != m_CortesMax)   {
        // Tener encuenta una escala lineal de menos puntos o mayor espaciado
        // que lo máximo permitido (Tramos de frecuencias)
        if (!Escala)    {
                fmin=(float)log(Frecuencia_min);
                fmax=(float)log(Frecuencia_max);  // Implementa una escala Log
                tempf=(fmax-fmin)/(Cortes-1);
                for (i=0; i<m_NCortes-1; i++)           {
                        temp=(int) exp(fmin+(tempf*i));
                        if (temp >= *(m_PuntosFFT+j))   {
                                *(m_Frecuencias+k)=temp;
                                k++;
                                j++;
                        }
                }
                *(m_Frecuencias+k)=Frecuencia_max;
// Quitar cortes con espaciado menor que 20 Hz, minimo que se calcula; estos
// se dan en f bajas por la escala Log
                m_NCortes=k+1;
        }
        else    {
                // Escala Lineal
                temp=(Frecuencia_max-Frecuencia_min)/m_NCortes;
                for (i=0; i<m_NCortes-1; i++)   {
                        *(m_Frecuencias+i)=Frecuencia_min+(temp*i);
                }
                *(m_Frecuencias+i)=Frecuencia_max;
        }

        // Hay que tener encuenta que no siempre se desea ver la banda completa
```

```
                // por lo que la agrupacion debe comenzar más arriba
                j=0;
                while (Frecuencia_min > ((j+1)*h_min))
                        j++;
                m_PuntofftInicio=j;
                for (i=0; i<m_NCortes-1; i++)   {
                        temp=(*(m_Frecuencias+i) + *(m_Frecuencias+(i+1)))>>1;
                        j++;
                        while (temp > *(m_PuntosFFT+j))
                                j++;
                        *(m_Agrupacion+i)=j;
                }
                *(m_Agrupacion+i)=(Frecuencia_max/h_min)-1;

        }
        // El último se rellena con el último m_CortesMax
        else    {
                // No se puede colocar una escala log si se quiere mostrar los componentes
                // Tal cual salen de la FFT porque eston están repartidos linealmente
                // sobre el dominio de la f
                for (i=0; i<m_CortesMax; i++)
                        *(m_Frecuencias+i)=h_min*(i+1); // Frecuencias que obtiene la FFT
        }

        // Inicializar los arrays de Persistencia para que estén a 0, al comenzar
        for (i=0; i< m_NCortes; i++)    {
                *(m_PersistenciaR+i)=-99;
                if (m_Canales == 2)
                        *(m_PersistenciaL+i)=-99;
        }

TRACE("EN CONFIGURAR EJES CALCULADOS\n");
/*m_CortesPorPantalla=35;
ddsd.dwWidth=1024;
ddsd.dwHeight=768;  // Util para pruebas sin necesidad de invocar a DirectDraw*/


        m_CortesPorPantalla=(ddsd.dwWidth-25)/28;
        m_Virt_X=0;                     // Inicializa para mostrar por la izq, las f bajas

        // Calculo de las coordenadas en la pantalla...

        // Obtener el area util de pantalla (Reduccion de la coordenada Y)

        if (m_Canales == 1)             {
                // Mono, un único canal
                totx=(m_NCortes-1)*28+24+10;
                toty=m_NEscalones*12;
```

```
if ((totx+25) < ddsd.dwWidth)   {       // Tener encuenta los nž de la izq
        OffsetX=ddsd.dwWidth>>1;
        OffsetY=ddsd.dwHeight>>1;       // Punto medio de la pantalla
        OffsetX-=totx>>1;
        OffsetY+=toty>>1;               // Ajustes para centrarlo
        OffsetX+=13;                    // Por las marcas de amplitud

        if (toty < (7*ddsd.dwHeight/10))        {
                // Todo bien, cabe a lo ancho y a lo alto
                // Los offset Y dentro del bmp son:
                // 1 para el Rojo, 10 Amarillo y 19 para el Verde
                Paso_y=12;
                Paso_x=28;
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                temp=5*m_NEscalones/10;         // 50% de color verde
                for (i=0; i<temp; i++)  {
                        *(m_Indice+i)=19;
                }
                j=i;
                temp=22*m_NEscalones/100;       // 22% En amarillo
                for (i=j; i<j+temp; i++)        {
                        *(m_Indice+i)=10;
                }
                j=i;
                for (i=j; i<m_NEscalones;i++)   { // Resto en Rojo
                        *(m_Indice+i)=1;
                }

        }
        else    {
                // No cabe de alto, hay que cambiar los indicadores por
                // los más estrechos
                // Los offset Y dentro del bmp son:
                // 28 para el Rojo, 34 Amarillo y 40 para el Verde
                Paso_y=6;
                Paso_x=28;
                toty=m_NEscalones*Paso_y;    // Rehacer cuentas para eje Y
                OffsetY=ddsd.dwHeight>>1;    // Punto medio de la pantalla
                OffsetY+=toty>>1;               // Ajustes para centrarlo
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                temp=5*m_NEscalones/10;         // 50% de color verde
                for (i=0; i<temp; i++)  {
                        *(m_Indice+i)=40;
                }
                j=i;
                temp=22*m_NEscalones/100;       // 22% En amarillo
                for (i=j; i<j+temp; i++)        {
```

```
                        *(m_Indice+i)=34;
                }
                j=i;
                for (i=j; i<m_NEscalones;i++)   { // Resto en Rojo
                        *(m_Indice+i)=28;
                }
        }
}
else    {
        // Hay mas cortes de los que pueden representarse...
        totx=(m_CortesPorPantalla-1)*28+24+10;
        OffsetX=ddsd.dwWidth>>1;
        OffsetX-=totx>>1;
        OffsetX+=13;                    // Por las marcas de amplitud
        if (toty < (7*ddsd.dwHeight/10))        {
           // Caben de alto, luego hay que implementar un tipo de Scroll
           // Se mostrarán los Cortes que entren simultaneamente

                OffsetY=ddsd.dwHeight>>1;   // Punto medio de la pantalla
                OffsetY+=toty>>1;           // Ajustes para centrarlo
                Paso_y=12;
                Paso_x=28;              // De alto no hay que hacer nada
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                temp=5*m_NEscalones/10;         // 50% de color verde
                for (i=0; i<temp; i++)  {
                        *(m_Indice+i)=19;
                }
                j=i;
                temp=22*m_NEscalones/100;       // 22% En amarillo
                for (i=j; i<j+temp; i++)        {
                        *(m_Indice+i)=10;
                }
                j=i;
                for (i=j; i<m_NEscalones;i++)   {// Resto en Rojo
                        *(m_Indice+i)=1;
                }
        }
        else    {
                // No cabe de alto, hay que cambiar los indicadores por
                // los más estrechos
                // Los offset Y dentro del bmp son:
                // 28 para el Rojo, 34 Amarillo y 40 para el Verde
                Paso_y=6;
                Paso_x=28;
                toty=m_NEscalones*Paso_y;    // Rehacer cuentas para eje Y
                OffsetY=ddsd.dwHeight>>1;    // Punto medio de la pantalla
                OffsetY+=toty>>1;            // Ajustes para centrarlo
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
```

```
                        temp=5*m_NEscalones/10;         // 50% de color verde
                        for (i=0; i<temp; i++)  {
                                *(m_Indice+i)=40;
                        }
                        j=i;
                        temp=22*m_NEscalones/100;       // 22% En amarillo
                        for (i=j; i<j+temp; i++)        {
                                *(m_Indice+i)=34;
                        }
                        j=i;
                        for (i=j; i<m_NEscalones;i++)  { // Resto en Rojo
                                *(m_Indice+i)=28;
                        }
                }
        }
}
else    {
        // Stereo, visualizacion de 2 canales a la vez; Arriba y abajo
        totx=((m_NCortes-1)*28)+24+10;
        toty=m_NEscalones*24;           // 24=12*2 Doble de alto
        if ((totx+25) < ddsd.dwWidth)   {       // Tener encuenta los nž de la izq
                // Entra en la pantalla, no Scroll
                OffsetX=ddsd.dwWidth>>1;
                OffsetX-=totx>>1;
                OffsetX+=13;                    // Por las marcas de amplitud

                if (toty < (7*ddsd.dwHeight/10))        {
                        // Todo bien, caben los dos visualizadores (tamaño normal)
                        Paso_y=12;
                        Paso_x=28;
                        OffsetY=ddsd.dwHeight>>2;
                        // 2 Puntos medio en la pantalla
                        OffsetY_L=ddsd.dwHeight-OffsetY; // Mitad inferior
                        OffsetY+=toty>>2;
                        OffsetY_L+=(toty>>2)-40;
                        // 2 Puntos medios para coordenada Y
                        m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                        temp=5*m_NEscalones/10;         // 50% de color verde
                        for (i=0; i<temp; i++)  {
                                *(m_Indice+i)=19;
                        }
                        j=i;
                        temp=22*m_NEscalones/100;       // 22% En amarillo
                        for (i=j; i<j+temp; i++)        {
                                *(m_Indice+i)=10;
                        }
                        j=i;
                        for (i=j; i<m_NEscalones;i++)   {// Resto en Rojo
```

```
                        *(m_Indice+i)=1;
                }
        }
        else    {
                // No cabe de alto, hay que cambiar los indicadores por
                // los más estrechos
                // Los offset Y dentro del bmp son:
                // 28 para el Rojo, 34 Amarillo y 40 para el Verde
                Paso_y=6;
                Paso_x=28;                  // Rehacer cuentas para eje Y
                toty=m_NEscalones*12;       // 12=6*2
                OffsetY=ddsd.dwHeight>>2;
                // 2 Puntos medio en la pantalla
                OffsetY_L=ddsd.dwHeight-OffsetY;// Mitad inferior
                OffsetY+=toty>>2;
                OffsetY_L+=(toty>>2)-40;
                // 2 Puntos medios para coordenada Y
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                temp=5*m_NEscalones/10;         // 50% de color verde
                for (i=0; i<temp; i++)  {
                        *(m_Indice+i)=40;
                }
                j=i;
                temp=22*m_NEscalones/100;       // 22% En amarillo
                for (i=j; i<j+temp; i++)        {
                        *(m_Indice+i)=34;
                }
                j=i;
                for (i=j; i<m_NEscalones;i++)  { // Resto en Rojo
                        *(m_Indice+i)=28;
                }
        }
}
else    {
        // Hay más cortes de los que se puede representar => SCROLL
        totx=(m_CortesPorPantalla-1)*28+24+10;
        OffsetX=ddsd.dwWidth>>1;
        OffsetX-=totx>>1;
        OffsetX+=13;                    // Por las marcas de amplitud
        if (toty < (7*ddsd.dwHeight/10))        {
                // Al menos caben de alto
                OffsetY=ddsd.dwHeight>>2;           // Mitad de arriba
                OffsetY_L=ddsd.dwHeight-OffsetY;    // Mitad de abajo
                OffsetY+=toty>>2;
                OffsetY_L+=(toty>>2)-40;
                Paso_y=12;
                Paso_x=28;
                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
```

```
                                temp=5*m_NEscalones/10;         // 50% de color verde
                                for (i=0; i<temp; i++)  {
                                        *(m_Indice+i)=19;
                                }
                                j=i;
                                temp=22*m_NEscalones/100;       // 22% En amarillo
                                for (i=j; i<j+temp; i++)        {
                                        *(m_Indice+i)=10;
                                }
                                j=i;
                                for (i=j; i<m_NEscalones;i++)   { // Resto en Rojo
                                        *(m_Indice+i)=1;
                                }
                        }
                        else    {
                                // Tampoco caben de alto, hay que cambiar los indicadores
                                // por los mas estrechos
                                // Los offset Y dentro del bmp son:
                                // 28 para el Rojo, 34 Amarillo y 40 para el Verde
                                Paso_y=6;
                                Paso_x=28;              // Rehacer cuentas para eje Y
                                toty=m_NEscalones*12;   // 12=6*2
                                OffsetY=ddsd.dwHeight>>2;          // Mitad de arriba
                                OffsetY_L=ddsd.dwHeight-OffsetY;   // Mitad de abajo
                                OffsetY+=toty>>2;
                                OffsetY_L+=(toty>>2)-40;
                                // Centra los dos canales en eje Y
                                m_Indice= (LPDWORD) malloc(m_NEscalones * sizeof(DWORD));
                                temp=5*m_NEscalones/10;         // 50% de color verde
                                for (i=0; i<temp; i++)  {
                                        *(m_Indice+i)=40;
                                }
                                j=i;
                                temp=22*m_NEscalones/100;       // 22% En amarillo
                                for (i=j; i<j+temp; i++)        {
                                        *(m_Indice+i)=34;
                                }
                                j=i;
                                for (i=j; i<m_NEscalones;i++)   { // Resto en Rojo
                                        *(m_Indice+i)=28;
                                }
                        }
                }
                toty=toty>>1;           // Solo mida un canal, es la referencia abajo
        }

TRACE("CALCULOS DE CENTRADO EN PANTALLA PASADOS; FIN CONFIGURAR\n");
        return 0;
```

```
}

void CDirectDraw::PreparardDatos()
{
        // Gestion de los datos devueltos por la FFT antes de pasarlos a la funcion
        // que los representará en la pantalla

        // Aqui se reliza desde el cálculo de los log de los datos hasta la
        // acomodacion del nuemro de cortes a representar con el numero de
        // cortes que devuelve la FFT

        int k,i,j;
        double temp;

        Fft.m_Csfft.Lock();             // No acceder si se está calculando la FFT

        // Tope de cuenta para todo el espectro hasta Frecuencia de Nyquist
        // (Fft.m_n+1)/2, para 44100 solo se calculan los 1000 primeros, pues el resto
        // se sale del espectro audible y no se representan => no los calcula como
        // tampoco se calcula el primero que es el nivel de Continua
        // Como se ha elegido un Nž impar de puntos no hay frecuencia de Nyquist, está
        // la DC y luego las componentes

        for (k = 1; k < m_CortesMax+1; ++k)  // (k < N/2 rounded up)
        *(m_OutR+(k-1)) =((*(Fft.m_RTCOutR+k)) * (*(Fft.m_RTCOutR+k))/ (Fft.m_n*Fft.m_n))+
                                ((*(Fft.m_RTCOutR+(Fft.m_n-k)) * (*(Fft.m_RTCOutR+
                                (Fft.m_n-k)))) / (Fft.m_n*Fft.m_n));

        if (m_NCortes != m_CortesMax)   {
                // Hay que agrupar para que coincidan los cortes a representar
                // Se usará el array m_Agrupacion para este menester
                temp=0;
                // K va a barrer todas las componentes calculadas, luego ha de empezar
                // por donde desea el usuario
                k=m_PuntofftInicio;
                j=0;
                for (i=0; i<m_NCortes; i++)             {
                        do      {
                                temp+=*(m_OutR+k);
                                j++;
                                k++;
                        } while(k<= *(m_Agrupacion+i));
                        *(m_OutR+i)=temp/j;
                        // Cálculo de los Log
                        if (*((m_OutR)+i) != 0)
                                *((m_OutR)+i)=(m_Cte)*log10(*((m_OutR)+i));
                        else
                                *((m_OutR)+i)=-99;
```

```
                j=0;
                temp=0;
        }
}
else    {
        // Se muestran las componentes en frecuencia tal cual se obtienen de la
        // FFT
        for (k=0; k<m_CortesMax; k++)   {
        if (*((m_OutR)+k) != 0)
                *((m_OutR)+k)=(m_Cte)*log10(*((m_OutR)+k));
        else
                *((m_OutR)+k)=-99;
        }
}

if (m_Canales == 2)     {
        // Hay 2 canales, modo Stereo

        for (k = 1; k < m_CortesMax+1; ++k)  // (k < N/2 rounded up)
                *(m_OutL+(k-1)) =((*(Fft.m_RTCOutL+k)) * (*(Fft.m_RTCOutL+k))/
                (Fft.m_n*Fft.m_n))+((*(Fft.m_RTCOutL+(Fft.m_n-k)) *
                (*(Fft.m_RTCOutL+(Fft.m_n-k)))) / (Fft.m_n*Fft.m_n));

        if (m_NCortes != m_CortesMax)   {
        // Hay que agrupar para que coincidan los cortes a representar
        // Se usará el array m_Agrupacion para este menester
                temp=0;
                k=m_PuntofftInicio;
                j=0;
                for (i=0; i<m_NCortes; i++)             {
                        do      {
                                temp+=*(m_OutL+k);
                                j++;
                                k++;
                        } while(k<= *(m_Agrupacion+i));
                        *(m_OutL+i)=temp/j;
                        // Gestion de los Log
                        if (*((m_OutL)+i) != 0)
                                *((m_OutL)+i)=(m_Cte)*log10(*((m_OutL)+i));
                        else
                                *((m_OutL)+i)=-99;
                        j=0;
                        temp=0;
                }
        }
        else    {
                // Se muestran los cortes tal cual los da la FFT
                for (k=0; k<m_CortesMax; k++)   {
```

```
                        if (*((m_OutL)+k) != 0)
                                *((m_OutL)+k)=(m_Cte)*log10(*((m_OutL)+k));
                        else
                                *((m_OutL)+k)=-99;
                        }
                }
        }
        Fft.m_Csfft.Unlock();   // Zona crítica
}


HRESULT CDirectDraw::RestaurarMemVid()
{
        // Por si la aplicación perdiera la posesión de la VGA y para continuar hay
        // que restaurar la memoria de video

    HRESULT  hr;

        hr = lpDDSPrimary->Restore();
    if (hr == DD_OK)
    {
        hr = lpDDSbmp->Restore();
        if (hr == DD_OK)
        {
            DDReLoadBitmap(lpDDSbmp, strbmp);
        }
    }
    return hr;
}


CDirectDraw::Destruir(bool Todo)
{
        // Volver al control de la pantalla desde la GDI de la MFC,
        // Destruccion de todas la interfaces


        TRACE("antes de LIBERACION DE TODOS LOS INTERFACES\n");
        if (lpDD4 != NULL)      {
                if (lpDDSBack != NULL)  {
                        // libera la Surface de atras
                        lpDDSBack->Release();
                        lpDDSBack=NULL;
                }
                if (lpDDSPrimary != NULL)       {
                        // Liberar los interfaces
                        lpDDSPrimary->Release();
                        lpDDSPrimary=NULL;
```

```
                }
                if (lpDDSbmp != NULL)   {
                        // Libera la memoria de video con los bmps
                        lpDDSbmp->Release();
                        lpDDSbmp=NULL;
                }
                if (Todo)       {
                        lpDD4->Release();
                        lpDD4=NULL;
                }
        }
        TRACE("LIBERACION DE TODOS LOS INTERFACES\n");

        // Liberacion de los arrays que se generan según las opciones del ususario
        free(m_Frecuencias);
        free(m_PuntosFFT);
        free(m_Escala);
        free(m_Escala_double);
        free(m_Agrupacion);
        free(m_OutR);
        free(m_Indice);
        free(m_PersistenciaR);
        if (m_Canales == 2)     {
                free(m_OutL);
                free(m_PersistenciaL);
        }

        return 0;
}

//////////////////////////////////////////////////////////////////////////////
//
// Funciones que pertenecen al Objeto DirectDraw pero que están tomadas de la
// SDK de DirectX 6.0 ddutil.h y ddutil.cpp son facilidades para importar y
// gestionar archivos bmp y meterlos en surfaces en memoria.
// Originalmente eran funciones C que se han integrado en mi objeto C++

//-----------------------------------------------------------------------------
// File: ddutil.cpp
//
// Desc: Routines for loading bitmap and palettes from resources
//
//
// Copyright (c) 1995-1998 Microsoft Corporation. All rights reserved.
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Name: DDLoadBitmap()
```

```
// Desc: Create a DirectDrawSurface from a bitmap resource.
//-----------------------------------------------------------------------------
CDirectDraw::DDLoadBitmap(IDirectDraw4 * pdd, LPCSTR szBitmap, int dx, int dy,
                                                    LPDIRECTDRAWSURFACE4* Salida)

//-----------------------------------------------------------------------------
// Name: DDReLoadBitmap()
// Desc: Load a bitmap from a file or resource into a directdraw surface.
//       normaly used to re-load a surface after a restore.
//-----------------------------------------------------------------------------
CDirectDraw::DDReLoadBitmap(IDirectDrawSurface4 * pdds, LPCSTR szBitmap)

//-----------------------------------------------------------------------------
// Name: DDCopyBitmap()
// Desc: Draw a bitmap into a DirectDrawSurface
//-----------------------------------------------------------------------------
CDirectDraw::DDCopyBitmap(IDirectDrawSurface4 * pdds, HBITMAP hbm, int x, int y,
             int dx, int dy)
```

3.1.4 DirectS

DirectS.h

```
// DirectS.h: interface for the CDirectS class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
#define AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_

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

#include "Global.h"


class CDirectS
{
        // Variables
private:
        int     m_Fs;           // Almacena la Frecuencia de muestreo com entero
        int m_Modo;             // Canales a tratar, Stereo(2) o Mono(1)
        int m_Bits;             // Bits por muestra 8 o 16

        int m_SubBuffer;        // Almacena cuanto mide el subbuffer
        int m_j, m_k;           // Contador para FFT
        bool contAnt;   // Para compensar los wraparound en bloqueo buffers
```

```
        LPDIRECTSOUND                   lpds;
        LPDIRECTSOUNDBUFFER             lpdsbPrimary, lpdsb;
        LPDIRECTSOUNDCAPTURE            lpdsc;
        LPDIRECTSOUNDCAPTUREBUFFER      lpdscb;
        LPDIRECTSOUNDNOTIFY             lpdsNotify; // Punteros a interfaces COM
        HANDLE                          rghEvent[NUMEVENTOS];
        DSBPOSITIONNOTIFY               rgdsbpn[NUMEVENTOS];
        DWORD                           m_TamBuffer;
        HMMIO                           hmmio;
        MMCKINFO                        mmckinfoData, mmckinfoParent;
        char                            *m_NArchivo;    // Nombre del archivo a analizar
        DWORD CursorLectura, CursorEscritura;           // Apuntan a los Buffers

public:
        HWND                      m_Hwnd; // Puntero a la ventana principal
        DSCAPS                    dscaps;
        DSCCAPS                   dsccaps; // Almacena propiedades de Objetos Ds y DsC
        GUID guidDs, guidDsC;              // Estructuras con info de los Dispositivos
        UINT TotalBytesEscritos, TotalBytesLeidos;      // Contadores para acceso a WAV

private:
        ProcesarEntrada(DWORD Posicion);
        ProcesarReproduccion(DWORD Posicion);
        TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total);
        TransferenciaFFT(LPBYTE Orig, DWORD Total);
        GenerarRR(LPBYTE Dest, DWORD Total);

        int WaveOpenFile(char *, HMMIO *, WAVEFORMATEX **, MMCKINFO *);
        int WaveStartDataRead(HMMIO *, MMCKINFO *, MMCKINFO *);
        int WaveReadFile(HMMIO, UINT, BYTE *, MMCKINFO *, UINT *);
        int WaveCloseReadFile(HMMIO *, WAVEFORMATEX **);

public:
        CDirectS();
        virtual ~CDirectS();
        CrearObjetos();
        Configurar(bool Captura);
        InicializarBuffers(bool Captura);
        LanzarCaptura();
        LanzarReproduccion();
        Parar();
        Liberar(bool Todo);

};

#endif // !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
```

DirectS.cpp

```
// DirectS.cpp: implementation of the CDirectS class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DirectS.h"
#include "Estructuras.h"
#include "Generar_RR.h"

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

const double pi = 3.1415926535;

CEvent g_TerminarHilo;

// Ruido Rosa
Generar_RR RR_8(128);           // Para 8 Bits
Generar_RR RR_16(65536);        // Para 16 Bits

extern CFft Fft;
extern Opc_Muestreo Muestreo;
extern Opc_Graficos Graficos;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CDirectS::CDirectS()
{
        // Inicializaciones adicionales; valores inicialies para punteros
        lpds=NULL;
        lpdsbPrimary=NULL;
        lpdsb=NULL;
        lpdsc=NULL;
        lpdscb=NULL;
        lpdsNotify=NULL;
}

CDirectS::~CDirectS()
{
}
```

```
CDirectS::Configurar(bool Captura)
{
        // Extrae los valores proporcionados por el usuario en el Dialogo_Muestreo y
        // los convierte a formatos deacuerdo con el Objeto Ds.
        // Pasa la Fs de CString a int, el Modo y Bits de bool a int

        WAVEFORMATEX *wfx;
        CString str;
        char temp[10];
        int i=0;

        if (Captura)    {
                // Se configura el Objeto DirectSound y sus buffers con los datos del
                // usuario, recogidos de "OPCIONES.CFG"

                // Comprobacion de la entrada de datos
                if (Muestreo.m_Fs.IsEmpty())
                        Muestreo.m_Fs="44100 Hz";

                for (i=0; i<Muestreo.m_Fs.GetLength() && Muestreo.m_Fs[i] !=' ' ; i++)  {
                        temp[i]=Muestreo.m_Fs[i];
                }
                temp[i]='\0';
                m_Fs=atoi(temp);
                Muestreo.m_Bits==0 ? m_Bits=8 : m_Bits=16;
                Muestreo.m_Modo==0 ? m_Modo=1 : m_Modo=2;
        }
        else    {
                // Se procede a leer un archivo wav con lo que hay que adaptar el objeto
                // DirectSound y sus buffers al formato en que haya sido grabado
                // Se leerá la configuracion y se escribira en la estructura muestreo,
                // para que el resto esté en cosonancia con los datos obtenidos

                // Conversion del nombre del archivo de CString a char
                m_NArchivo=(char *) malloc((Graficos.m_NArchivo.GetLength()+1)*
                           sizeof(char));

                for (int i=0; i<Graficos.m_NArchivo.GetLength(); i++)   {
                        m_NArchivo[i]=Graficos.m_NArchivo[i];
                }
                m_NArchivo[i]='\0';

                if (WaveOpenFile(m_NArchivo, &hmmio, &wfx, &mmckinfoParent) !=0)
                        return -1;
                // Si todo ha ido bien wfx posee el formato del archivo, se copia
                str.Format("%d",wfx->nSamplesPerSec);
                Muestreo.m_Fs=str;
                Muestreo.m_Bits=(wfx->wBitsPerSample == 8) ? 0 : 1;
```

```
                Muestreo.m_Modo=(wfx->nChannels == 1) ? 0 : 1;
                // Escribe el formato directo en las variables del objeto
                m_Fs=wfx->nSamplesPerSec;
                m_Bits=wfx->wBitsPerSample;
                m_Modo=wfx->nChannels;

                // Como solo se necesita el nombre del archivo para abrirlo se libera la
                // memoria, del array dinámico
                free(m_NArchivo);
        }
        return 0;
}


CDirectS::CrearObjetos()
{
        // Creacion de los objetos con sus interfaces para su posterior uso
        // Solo se crean una vez por ejecucion de la aplicacion, al fianlizar se liberan

        // Cuidado con las enumeraciones de los dispositivos

        m_Hwnd=AfxGetMainWnd()->GetSafeHwnd();
        if (lpds != NULL)       {
                lpds->Release();
                lpds=NULL;
        }
        // Creacion objeto DirectSound
        if (DS_OK != DirectSoundCreate(&guidDs, &lpds, NULL))   {
                // No se ha podido crear
                TRACE("Error en la creacion del DirectSound\n");
                return -1;
        }
        // No permite que suene nada más que esta aplicación EXCLUSIVE
        if (DS_OK != lpds->SetCooperativeLevel(m_Hwnd, DSSCL_EXCLUSIVE))        {
                // No se ha podido establecer la prioridad
                TRACE("ERROR: al establecer la prioridad\n");
                return -1;
        }

        if (lpdsc != NULL)      {
                lpdsc->Release();
                lpdsc=NULL;
        }
        // Creacion del objeto de Captura
        if (DS_OK != DirectSoundCaptureCreate(&guidDsC, &lpdsc, NULL))  {
                TRACE("Fue mal la creación de CDirectSoundCapture\n");
                return -1;
        }
```

```
        return 0;
}


CDirectS::InicializarBuffers(bool Captura)
{
        // Esta funcion crea los buffers necesarios para la aplicacion, es decir, se
        // crea lo necesario para un funcionamiento en FULL-DUPLEX, puesto que filtrado
        // lo requiere y Analisis si se quiere RuidoRosa tambien lo ha de implementar.
        // Así se crean 2 buffers para Reproduccion y 1 para captura:
        // Reproduccion: Primario, marca el formato en que se oirá y mezcla todos los
        // secundarios que se estén reproduciendo, el secundario recibira los datos del
        // de captura tratados o generados por el Ruido Rosa
        // Captura: Unicamente recibe datos de la Tarjeta para que se puedan leer

        WAVEFORMATEX            wfx;
        DSBUFFERDESC            dsbd;
        DSCBUFFERDESC           dscbd;

        LPBYTE LpByte, LpByte2;
        DWORD CuenByte2, CuenByte;

        // Todos los buffers tendrán el mismo formato, puesto que evita sobrecarga de
        // CPU y no es el propósito de la aplicacion convertir éstos
        ZeroMemory(&wfx, sizeof(WAVEFORMATEX));
        wfx.wFormatTag=WAVE_FORMAT_PCM;
        wfx.nChannels=m_Modo;
        wfx.nSamplesPerSec=m_Fs;
        wfx.wBitsPerSample=m_Bits;
        wfx.nBlockAlign=wfx.wBitsPerSample/8*wfx.nChannels;
        wfx.nAvgBytesPerSec=wfx.nSamplesPerSec*wfx.nBlockAlign;

        // Creacion del Buffer primario
        ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
        dsbd.dwSize=sizeof(DSBUFFERDESC);
        dsbd.dwFlags=DSBCAPS_PRIMARYBUFFER;
        if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsbPrimary, NULL))       {
                TRACE("Error:Creacion del buffer Primario\n");
                return -1;
        }

        // Ajuste del formato deseado para B.Primario
        if (DS_OK != lpdsbPrimary->SetFormat(&wfx))     {
                TRACE("ERROR:Ajuste del formato al buffer primario\n");
                return -1;
        }

        // Creacion del buffer de captura
```

```
// Cuentas para tamaño de los buffers
m_SubBuffer=max(4096, wfx.nAvgBytesPerSec/8);
m_SubBuffer -= m_SubBuffer % wfx.nBlockAlign;   // Quede N entero de muestras
                                                // en los buffers

// Creacion del buffer Secundario
// Debe ser exactamente igual que el de captura, porque entre ellos se pasarán
// los datos con o sin procesamiento
ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize=sizeof(DSBUFFERDESC);
dsbd.dwFlags=DSBCAPS_GETCURRENTPOSITION2        // Obtencion de cursor
                        | DSBCAPS_GLOBALFOCUS   // Se oiga siempre
                        | DSBCAPS_CTRLPOSITIONNOTIFY; // Use notificacion el buffer
dsbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
dsbd.lpwfxFormat=&wfx;
if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsb, NULL))      {
        TRACE("Error:Creacion Buffer Secundario\n");
        return -1;
}
// Por último llenar el buffer secundario de reproduccion con silencio, evita
// chispazos y picos indeseados
if (DS_OK != lpdsb->Lock(0, 0, (LPVOID*)&LpByte, &CuenByte,
        (LPVOID*)&LpByte2, &CuenByte2, DSBLOCK_ENTIREBUFFER))
        TRACE("Error al bloquear el buffer entero\n");
else    {
        if (m_Bits == 8)        {
                FillMemory(LpByte, CuenByte, 128);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 128);
        }
        else    {
                // Muestras de 16 Bits
                FillMemory(LpByte, CuenByte, 0);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 0);
        }
}
if (DS_OK != lpdsb->Unlock((LPVOID)LpByte, CuenByte, (LPVOID)LpByte2,
        CuenByte2))
        TRACE("Error al desbloquear el buffer entero\n");

m_TamBuffer=dsbd.dwBufferBytes;

// Creacion y configuracion de los eventos
// Tener encuenta quien pedirá eventos, captura (Analisis, Filtrado y Grabacion
// a archivo) o reproduccion (Lectura de archivo)
if (Captura)    {
        // Creación del buffer de Captura solo cuando sea necesario
```

```
        ZeroMemory(&dscbd, sizeof(DSCBUFFERDESC));
        dscbd.dwSize=sizeof(DSCBUFFERDESC);
        dscbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
        dscbd.lpwfxFormat=&wfx;

        if (DS_OK != lpdsc->CreateCaptureBuffer(&dscbd, &lpdscb, NULL)) {
                TRACE("Error:Creacion del buffer de Captura\n");
                return -1;
        }

        // Los Eventos se asociarán a la Captura
        for (int i=0; i<NUMEVENTOS; i++)        {
        rghEvent[i]=CreateEvent(NULL, false, false, NULL);
        if (NULL==rghEvent[i])
                return -1;
        }
        for (i=0; i<NUMEVENTOS; i++)    {
                rgdsbpn[i].dwOffset=((m_SubBuffer*i)+m_SubBuffer-1);
                rgdsbpn[i].hEventNotify=rghEvent[i];
        }

        //Asegura que lpdscNotify esta libre
        if (lpdsNotify != NULL) {
                lpdsNotify->Release();
                lpdsNotify=NULL;
        }
        if (DS_OK != lpdscb->QueryInterface(IID_IDirectSoundNotify,
                (void **)&lpdsNotify))  {
                g_Error=35;
                return -1;
        }
        if (DS_OK != lpdsNotify->SetNotificationPositions(NUMEVENTOS, rgdsbpn)) {
                lpdsNotify->Release();
                g_Error=35;
                return -1;
        }

}
else    {
        // Aqui se configuran los eventos sobre el buffer secundario, cuando se
        // lean archivos WAV

        // Ya se ha abierto el archivo para leer su formato...
        if (WaveStartDataRead(&hmmio, &mmckinfoData, &mmckinfoParent) != 0)
                return -1;

        // Creación de las notificaciones pero esta vez para el buffer secundario,
        // es decir, en la reproduccion
```

```
                for (int i=0; i<NUMEVENTOS; i++)        {
                rghEvent[i]=CreateEvent(NULL, false, false, NULL);
                if (NULL==rghEvent[i])
                        return -1;
                }
                for (i=0; i<NUMEVENTOS; i++)    {
                        rgdsbpn[i].dwOffset=((m_SubBuffer*i)+m_SubBuffer-1);
                        rgdsbpn[i].hEventNotify=rghEvent[i];
                }

                //Asegura que lpdscNotify esta libre
                if (lpdsNotify != NULL) {
                        lpdsNotify->Release();
                        lpdsNotify=NULL;
                }

                if (DS_OK != lpdsb->QueryInterface(IID_IDirectSoundNotify,
                        (void **)&lpdsNotify))  {
                        g_Error=35;
                        return -1;
                }
                if (DS_OK != lpdsNotify->SetNotificationPositions(NUMEVENTOS, rgdsbpn)) {
                        lpdsNotify->Release();
                        g_Error=35;
                        return -1;
                }
        }

        // En ambos casos hay que esperar a que la FFT esté inicializada para poder
        // comenzar o bien a capturar o a leer del archivo

        TRACE("SE HAN CREADO LOS EVENTOS...\n");
        m_j=0;          // Para controlar los datos a FFT
        m_k=0;          // Para la FFT de RuidoRosa

        do      {                           // Espera a que este preparado
                                            // el objeto para la Fft
        }while (WaitForSingleObject(g_FftOk, 0) != WAIT_OBJECT_0);
        // Para que no lo reseteé despues de Chequearlo
        g_FftOk.SetEvent();

        return 0;
}


CDirectS::LanzarCaptura()
{
        // Se centraliza en esta funcion, tanto la grabacion a disco, como el
```

```
// funcionamiento en FULL-DUPLEX de la aplicacion, puesto que los dos modos
// se basan en la captura y será esta la que dirija toda la ejecucion en la
// gestion de los eventos según se van produciendo y atendiendo

// Se podría seleccionar aquí según la modalidad de operacion elegida por el
// ususario..
// if (g_Grabar)        {
// if (g_Filtrado)      {
// En caso de que haya que grabar audio a un archivo, se inicializa aqui
// todo lo necesario para acceder al disco, preparacion del archivo WAV
// Solo en Ecualizador.exe
// Para el caso de filtrado que tampoco se trata en Analisis.exe pero se
// mantiene por consistencia con los otros programas y las variables
// globales


if (g_Analisis) {
        // Para Analisis y Filtrado lanzar la Captura, y para evitar los chispazos
        // picos o chasquidos de paro-marcha por la programación del DMA, se
        // coloca el buffer primario en reproduccion continua, así, esta
        // mezclando continuamente.
        if (DS_OK != lpdsbPrimary->Play(0, 0, DSBPLAY_LOOPING)) {
                TRACE("Error:Arranque del buffer Primario\n");
                g_Error=37;
                return -1;
        }
        if (DS_OK != lpdscb->Start(DSCBSTART_LOOPING))  {
                TRACE("Error: Arranque de la Captura\n");
                g_Error=36;
                return -1;
        }
        if (Graficos.m_RuidoRosa)               {
                if (DS_OK != lpdsb->SetCurrentPosition(0))      {
                        TRACE("Error:Posicionamiento antes de reproducir\n");
                        g_Error=37;
                        return -1;
                }
                if (DS_OK != lpdsb->Play(0, 0, DSBPLAY_LOOPING))        {
                        TRACE("Error:Arranque del buffer Secundario\n");
                        g_Error=37;
                        return -1;
                }
        }

        // Gestion de los eventos
        // Inicializa el cursor escribir datos en B.Secundario
        CursorEscritura=0xFFFFFFFF;
```

```
                BOOL Done=false;
                while(!Done)    {
                        DWORD dwEvt=MsgWaitForMultipleObjects(
                                NUMEVENTOS,             // Numero de Eventos
                                rghEvent,               // Array donde están los Handles
                                FALSE,                  // £Esperar a todos?
                                INFINITE,               // £Cuanto espero?
                                QS_ALLINPUT);           // £Cualquier Mensage es suceso?
                        // Se normalizan los eventos; WAIT_OBJECT_0=0;
                        dwEvt -= WAIT_OBJECT_0;
TRACE("                                                         dwEvt=%d\n",dwEvt);
                        if (dwEvt < (DWORD) NUMEVENTOS) {
                                ProcesarEntrada(dwEvt); // Copia datos al fichero o buffer
                        }
                        // Comprueba el evento de salida
                        if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                                Done=true;              // Sale del bucle
                        }
                }               // Final del procesamiento de Eventos
        }
        return 0;
}


CDirectS::LanzarReproduccion()
{
        // Funcion homóloga a LanzarCaptura() pero para lectura de un archivo wav, el
        // fundamento es el mismo solo que ahora en cada peticion de un evento, se
        // colocan datos delante para que los pueda ir leyendo

        // Arrancar la reproduccion de los buffers, primero el primario, para evitar
        // arranques y paradas en el DMA por el mezclador y por último el secundario

        int result;

        if (DS_OK != lpdsbPrimary->Play(0, 0, DSBPLAY_LOOPING)) {
                        TRACE("Error:Arranque del buffer Primario\n");
                        g_Error=37;
                        return -1;
                }
        if (DS_OK != lpdsb->SetCurrentPosition(0))      {
                TRACE("Error:Posicionamiento antes de reproducir\n");
                g_Error=37;
                return -1;
        }
        if (DS_OK != lpdsb->Play(0, 0, DSBPLAY_LOOPING))        {
                TRACE("Error:Arranque del buffer Secundario\n");
                g_Error=37;
```

```
                return -1;
        }

        TotalBytesLeidos=0;

        // Gestion de los eventos
        // Inicializa el cursor escribir datos en B.Secundario
        CursorEscritura=0xFFFFFFFF;

        result=0;
        BOOL Done=false;
        while(!Done)    {
                DWORD dwEvt=MsgWaitForMultipleObjects(
                        NUMEVENTOS,             // Numero de Eventos
                        rghEvent,               // Array donde están los Handles
                        FALSE,                  // £Esperar a todos?
                        INFINITE,               // £Cuanto espero?
                        QS_ALLINPUT);           // £Cualquier Mensage es suceso?
                // Se normalizan los eventos; WAIT_OBJECT_0=0;
                dwEvt -= WAIT_OBJECT_0;
TRACE("                                                         dwEvt=%d\n",dwEvt);
                if (dwEvt < (DWORD) NUMEVENTOS) {
                        result=ProcesarReproduccion(dwEvt);    // Copia datos al buffer
                }
                // Comprueba el evento de salida
                if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                        Done=true;              // Sale del bucle
                }
                if (result == 1)        {
                        Done=true;
                }
        }               // Final del procesamiento de Eventos

        // Si se ha salido, solo queda cerrar el archivo
        // Estructura WAVEFORMATEX necesaria para cerra el archivo
        WAVEFORMATEX wfx, *pwfx;

        ZeroMemory(&wfx, sizeof(WAVEFORMATEX));
        wfx.wFormatTag=WAVE_FORMAT_PCM;
        wfx.nChannels=m_Modo;
        wfx.nSamplesPerSec=m_Fs;
        wfx.wBitsPerSample=m_Bits;
        wfx.nBlockAlign=wfx.wBitsPerSample/8*wfx.nChannels;
        wfx.nAvgBytesPerSec=wfx.nSamplesPerSec*wfx.nBlockAlign;

        pwfx=&wfx;
        WaveCloseReadFile(&hmmio, &pwfx);
```

```
        // Para distinguir si el usuario a decidido salir o se terminó el archivo wav
        if (result == 1)        {
                // Se ha terminado el archivo; asi que hay que terminar desde aqui todo
                // lo relacionado con DirectDraw
                g_PararTimer.SetEvent();     // Desencadena la parada, igual que cuando se
                                             // captura ESCAPE en la vista
        }
        return 0;
}


CDirectS::Parar()
{
        // Centraliza las órdenes de para de los buffers
        if (lpdscb != NULL)
                lpdscb->Stop();         // Para la Captura
        if (lpdsbPrimary != NULL)
                lpdsbPrimary->Stop();
        if (g_Filtrado || (g_Analisis && Graficos.m_RuidoRosa)) {
                if (lpdsb != NULL)
                        lpdsb->Stop();
        }
        return 0;
}


CDirectS::Liberar(bool Todo)
{
        // LLeva a cabo la tarea de la liberacion de todo; tanto los buffers solamente,
        // si se cambia el formato, u objetos y buffers si se procede a salir de la
        // aplicacion

        if ((lpdscb !=NULL))    {
                lpdscb->Release();
                lpdscb=NULL;
        }
        if (lpdsbPrimary != NULL)       {
                lpdsbPrimary->Release();
                lpdsbPrimary=NULL;
        }
        if (lpdsb != NULL)              {
                lpdsb->Release();
                lpdsb=NULL;
        }
        if (lpdsNotify != NULL) {
                lpdsNotify->Release();
                lpdsNotify=NULL;
        }
```

```
        if ((lpdsc != NULL) && Todo)    {
                lpdsc->Release();
                lpdsc=NULL;
        }
        if ((lpds != NULL) && Todo)             {
                lpds->Release();
                lpds=NULL;
        }
        return 0;
}


CDirectS::ProcesarEntrada(DWORD Posicion)
{
        // Gestion de los bloqueos de los buffers tanto de Captura como de reproduccion,
        // para pasar los datos de uno a otro, o bien directamente copiarlo a un archivo
        // .WAV, puesto que los eventos en ambos casos son del Buffer de Captura

        DWORD                   dwNumBytes;
        LPBYTE                  pbInput1, pbInput2, pbOutput1, pbOutput2;
        DWORD                   cbInput1, cbInput2, cbOutput1, cbOutput2;

        if (Posicion == 0)      {
                // Primer SubBuffer del Buffer
                CursorLectura=0;
                dwNumBytes=(rgdsbpn[Posicion].dwOffset)+1;      // Cálculo de bytes mover
        }
        else    {
                // Resto de subdivisiones, en las que está compuesto el buffer
                CursorLectura=(rgdsbpn[Posicion-1].dwOffset)+1;
                dwNumBytes=m_SubBuffer;
                // rgdsbpn[Posicion].dwOffset - rgdsbpn[Posicion-1].dwOffset;
                // Da igual restar la distancia entre subbufers contiguos
        }
        // Bloqueo del buffer de Captura

// Nunca hay Wraparound porque las peticiones de bloqueo se hacen ajustandose
// al buffer de Captura, esta subdividido en 16 bloques y se seleccionan cada
// uno de éstos, por lo que pbInput2 y cbInput2 no sirven para nada

        if (DS_OK != lpdscb->Lock(CursorLectura, dwNumBytes,
                        (LPVOID *)&pbInput1, &cbInput1,
                        (LPVOID *)&pbInput2, &cbInput2, 0))     {
                TRACE("Error:Bloqueo Buffer de Captura\n");
                g_Error=38;
                return -1;
        }
```

```
else    {
        if (g_Analisis) {
           // Calcular la FFT a partir de las muestras capturadas, asi que solo
           // atender buffer de captura
                TransferenciaFFT(pbInput1, cbInput1);

           // Una vez que se ha gestionado la captura se procede a la gestion del
           // Ruido Rosa si lo seleccionó el usuario
                if (Graficos.m_RuidoRosa)       {
                   // Hay que gestionar para poder escuchar el ruido rosa, el
                   // buffer secundario
                        if (CursorEscritura == 0xFFFFFFFF)      {
          // En la primera escritura, hay que saber por donde va reproducciendo
          // y adelantarse un poco para que no nos pille poniendo datos
                                if (DS_OK != lpdsb->GetCurrentPosition(NULL,
                                    &CursorEscritura)) {
                                        TRACE("Error:Peticion de posicion en Buffer
                                               Secundario\n");
                                        g_Error=39;
                                }
                                else    {
                                        // Adelanta 2, 3 o 4 SubBuffers
                                        CursorEscritura+=(2*m_SubBuffer);
                                        while (CursorEscritura >= m_TamBuffer)
                                                CursorEscritura-=m_TamBuffer;
                                }
                        }

// Una vez hechas la comprobaciones pertinentes, por si es la primera vez que
// Bloqueamos el buffer, se procede a bloquear el buffer y a leer del archivo
                        if (DS_OK != lpdsb->Lock(CursorEscritura, m_SubBuffer,
                                        (LPVOID *)&pbOutput1, &cbOutput1,
                                        (LPVOID *)&pbOutput2, &cbOutput2, 0))   {
                                TRACE("Error:Bloqueo del buffer Secundario\n");
                                g_Error=39;
                        }
         // Da igual con dwNumBytes, coinciden siempre, manda la captura
                        else    {
                                GenerarRR(pbOutput1, cbOutput1);

                                // Actualizar los cursores para la siguiente
                                CursorEscritura+=cbOutput1;
                                while (CursorEscritura >= m_TamBuffer)
                                        CursorEscritura-=m_TamBuffer;

                                if (pbOutput2 != NULL)  {
                                        TRACE("Hay pbOutput2\n");
                                        GenerarRR(pbOutput2, cbOutput2);
```

```
                                                CursorEscritura+=cbOutput2;
                                                while (CursorEscritura >= m_TamBuffer)
                                                        CursorEscritura-=m_TamBuffer;
                                        }
                                }
                                // Desbloqueo del Buffer Secundario
                                if (DS_OK != lpdsb->Unlock(pbOutput1, cbOutput1,
                                    pbOutput2, cbOutput2)){
                                        TRACE("Error:Desbloqueo del Buffer Secundario\n");
                                        g_Error=41;
                                }
                        }
                }
                // Desbloqueo del buffer de captura
                if (DS_OK != lpdscb->Unlock(pbInput1, cbInput1, pbInput2, cbInput2))  {
                        TRACE("Error:Desbloqueo del Buffer de Captura\n");
                        g_Error=40;
                        return -1;
                }
        }
        // Puesta en orden del Cursor de Lectura
        CursorLectura+=dwNumBytes;
        while (CursorLectura >= m_TamBuffer)
                CursorLectura-=m_TamBuffer;

        return 0;
}


CDirectS::ProcesarReproduccion(DWORD Posicion)
{
        // Gestion de la transferencia entre el archivo y el buffer secundario
        // Al igual que en captura cada vez que se bloquea el buffer es para mover
        // al 1 subBuffer, por lo tanto solo se calcula la posicion y no la cantidad

        LPBYTE                  pbOutput1, pbOutput2;
        DWORD                   cbOutput1, cbOutput2;
        UINT                    BytesLeidos;
        bool                    salir=false;

        if (CursorEscritura == 0xFFFFFFFF)      {
        // La primera escritura, hay que saber por donde va reproducciendo
        // y adelantarse un poco para que no nos pille poniendo datos
                if (DS_OK != lpdsb->GetCurrentPosition(NULL, &CursorEscritura)) {
                        TRACE("Error:Peticion de posicion en Buffer Secundario\n");
                        g_Error=39;
                }
```

```
        else    {
                // Adelanta 2, 3 o 4 SubBuffers, para que se note poco el retraso
                // entre lo que se muestra por pantalla lo que se oye.
                CursorEscritura+=(1*m_SubBuffer);
                while (CursorEscritura >= m_TamBuffer)
                        CursorEscritura-=m_TamBuffer;
        }
}

// Una vez hechas la comprobaciones pertinentes, por si es la primera vez que
// Bloqueamos el buffer, se procede a bloquear el buffery a leer del archivo
if (DS_OK != lpdsb->Lock(CursorEscritura, m_SubBuffer,
                (LPVOID *)&pbOutput1, &cbOutput1,
                (LPVOID *)&pbOutput2, &cbOutput2, 0))   {
        TRACE("Error:Bloqueo del buffer Secundario\n");
        g_Error=39;
}
// En este caso tambien se mueve una cantidada fija porque está todo en el
// archivo y se puede coger cuando se quiera
else    {
// En este momento el buffer secundario tiene un trozo bloqueado por delante
// de donde reproduce asi que se escribirám ahí los datos leidos del archivo
        WaveReadFile(hmmio, cbOutput1, (BYTE *)pbOutput1, &mmckinfoData,
                &BytesLeidos);

        // Como en las funciones de lectura del archivo se extraen los datos Byte
        // Byte se gestiona la adaptación de los datos con la FFT una vez en
        // memoria porque el direccionamiento es más fácil, creo, más rápido y
        // puedo usar TransferenciaFFT(...)
        TransferenciaFFT(pbOutput1, cbOutput1);

        // Hay que comprobar si se acaba el fichero, esto sucederá cuando se lean
        // menos bytes de los pedidos
        if (BytesLeidos < cbOutput1)    {
                // Se terminó el archivo
                // Se rellena lo que queda de buffer con silencio
                if (m_Bits == 8)        {
                        FillMemory(pbOutput1+BytesLeidos, cbOutput1-BytesLeidos,
                                   128);
                        if (pbOutput2 != NULL)
                                FillMemory(pbOutput2, cbOutput2, 128);
                }
                else    {
                        // Muestras de 16 Bits
                        FillMemory(pbOutput1+BytesLeidos, cbOutput1-BytesLeidos,
                                   0);
                        if (pbOutput2 != NULL)
                                FillMemory(pbOutput2, cbOutput2, 0);
```

```
                        }
                        salir=true;             // Se teminó el archivo WAV
                }
                // Actualizar los cursores para la siguiente
                CursorEscritura+=BytesLeidos;
                while (CursorEscritura >= m_TamBuffer)
                        CursorEscritura-=m_TamBuffer;
                TotalBytesLeidos+=BytesLeidos;

                if (pbOutput2 != NULL)  {
                     TRACE("Hay pbOutput2\n");
                // Si que puede haber wraparound porque la primera vez que escribimos
                // no lo hacemos coincidiendo exactamente con los subbuffers, puesto que
                // se pide info acerca de donde es seguro escribir y se adelante varios
                // subbuffers esa posicion...
                        WaveReadFile(hmmio, cbOutput2, (BYTE *)pbOutput2, &mmckinfoData,
                                &BytesLeidos);
                        // Cálculo de la FFT
                        TransferenciaFFT(pbOutput2, cbOutput2);
                        if (BytesLeidos < cbOutput2)    {
                                // Si se teminó el archivo...
                                if (m_Bits == 8)        {
                                        FillMemory(pbOutput2+BytesLeidos,
                                                   cbOutput2-BytesLeidos, 128);
                                }
                                else    {
                                        // Muestras de 16 Bits
                                        FillMemory(pbOutput2+BytesLeidos,
                                                   cbOutput2-BytesLeidos, 0);
                                }
                                salir=true;
                        }
                        CursorEscritura+=BytesLeidos;
                        while (CursorEscritura >= m_TamBuffer)
                                CursorEscritura-=m_TamBuffer;
                        TotalBytesLeidos+=BytesLeidos;
                }
        }
        // Desbloqueo del Buffer Secundario
        if (DS_OK != lpdsb->Unlock(pbOutput1, cbOutput1, pbOutput2, cbOutput2)){
                TRACE("Error:Desbloqueo del Buffer Secundario\n");
                g_Error=41;
        }
        if (salir)
                return 1;               // Si se acabó el fichero WAV

        return 0;
}
```

```
CDirectS::TransferenciaFFT(LPBYTE Orig, DWORD Total)
{
        // Realiza las transferencias entre el buffer de captura y prepara los datos
        // para pasarlos al objeto que calcula la FFT.

        register DWORD i;       // Contador
        DWORD *DOrig32;         // Para apuntar los buffer en accesos DWORD
        WORD  *WOrig16;         // Barrer la memoria con accesos WORD

        if (m_Bits == 8)        {
           // No se aplica la ventana de Hanning porque al tener el valor en DC de 128
           // el hecho de multiplicar por la ventana mete un tono de fs/2205.
           // 8 bits se usará poco y los errores son en las colas de los tonos que
           // no coinciden con la discretización de la FFT, la máxima precisión la
           // da a 44100 y 16 Bits
                if (m_Modo == 1)        {
                        // 8 Bits y Mono
                        for (i=0; i<Total; i++)         {
                                // Ajuste con rfftw
                                *((Fft.m_RTCInR)+m_j)=(BYTE)(*Orig++);
                                *((Fft.m_RTCInR)+m_j)=((*((Fft.m_RTCInR)+m_j))/64);
                                m_j++;
                                if (m_j >= Fft.m_n)     {
                                        Fft.CalcularTF(Fft.m_RTCInR, Fft.m_RTCOutR);
                                        m_j=0;
                                }
                        }
                }
                else    {
                        // 8 Bits y Stereo
                        WOrig16=(LPWORD)Orig;
                        Total=Total>>1;         // Mitad, porque el acceso es de 16 bits
                        for (i=0; i<Total; i++)         {
                                *((Fft.m_RTCInR)+m_j)=*((LPBYTE)WOrig16);
                                *((Fft.m_RTCInL)+m_j)=*((LPBYTE)((LPBYTE)WOrig16+1));
                                *((Fft.m_RTCInR)+m_j)=*((Fft.m_RTCInR)+m_j)/64;
                                *((Fft.m_RTCInL)+m_j)=*((Fft.m_RTCInL)+m_j)/64;
                                m_j++;
                                if (m_j >= Fft.m_n)     {
                                        Fft.CalcularTF(Fft.m_RTCInR, Fft.m_RTCOutR);
                                        Fft.CalcularTF(Fft.m_RTCInL, Fft.m_RTCOutL);
                                        m_j=0;
                                }
                                WOrig16++;
                        }
                }
```

```
        }
        else    {
                if (m_Modo == 1)        {
                        // 16 Bits y Mono
                        WOrig16=(LPWORD)Orig;
                        Total=Total>>1; // Mitad, para transferencias de 16 bits
                        for (i=0; i<Total ; i++)        {
                                // Aqui el ajuste con rfftw ...............
                                *((Fft.m_RTCInR)+m_j)=((short int)*WOrig16++)*(.5+.5*
                                                    cos(2*pi*(m_j-1102)/2205));
                                *((Fft.m_RTCInR)+m_j)=(*((Fft.m_RTCInR)+m_j)/16384);
                                m_j++;
                                if (m_j >= Fft.m_n)     {
                                        Fft.CalcularTF(Fft.m_RTCInR, Fft.m_RTCOutR);
                                        m_j=0;
                                }
                        }
                }
                else    {
                        // 16 Bits y Stereo
                        DOrig32=(LPDWORD)Orig;
                        Total=Total>>2 ;   // Division entre 4, transferencias 32 Bits
                        for (i=0; i<Total; i++) {
                                // Ajuste con rfftw
                                *((Fft.m_RTCInR)+m_j)=((short int)(*(LPWORD)DOrig32))*
                                                     (.5+.5*cos(2*pi*(m_j-1102)/2205));
                                *((Fft.m_RTCInL)+m_j)=((short int)(*(LPWORD)
                                  ((LPWORD)DOrig32+1)))*(.5+.5*cos(2*pi*(m_j-1102)/2205));
                                *((Fft.m_RTCInR)+m_j)=*((Fft.m_RTCInR)+m_j)/16384;
                                *((Fft.m_RTCInL)+m_j)=*((Fft.m_RTCInL)+m_j)/16384;
                                m_j++;
                                if (m_j >= Fft.m_n)     {
                                        Fft.CalcularTF(Fft.m_RTCInR, Fft.m_RTCOutR);
                                        Fft.CalcularTF(Fft.m_RTCInL, Fft.m_RTCOutL);
                                        m_j=0;
                                }
                                DOrig32++;
                        }
                }
        }
        return 0;
}

CDirectS::TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total)
{
        // Realiza las transferencias entre buffers, con dos transformadas FFT y
        // filtrado entre medias.
        // Se realiza la FFT del T a F, se modifican las componentes en la frecuencia
```

```
        // como corresponde a un filtro y por último se calcula FFT de F a T y se
        // llevan los datos al buffer de salida

        // Esta función miembro en este programa no se usa nunca

        // Comprueba como van colocados los datos en los buffers
        int flag=0;
        for (UINT i=0; i<Total; i++)    {
                switch (flag)   {
                        case 0:
                                *Dest=*Orig;
                                break;
                        case 1:
                                *Dest=*Orig;
                                break;
                        case 2:
                                *Dest=*Orig;
                                break;
                        case 3:
                                *Dest=*Orig;
                                break;
                }
                Dest++;
                Orig++;
                flag++;
                if (flag >3) flag=0;
                //*Dest++=*Orig++;              // Transferencia directa entre buffers
        }
        if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                m_j=0;                      // Al terminar se puede cambiar de formato
                                            // luego se inicializa todo
                g_TerminarHilo.SetEvent();  // Evita Resets automaticos por el chequeo
        }

        return 0;
}


CDirectS::GenerarRR(LPBYTE Dest, DWORD Total)
{
        // Genera datos para el ruido rosa, se generan en frecuencia y se transforman
        // al dominio del tiempo para mandarlos a la tarjeta de sonido

        DWORD i;
        short int tempW;
        unsigned int tempD;
        DWORD *DDest32;         // Para apuntar los buffer en accesos DWORD
        WORD  *WDest16;         // Barrer la memoria con accesos WORD
```

```
//      RAND_MAX=0x7fffh => 32768       Tope superior para la generacion aleatoria de rand
                                      // Definido en Stdlib.h

        if (m_Bits == 8)        {
                if (m_Modo == 1)        {
                        // 8 Bits y Mono
                        for (i=0; i<Total; i++)         {
                                // Se genra el Ruido Rosa directamente en el dominio del
                                // tiempo así que se escribe directamente el resultado
                                // en el buffer secundario
                                *Dest++=(BYTE)RR_8.GetNextValue();
                        }
                }
                else    {
                        // 8 Bits y Stereo
                        // Acceso de 16 bits a los buffers
                        WDest16=(LPWORD) Dest;
                        Total=Total>>1;                 // Mitad de transferencias
                        for (i=0; i<Total; i++)         {
                                tempW=(BYTE)RR_8.GetNextValue();
                                tempW=tempW<<8;
                                tempW+=(BYTE)RR_8.GetNextValue();
                                *WDest16++=tempW;
                        }
                }
        }
        else    {
                if (m_Modo == 1)        {
                        TRACE("VEAMOS EL TOTAL=%d\n",Total);
                        // 16 Bits y Mono
                        // Transferencias a los buffer de 16 Bits
                        WDest16=(LPWORD) Dest;
                        Total=Total>>1;                 // Mitad de transferencias
                        for (i=0; i<Total; i++)         {
                                // Centrado en 0=>[-32768, 32768]
                                *WDest16++=(short int)RR_16.GetNextValue()-32768;
                        }
                }
                else    {
                        // 16 Bits y Stereo
                        // Transferencias de 32 bits
                        DDest32=(LPDWORD)Dest;
                        Total=Total>>2;
                        for (i=0; i<Total; i++)         {
                                tempD=(short int)RR_16.GetNextValue()-32768;
                                tempD=tempD<<16;
                                tempD+=(short int)RR_16.GetNextValue()-32768;
                                *DDest32++=tempD;
```

```
                        }
                }
        }
        return 0;
}



// Funciones proporcionadas por el SDK de DirectX 6.0, para la gestion de archivos
// multimedia en general y wav en particular, no están desarrollados por mi.
// Funciones implementadas en C añadidas a mi objeto en C++, necesita wave.h, donde
// se encuentran las cabeceras

/*==========================================================================
 *
 *  Copyright (C) 1995-1997 Microsoft Corporation. All Rights Reserved.
 *
 *  File:               wave.c
 *  Content:    Wave library routines.
 *      This file is used for loading/saving waves, and reading and
 *      writing waves in smaller blocks.
 *      Uses WaveOpenFile, WaveReadFile and WaveCloseReadFile for
 *      single block access to reading wave files.
 *      Uses WaveCreateFile, WaveWriteFile, WaveCloseWriteFile for
 *      single block access for writing  wave files.
 *      Uses WaveLoadFile to load a whole wave file into memory.
 *      Uses WaveSaveFile to save a whole wave file into memory.
 *
 ***************************************************************************/


/* ROUTINES */
/* -------------------------------------------------------*/

/* This function will open a wave input file and prepare it for reading,
 * so the data can be easily
 * read with WaveReadFile. Returns 0 if successful, the error code if not.
 *      pszFileName - Input filename to load.
 *      phmmioIn    - Pointer to handle which will be used
 *          for further mmio routines.
 *      ppwfxInfo   - Ptr to ptr to WaveFormatEx structure
 *          with all info about the file.
 *
*/
CDirectS::WaveOpenFile(
        char *pszFileName,                              // (IN)
        HMMIO *phmmioIn,                                // (OUT)
        WAVEFORMATEX **ppwfxInfo,                       // (OUT)
```

```
        MMCKINFO *pckInRIFF                             // (OUT)
                        )


/*  This routine has to be called before WaveReadFile as it searchs for the chunk to
        descend into for reading, that is, the 'data' chunk.  For simplicity, this used
        to be in the open routine, but was taken out and moved to a separate routine so
        there was more control on the chunks that are before the data chunk, such as
        'fact', etc... */
CDirectS::WaveStartDataRead(
                                        HMMIO *phmmioIn,
                                        MMCKINFO *pckIn,
                                        MMCKINFO *pckInRIFF
                                        )

/*  This will close the wave file openned with WaveOpenFile.
        phmmioIn - Pointer to the handle to input MMIO.
        ppwfxSrc - Pointer to pointer to WaveFormatEx structure.

        Returns 0 if successful, non-zero if there was a warning.
*/
CDirectS::WaveCloseReadFile(
                        HMMIO *phmmio,                          // IN
                        WAVEFORMATEX **ppwfxSrc                 // IN
                        )

/*  This will read wave data from the wave file.  Makre sure we're descended into
        the data chunk, else this will fail bigtime!
        hmmioIn         - Handle to mmio.
        cbRead          - # of bytes to read.
        pbDest          - Destination buffer to put bytes.
        cbActualRead- # of bytes actually read.
*/
CDirectS::WaveReadFile(
                HMMIO hmmioIn,                          // IN
                UINT cbRead,                            // IN
                BYTE *pbDest,                           // IN
                MMCKINFO *pckIn,                        // IN.
                UINT *cbActualRead                      // OUT.

                )
```

3.1.5 Estructuras.h

```
// Estructuras para recoger las opciones de los dialogos
// Que contendrán los datos para la personalizacion del analisis
struct Opc_Muestreo
{
```

```
        CString m_Fs;
        int     m_Bits;
        int     m_Modo;
};

struct Opc_Graficos
{
        int             m_Cortes;
        BOOL            m_Escala;
        int             m_Escalones;
        int             m_Escalones_max;
        int             m_Escalones_min;
        int             m_Frecuencia_max;
        int             m_Frecuencia_min;
        int             m_RuidoRosa;
        int             m_Archivo;
        CString         m_NArchivo;
};
```

3.1.6 Fft

Fft.h

```
// Fft.h: interface for the CFft class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_)
#define AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_

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

#include "Global.h"

class CFft
{
public:
        int DestruirPlan();
        int CalcularTF(fftw_real *In, fftw_real *Out);
        int CalcularFT(fftw_real *In, fftw_real *Out);
        int CrearPlan();
        CFft();
        virtual ~CFft();

private:
        rfftw_plan m_p, m_q;
```

```
public:
        fftw_real *m_RTCInR, *m_RTCOutR;        // Para el canal de la Derecha (R)
        fftw_real *m_RTCInL, *m_RTCOutL;        // Para el canal de la Izquierda (L)
        fftw_real *m_CTRInR, *m_CTROutR;
        fftw_real *m_CTRInL, *m_CTROutL;
        int m_n;                                // La precision de la transformada
        BOOL m_PlanTF, m_PlanFT;
        CCriticalSection m_Csfft;

};

#endif // !defined(AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_)
```

Fft.cpp

```
// Fft.cpp: implementation of the CFft class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Estructuras.h"
#include "Fft.h"
#include "Global.h"

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

CEvent g_FftOk;

extern CDirectS Ds;
extern Opc_Muestreo Muestreo;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFft::CFft()
{
        m_PlanFT=false;
        m_PlanTF=false;
        m_RTCInR=NULL;
        m_RTCOutR=NULL;
        m_RTCInL=NULL;
        m_RTCOutL=NULL;
        m_CTRInR=NULL;
```

```
        m_CTROutR=NULL;
        m_CTRInL=NULL;
        m_CTROutL=NULL;
}

CFft::~CFft()
{
}

int CFft::CrearPlan()
{
        // Antes de realizar el cálculo de la Real FFT hay que hacer cálculos
        // previos antes de la transformada

        // fftw_real in[N], out[N], power_spectrum[N/2+1];
        // Se crean los arrays que contedran la entrada y salida de datos con el
        // objeto puesto que se lanzará con el hilo que lo ejecuta y si cambia
        // se destruye el hilo y se vuelve a crear con la nueva configuracion

        m_n=2205;                       // Son 2204 componentes y la DC

        // Prototipo:
        // rfftw_plan rfftw_create_plan(int n, fftw_direction dir, int flags);

    //m_p = rfftw_create_plan(m_n, FFTW_REAL_TO_COMPLEX, FFTW_MEASURE);

    // Los flags son: FFTW_MEASURE or FFTW_ESTIMATE
        // La direccion: FFTW_REAL_TO_COMPLEX or FFTW_COMPLEX_TO_REAL

        // Prototipo:
        // rfftw_plan rfftw_create_plan_specific(int n, fftw_direction dir, int flags,
        //              fftw_real *in, int istride, fftw_real *out, int ostride);

        if (!m_PlanTF)  {
                g_CreandoPlan=2;
                m_RTCInR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
                m_RTCOutR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
                m_RTCInL=(fftw_real*) malloc(m_n * sizeof(fftw_real));
                m_RTCOutL=(fftw_real*) malloc(m_n * sizeof(fftw_real));

                // Se crea el plan de conversion en la direccion Tiempo -> frecuencia
                m_p=rfftw_create_plan_specific(m_n, FFTW_REAL_TO_COMPLEX,
                        FFTW_ESTIMATE, m_RTCInR, 1, m_RTCOutR, 1);
                m_PlanTF=true;
                TRACE("SE HA CREADO PLAN T->F\n");
        }
        if (!m_PlanFT)          {
                g_CreandoPlan=2;
```

```
            m_CTRInR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
            m_CTRInL=(fftw_real*) malloc(m_n * sizeof(fftw_real));
            m_CTROutR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
            m_CTROutL=(fftw_real*) malloc(m_n * sizeof(fftw_real));


            // Se crea el plan de conversion de Frecuencia -> Tiempo
            m_q=rfftw_create_plan_specific(m_n, FFTW_COMPLEX_TO_REAL,
                    FFTW_ESTIMATE, m_RTCInR, 1, m_RTCOutR, 1);
            m_PlanFT=true;
            TRACE("SE HA CREADO PLAN F->T\n");
        }

        // Para evitar posibles chispazos en el buffer de salida, puesto que los
        // buffers se rellenan con silencios, los primeros 2205 se copian antes de
        // calcular ninguna FFT, es necesario poner silencio, pero deacuerdo con
        // el buffer, el tipo de datos que lleve; 8 ó 16 bits

        for (int i=0; i<m_n; i++)       {
            if (Muestreo.m_Bits)    {
                    // Datos de 16 bits=> Ca2 luego 0
                    *(m_CTROutR+i)=0;
                    *(m_CTROutL+i)=0;
            }
            else    {
                    // Datos de 8 bits, poner 128 que esta el 0
                    *(m_CTROutR+i)=128;
                    *(m_CTROutL+i)=128;
            }
            // Resetear tambien las salidas de los arrays con las componenetes en f
            // por la persistencia, que no coja valores muy altos que costará tiempo
            // en envejecer
            *(m_RTCOutR+i)=0;
            *(m_RTCOutL+i)=0;
        }

        g_FftOk.SetEvent();
        g_CreandoPlan=false;

        // Comprobar si la opcion de creacion puede acarrear algún tipo de error para
        // recogerlo aqui, de momento asin.

        g_Error=0;

        return 0;
}
```

```
int CFft::CalcularTF(fftw_real *In, fftw_real *Out)
{
        // Esta funcion lanza la rutina encargada de transformar los datos
        // Realiza el calculo del dominio del tiempo a frecuencia
        // Prototipo:
        // void rfftw_one(rfftw_plan plan, fftw_real *in, fftw_real *out);

        m_Csfft.Lock();                         // Entra en seccion crítica
        rfftw_one(m_p, In, Out);
        m_Csfft.Unlock();                       // Sale de la seccion critica

        return 0;
}


int CFft::CalcularFT(fftw_real *In, fftw_real *Out)
{
        // En esta funcion se calcula la FFt del dominio de la frecuencia al tiempo

        m_Csfft.Lock();
        rfftw_one(m_q, In, Out);
        m_Csfft.Unlock();

        return 0;
}


int CFft::DestruirPlan()
{
        // Antes de terminar hay que destruir el plan creado
        rfftw_destroy_plan(m_p);
        rfftw_destroy_plan(m_q);

        // Liberacion de la memoria asignada dinámicamente

        free(m_RTCInR);
        free(m_RTCOutR);
        free(m_RTCInL);
        free(m_RTCOutL);
        free(m_CTRInR);
        free(m_CTROutR);
        free(m_CTRInL);
        free(m_CTROutL);

        return 0;
}
```

3.1.7 Generar_RR

Generar_RR.h

```
// Generar_RR.h: interface for the Generar_RR class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_GENERAR_RR_H__8ECEC3E1_64AE_11D5_871D_848C2C8D2C7A__INCLUDED_)
#define AFX_GENERAR_RR_H__8ECEC3E1_64AE_11D5_871D_848C2C8D2C7A__INCLUDED_

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

// Esta clase la he cogido de internet de la pagina
// www.firstpr.com.audsppink-noise.html
// El autor es Thomas Hudson

class Generar_RR
{
private:
        int max_key;
        int key;
        unsigned int white_values[8];
        unsigned int range;

public:
        Generar_RR(unsigned int range);
        virtual ~Generar_RR();
        int GetNextValue();
};

#endif // !defined(AFX_GENERAR_RR_H__8ECEC3E1_64AE_11D5_871D_848C2C8D2C7A__INCLUDED_)
```

Generar_RR.cpp

```
// Generar_RR.cpp: implementation of the Generar_RR class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Analisis.h"
#include "Generar_RR.h"

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

```
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Generar_RR::Generar_RR(unsigned int range)
{
        // Constructor
        // range defina el margen dinámico de los datos que contendrán el ruido rosa
        // Se selecciona desde el exterior, según la configuración de la tarjeta de
        // sonido 8 o 16 bits 256 o 65536

        max_key = 0xff;                 // 5,6, 7 u 8 bits set
        this->range = range;
        key = 0;
        for (int i = 0; i < 8; i++)
                white_values[i] = rand() % (range/8);

        return;
}

Generar_RR::~Generar_RR()
{
}


int Generar_RR::GetNextValue()
{
        int last_key = key;
        unsigned int sum;
        key++;
        if (key > max_key)
                key = 0;
        // Exclusive-Or previous value with current value. This gives
        // a list of bits that have changed.

        int diff = last_key ^ key;
        sum = 0;
        for (int i = 0; i < 8; i++)             {
                // If bit changed get new random number for corresponding white_value
                if (diff & (1 << i))
                        white_values[i] = rand() % (range/8);
                sum += white_values[i];
        }
        return sum;
}
```

3.1.8 MainFrm

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__7FA5BF4B_4AE6_11D5_871D_8995A9A7F779__INCLUDED_)
#define AFX_MAINFRM_H__7FA5BF4B_4AE6_11D5_871D_8995A9A7F779__INCLUDED_

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

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        // Mis funciones...
        void Serialize(CFile *File, bool Salvar);
        void OnParar();

        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

protected:
        // Variables necesarias...

// 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)
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnComenzarAnalisis();
        afx_msg void OnClose();
        //}}AFX_MSG
        afx_msg void OnError();
        DECLARE_MESSAGE_MAP()
};


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

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

#endif // !defined(AFX_MAINFRM_H__7FA5BF4B_4AE6_11D5_871D_8995A9A7F779__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Analisis.h"

#include "MainFrm.h"
#include "Global.h"
#include "Estructuras.h"

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

CDirectS Ds;
CDirectDraw Dd;
CFft Fft;

Opc_Muestreo Muestreo;
Opc_Graficos Graficos;

int g_Error;
bool g_Analisis;
bool g_Filtrado;
```

```
bool g_Grabar;
int g_CreandoPlan;

CEvent g_PararTimer;

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_TIMER()
        ON_COMMAND(ID_COMENZAR_ANALISIS, OnComenzarAnalisis)
        ON_WM_CLOSE()
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_ERROR, OnError)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // Inicializaciones de variables globales para que todo vaya bien
        g_Analisis=false;
        g_Filtrado=false;
        g_Grabar=false;
        g_CreandoPlan=1;
        g_PararTimer.ResetEvent();
        g_FftOk.ResetEvent();

}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        // Modifico las opciones por defecto de la venta, puesto que no se usa para nada
        // se pretende que sea lo justo para mostrar el mensage y pasar al modo pantalla
        // completa controlado por DirectDraw

        // Sin botones de Maximizar ni minimizar, ni borde para reescalar
    cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER;

    // El tamaño 1/3 del tamaño de la pantalla y centrado
```

```
    cs.cy = (::GetSystemMetrics(SM_CYSCREEN) / 3) < 256 ? 256 :
                (::GetSystemMetrics(SM_CYSCREEN) / 3);
        cs.cx = (::GetSystemMetrics(SM_CXSCREEN) / 3) < 341 ? 341 :
                (::GetSystemMetrics(SM_CXSCREEN) / 3);
        cs.y = cs.cy >= 256 ? ((cs.cy * 3) - cs.cy) / 2 :
                ::GetSystemMetrics(SM_CYSCREEN)/2 - cs.cy;
        cs.x = cs.cx >= 341 ? ((cs.cx * 3) - cs.cx) / 2 :
                ::GetSystemMetrics(SM_CXSCREEN) /2 - cs.cx;

    // Call the base-class version
        return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnComenzarAnalisis()
{
        // En esta funcion se centraliza todas la inicializaciones y configuraciones
        // necesarias para que el Analisis gráfico en frecuencia sea posible

        // Se supone que existe un archivo con nombre "Opciones.cfg" en el mismo
        // directorio en el que está este ejecutable y la lanzadera, puesto que de él
        // se recoge las preferencias del usuario, en cuanto a la captura y los
        // parámetros para la visualizacion

        CFile File;
        CString str("OPCIONES.CFG");
        int result;

        if (!g_Analisis)        {
                // Evita Reentradas que provoquen errores inesperados
                // Lectura del archivo
```

```
if (!File.Open(str, CFile::modeRead))   {
        TRACE("No se ha podido leer OPCIONES.CFG\n");
        g_Error=90;
        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
        return;
        }
else    {
        // Lectura del archivo
        Serialize(&File, false);
}

g_TerminarHilo.ResetEvent();
g_Analisis=true;

// Comprobacion de la instalacion de los DirectX o cualquier otro error
if (g_Error != 0)       {
        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
        return;
}

result=Ds.CrearObjetos();
if (result != 0)        {
        g_Error=30;
        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
        return;
}
// Se Crean una sola vez los Objetos DirectDraw
result=Dd.Crear();
if (result != 0)        {
        g_Error=10;
        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
        return;
}

// Llamada a la Inicializacion del Plan para FFT
Fft.CrearPlan();

if (!Graficos.m_Archivo)                {
// Analisis del audio que se captura con la tarjeta de sonido, se prepara
// con los datos de "OPCIONES.CFG" y los eventos son controlados por la
// captura.

// Inicializacion del Objeto DirectSound
        result=Ds.Configurar(true);
        if (result != 0)        {
                g_Error=31;
                ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                return;
```

```
                }

// Segun el parametro que se le pase a InicializarBuffers, el Objeto Ds, creara
// los eventos en el buffer de Captura (Analiss, Filtrado y Grabacion) o en el
// secundario (Lectura de archivo WAV)

                result=Ds.InicializarBuffers(true);
                if (result != 0)        {
                        g_Error=32;
                        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                        return;
                }
        }
        else    {
        // En el caso de que haya que leer de un archivo, implica que hay que
        // cambiar, la configuracion al formato que tenga el archivo y la
        // estrategia de eventos en cuanto a los buffers

        // Inicializacion del Objeto DirectSound
                result=Ds.Configurar(false);
                if (result != 0)        {
                        g_Error=31;
                        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                        return;
                }

                // La inicialización se hace ahora teniendo encuenta los eventos
                // en la reproducción puesto que se leerá un archivo del disco
                result=Ds.InicializarBuffers(false);
                if (result != 0)        {
                        g_Error=32;
                        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                        return;
                }
        }

        // Por último lanzar el audio para que comienze la captura y reproduccion,
        // cuando sea necesario
        AfxBeginThread(Hilo_LanzarAudio, Ds.m_Hwnd,
                       THREAD_PRIORITY_TIME_CRITICAL);

        // La parte grafica ...
        ShowCursor(false);              // hace desaparecer el puntero del raton

        result=Dd.IniciarSurfaces();
        if (result != 0)        {
                g_Error=10;
                ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
```

```
                        return;
                }
                result=Dd.Configurar(); // Se configura...
                if (result != 0)        {
                        g_Error=11;
                        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                        return;
                }

                // Una vez que está todo se puede lanzar el temporizador para que comience
                // a refrescar la pantalla
                UINT timer=SetTimer(1, 30, NULL);               //40 msg => 25 fps
                ASSERT(timer != 0);
        }
        return;
}


void CMainFrame::OnTimer(UINT nIDEvent)
{
        // Control del refresco de pantalla para no interrumpir mientras se esta
        // realizando, no pararlo a medias

        int result;

        if (nIDEvent == 1)              {
                // Solo mi temporizador
                g_RefrescoPantalla.ResetEvent();
                result=Dd.Refrescar();
                if (result != 0)        {
                        // Algo fue mal
                        KillTimer(1);
                        ::PostMessage(Ds.m_Hwnd, WM_ERROR, 0, 0);
                }
                g_RefrescoPantalla.SetEvent();
                if (WaitForSingleObject(g_PararTimer, 0) == WAIT_OBJECT_0)      {
                        // Se ha pulsado ESCAPE hay que terminar
                        g_PararTimer.SetEvent();  // Evita el reset por el test positivo
                        OnParar();
                }
        }
        CFrameWnd::OnTimer(nIDEvent);
}


void CMainFrame::OnParar()
{
        // Activa el objeto evento para que en el bucle que recoge los eventos
```

```
        // de gestion del buffer compruebe si tiene que terminar.
        // No es posible enviarle un mensaje puesto que no tiene ventana es un
        // hilo trabajador

        g_TerminarHilo.SetEvent();              // Manda parar los dos hilos

        UINT timer=KillTimer(1);      // Para el temporizador  y saca de la cola las
        ASSERT(timer != 0);           // las peticiones existentes

        Sleep(200);             // Para que termine...

        Dd.Restaurar();
        ShowCursor(true);       // Devuelve el puntero del raton a la vista
        Dd.Destruir(false);
        // Repinte la GDI toda la pantalla
        DrawMenuBar();
        RedrawWindow(NULL, NULL, RDW_FRAME | RDW_INVALIDATE | RDW_ERASE);
        UpdateWindow();         // Se fuerza un repintado de la pantalla

        // Colocar las variables de estado en reposo
        Sleep(1000);            // No se por qué, pero le va bien
        g_Analisis=false;
        g_Error=0;                      // Da igual

        PostMessage(WM_CLOSE, 0 ,0);    // Cierra la aplicación

        return;
}


void CMainFrame::Serialize(CFile *File, bool Salvar)
{
        // Para Guardar o Cargar del disco el contenido de algunas variables miembro

        int i;

        // save=true Salva los datos al disco y save=false carga datos del disco
        if (Salvar)     {
                // Se escribe en el disco...
                CArchive Archivo(File, CArchive::store, 512);
                CObject::Serialize(Archivo);
                if (Archivo.IsStoring())        {
                        // Aqui solo se lee...
                }
        }
        else    {
                // Se lee del disco...
                CArchive Archivo(File, CArchive::load, 512);
```

```
                CObject::Serialize(Archivo);            // Llamada a la clase base
                if (!Archivo.IsStoring())       {
                        Archivo >> Muestreo.m_Fs >> Muestreo.m_Bits >> Muestreo.m_Modo;
                        // Hay que rescatar la estructuras elemento por elemento
                        Archivo >> Ds.guidDs.Data1 >> Ds.guidDs.Data2 >> Ds.guidDs.Data3;
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDs.Data4[i];
                        Archivo >> Ds.guidDsC.Data1 >> Ds.guidDsC.Data2
                                >> Ds.guidDsC.Data3;
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDsC.Data4[i];

                        Archivo >> Graficos.m_Cortes >> Graficos.m_Escalones
                                >> Graficos.m_Escalones_max;
                        Archivo >> Graficos.m_Escalones_min >> Graficos.m_Frecuencia_max;
                        Archivo >> Graficos.m_Frecuencia_min >> Graficos.m_Escala;

                        // Opciones adidionales
                        Archivo >> Graficos.m_RuidoRosa >> Graficos.m_Archivo;
                        if (Graficos.m_Archivo)
                                Archivo >> Graficos.m_NArchivo;
                }
        }
        return;
}

void CMainFrame::OnClose()
{
        // Gestion de la aplicacion antes de terminar
        // En caso de cerrar la aplicacion se comprueba que se deja todo para
        // evitar fugas de memoria o punteros asignados

        if (g_Analisis) {
                // Esta todo funcionando Analizando => hay que pararlo todo
                g_TerminarHilo.SetEvent();      // Solo para el Hilo NO PROGRESAN LOS MSG
                g_PararTimer.SetEvent();
                Sleep(250);
                Ds.Liberar(true);               // Para y Libera el Objeto Ds
                Dd.Destruir(true);              // Destruye los interfaces a DirectDraw
        }

        if (g_CreandoPlan==2 || g_CreandoPlan==0)       {
                Fft.DestruirPlan();
        }
        TRACE("PASO POR ONCLOSE\n");
        CFrameWnd::OnClose();
}
```

```
void CMainFrame::OnError()
{
        // Gestion de los mensajes de error
        // Si el flujo del programa a llegado hasta aquí algo ha fallado...
        // La variable global g_Error contendrá el entero que identificará que
        // error es el que ha sucedido;

        CDialog_Error Dlg;              // Crea un objeto para representar el dialogo

        switch (g_Error)        {
        case 15:
        case 14:
        case 13:
        case 12:
        case 11:
                Dd.Restaurar();
        case 10:
                ShowCursor(true);
                Dd.Destruir(true);
        case 41:
        case 40:
        case 39:
        case 38:
        case 37:
        case 36:
                Ds.Parar();
        case 35:
        case 34:
        case 33:
        case 32:
        case 31:
        case 30:
                Ds.Liberar(true);
                break;
        default:
                break;
        }

        Dlg.DoModal();                  // Lanza el dialogo a la pantalla

        while (g_CreandoPlan == 2)      {
                // Espera a que acabe de crear plan para FFT
                g_Analisis=true;
                g_Filtrado=true;
        }
        g_Analisis=false;
        g_Filtrado=false;
```

```
        PostMessage(WM_CLOSE, 0, 0);
        return;
}



// ##############################################################################
// ##
// ## Funcón global que no tiene nada que ver con el Objeto DirectS, necesaria
// ## por motivos de implementación de hilos en MFC
// #######

UINT Hilo_LanzarAudio(LPVOID pParam)
{
        // Funcion externa que envuelve a la funcion que realmente controla la gestión
        // del audio, por requerimeintos de los hilos

        int result;

        if (g_Analisis)         {
                if (!Graficos.m_Archivo)                {
                // Analisis normal, capturando la señal de la tarjeta de sonido
                        result=Ds.LanzarCaptura();
                        if (result != 0)
                                PostMessage((HWND) pParam, WM_ERROR, 0, 0);
                }
                else    {
                        // Hay que relizar el analisis leyendo de un archivo wav
                        result=Ds.LanzarReproduccion();
                        if (result != 0)
                                PostMessage((HWND) pParam, WM_ERROR, 0, 0);
                }
        }
        // Si hay error tambien hay que parralo todo, asi que dejo que siga
        // la parada normal...
        // g_Error se asigna donde se produzca el error

        // Deja el Ojeto Ds preparado para lo siguiente
        Ds.Parar();
        Ds.Liberar(false);

        g_TerminarHilo.ResetEvent();            // Desseñala para la siguiente vez
        return 0;
}
```

3.2 Ecualizador

Se detallará acontinuación los archivos principales que generan el lanzador de la aplicación, es decir, *Ecualidor.exe*.

3.2.1 Dialog_CapsCaptura

Dialog_CapsCaptura.h

```
#if !defined(AFX_DIALOG_CAPSCAPTURA_H__6D0A71E1_55AA_11D5_871D_8990FE7CD679__INCLUDED_)
#define AFX_DIALOG_CAPSCAPTURA_H__6D0A71E1_55AA_11D5_871D_8990FE7CD679__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_CapsCaptura.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsCaptura dialog

class CDialog_CapsCaptura : public CDialog
{
// Construction
public:
        CDialog_CapsCaptura(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialog_CapsCaptura)
        enum { IDD = IDD_DIALOG_CAPSCAPTURA };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_CapsCaptura)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_CapsCaptura)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

```
};

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

#endif // !defined(AFX_DIALOG_CAPSCAPTURA_H__6D0A71E1_55AA_11D5_871D_8990FE7CD679
                   __INCLUDED_)
```

Dialog_CapsCaptura.cpp

```
// Dialog_CapsCaptura.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_CapsCaptura.h"
#include "Global.h"

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

extern CDirectS Ds;

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsCaptura dialog


CDialog_CapsCaptura::CDialog_CapsCaptura(CWnd* pParent /*=NULL*/)
        : CDialog(CDialog_CapsCaptura::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDialog_CapsCaptura)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CDialog_CapsCaptura::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDialog_CapsCaptura)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CDialog_CapsCaptura, CDialog)
        //{{AFX_MSG_MAP(CDialog_CapsCaptura)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsCaptura message handlers

BOOL CDialog_CapsCaptura::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Preparación de los datos para mostrarlos en el dialogo

        if (Ds.dsccaps.dwFlags & DSCCAPS_EMULDRIVER)
                SetDlgItemText(IDC_EMULDRIVER, "Sí");
        else
                SetDlgItemText(IDC_EMULDRIVER, "No");

        SetDlgItemInt(IDC_DWCHANNELS, Ds.dsccaps.dwChannels, false);

        // Busco el mayor fromato de captura que soporta el dispositivo
        if (Ds.dsccaps.dwFlags & WAVE_FORMAT_4S16)      {
                // 44.1 kHz, stereo, 16-bit
                SetDlgItemText(IDC_DWFORMATS, "44.1 kHz, stereo, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_4M16) {
                // 44.1 kHz, mono, 16-bit
                SetDlgItemText(IDC_DWFORMATS,"44.1 kHz, mono, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_4S08) {
                // 44.1 kHz, stereo, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "44.1 kHz, stereo, 8-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_4M08)         {
                // 44.1 kHz, mono, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "44.1 kHz, mono, 8-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_2S16)         {
                // 22.05 kHz, stereo, 16-bit
                SetDlgItemText(IDC_DWFORMATS, "22.05 kHz, stereo, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_2M16)         {
                // 22.05 kHz, mono, 16-bit
                SetDlgItemText(IDC_DWFORMATS, "22.05 kHz, mono, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_2S08)         {
```

```
                // 22.05 kHz, stereo, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "22.05 kHz, stereo, 8-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_2M08)         {
                // 22.05 kHz, mono, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "22.05 kHz, mono, 8-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_1S16)         {
                // 11.025 kHz, stereo, 16-bit
                SetDlgItemText(IDC_DWFORMATS, "11.025 kHz, stereo, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_1M16)         {
                // 11.025 kHz, mono, 16-bit
                SetDlgItemText(IDC_DWFORMATS, "11.025 kHz, mono, 16-bit");
        }
        else if (Ds.dsccaps.dwFlags & WAVE_FORMAT_1S08)         {
                // 11.025 kHz, stereo, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "11.025 kHz, stereo, 8-bit");
        }
        else    {
                // 11.025 kHz, mono, 8-bit
                SetDlgItemText(IDC_DWFORMATS, "11.025 kHz, mono, 8-bit");
        }

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

3.2.2 Dialog_CapsReproduccion

Dialog_CapsReproduccion.h

```
#if !defined(AFX_DIALOG_CAPSREPRODUCCION_H__31900E21_551B_11D5_871D_E3F3F332FE79
             __INCLUDED_)
#define AFX_DIALOG_CAPSREPRODUCCION_H__31900E21_551B_11D5_871D_E3F3F332FE79__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_CapsReproduccion.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsReproduccion dialog

class CDialog_CapsReproduccion : public CDialog
{
// Construction
```

```
public:
        CDialog_CapsReproduccion(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialog_CapsReproduccion)
        enum { IDD = IDD_DIALOG_CAPSREPRODUCCION };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_CapsReproduccion)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_CapsReproduccion)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_CAPSREPRODUCCION_H__31900E21_551B_11D5_871D_E3F3F332FE79
                   __INCLUDED_)
```

Dialog_CapsReproduccion.cpp

```
// Dialog_CapsReproduccion.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_CapsReproduccion.h"
#include "Global.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

extern CDirectS Ds;

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsReproduccion dialog


CDialog_CapsReproduccion::CDialog_CapsReproduccion(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_CapsReproduccion::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_CapsReproduccion)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CDialog_CapsReproduccion::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_CapsReproduccion)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_CapsReproduccion, CDialog)
	//{{AFX_MSG_MAP(CDialog_CapsReproduccion)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_CapsReproduccion message handlers

BOOL CDialog_CapsReproduccion::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Inicializacion de los datos a mostrar por el dialogo
	SetDlgItemInt(IDC_DWMINSECONDARYSAMPLERATE, Ds.dscaps.dwMinSecondarySampleRate,
		false);
	SetDlgItemInt(IDC_DWMAXSECONDARYSAMPLERATE, Ds.dscaps.dwMaxSecondarySampleRate,
		false);
	SetDlgItemInt(IDC_DWMAXHWMIXINGALLBUFFERS, Ds.dscaps.dwMaxHwMixingAllBuffers,
		false);
	SetDlgItemInt(IDC_DWMAXHWMIXINGSTATICBUFFERS,Ds.dscaps.dwMaxHwMixingStaticBuffers,
		false);
```

```
        SetDlgItemInt(IDC_DWMAXHWMIXINGSTREAMINGBUFFERS, Ds.dscaps.
                      dwMaxHwMixingStreamingBuffers, false);

        SetDlgItemInt(IDC_DWMAXHW3DALLBUFFERS, Ds.dscaps.dwMaxHw3DAllBuffers,
                false);
        SetDlgItemInt(IDC_DWMAXHW3DSTATICBUFFERS, Ds.dscaps.dwMaxHw3DStaticBuffers,
                false);
        SetDlgItemInt(IDC_DWMAXHW3DSTREAMINGBUFFERS, Ds.dscaps.dwMaxHw3DStreamingBuffers,
                false);

        SetDlgItemInt(IDC_DWTOTALHWMEMBYTES, Ds.dscaps.dwTotalHwMemBytes,
                false);

        // Analisis de la variable dwFlags, hay que testearla bit a bit
        if (Ds.dscaps.dwFlags & DSCAPS_CERTIFIED)
                SetDlgItemText(IDC_CERTIFIED, "Sí");
        else
                SetDlgItemText(IDC_CERTIFIED, "No");
        if (Ds.dscaps.dwFlags & DSCAPS_CONTINUOUSRATE)
                SetDlgItemText(IDC_CONTINOUSRATE, "Sí");
        else
                SetDlgItemText(IDC_CONTINOUSRATE, "No");
        if (Ds.dscaps.dwFlags & DSCAPS_EMULDRIVER)
                SetDlgItemText(IDC_EMULDRIVER, "Sí");
        else
                SetDlgItemText(IDC_EMULDRIVER, "No");
        if (Ds.dscaps.dwFlags & DSCAPS_PRIMARY16BIT)
                SetDlgItemText(IDC_16BITS, "Sí");
        else
                SetDlgItemText(IDC_16BITS, "No");
        if (Ds.dscaps.dwFlags & DSCAPS_PRIMARYSTEREO)
                SetDlgItemText(IDC_STEREO, "Sí");
        else
                SetDlgItemText(IDC_STEREO, "No");

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

3.2.3 Dialog_Dispositivo

Dialog_Dispositivo.h

```
#if !defined(AFX_DIALOG_DISPOSITIVO_H__5750D1A1_54F6_11D5_871D_D84C9CCEE079__INCLUDED_)
#define AFX_DIALOG_DISPOSITIVO_H__5750D1A1_54F6_11D5_871D_D84C9CCEE079__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
```

```
#endif // _MSC_VER >= 1000
// Dialog_Dispositivo.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Dispositivo dialog

class CDialog_Dispositivo : public CDialog
{
// Construction
public:
	CDialog_Dispositivo(CWnd* pParent = NULL);	// standard constructor
// Dialog Data
	//{{AFX_DATA(CDialog_Dispositivo)
	enum { IDD = IDD_DIALOG_DISPOSITIVO };
	CString m_DispRepro;
	CString m_DispCaptura;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDialog_Dispositivo)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CDialog_Dispositivo)
	virtual BOOL OnInitDialog();
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_DISPOSITIVO_H__5750D1A1_54F6_11D5_871D_D84C9CCEE079
		__INCLUDED_)
```

Dialog_Dispositivo.cpp

```
// Dialog_Dispositivo.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Dispositivo.h"
#include "DirectS.h"

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

extern CDirectS Ds;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Dispositivo dialog

CDialog_Dispositivo::CDialog_Dispositivo(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Dispositivo::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Dispositivo)
	m_DispRepro = _T("");
	m_DispCaptura = _T("");
	//}}AFX_DATA_INIT
}


void CDialog_Dispositivo::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Dispositivo)
	DDX_CBString(pDX, IDC_ENUMDS, m_DispRepro);
	DDV_MaxChars(pDX, m_DispRepro, 60);
	DDX_CBString(pDX, IDC_ENUMDSC, m_DispCaptura);
	DDV_MaxChars(pDX, m_DispCaptura, 60);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Dispositivo, CDialog)
	//{{AFX_MSG_MAP(CDialog_Dispositivo)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CDialog_Dispositivo message handlers

BOOL CDialog_Dispositivo::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Inicializacion con datos dinámicos antes de que se muestre en pantalla
	CString str;
	PDSDEVICEDESC lpdsddtemp;
	int nItem;

	CComboBox* pCombo=(CComboBox*) GetDlgItem(IDC_ENUMDS);
	// Rastrear la lista de enumeracion de dispositivos
	lpdsddtemp=Ds.lpdsddOut;
	do	{
		nItem=pCombo->AddString(lpdsddtemp->lpDeviceDesc);
		lpdsddtemp=lpdsddtemp->pNext;
	} while (lpdsddtemp != NULL);

	// Para el COmboBox de captura
	pCombo=(CComboBox*) GetDlgItem(IDC_ENUMDSC);
	// Rastrear la lista de enumeracion de dispositivos
	lpdsddtemp=Ds.lpdsddIn;
	do	{
		nItem=pCombo->AddString(lpdsddtemp->lpDeviceDesc);
		lpdsddtemp=lpdsddtemp->pNext;
	} while (lpdsddtemp != NULL);

	return TRUE;  // return TRUE unless you set the focus to a control
		      // EXCEPTION: OCX Property Pages should return FALSE
}

void CDialog_Dispositivo::OnOK()
{
	// Para saber si el usuario ha elegido dispositivo, sino no se puede seguir
	g_DispositivoOk=true;

	CDialog::OnOK();

	// Puede haber cambiado el dispositivo luego reiniciar los objetos
	Ds.Liberar(true);
	Ds.CrearObjetos();
}
```

3.2.4 Dialog_Error

Dialog_Error.h

```
#if !defined(AFX_DIALOG_ERROR_H__104E5535_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_DIALOG_ERROR_H__104E5535_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Error.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error dialog

class CDialog_Error : public CDialog
{
// Construction
public:
        CDialog_Error(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialog_Error)
        enum { IDD = IDD_DIALOG_ERROR };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Error)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_Error)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
```

```
// previous line.

#endif // !defined(AFX_DIALOG_ERROR_H__104E5535_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

Dialog_Error.cpp

```
// Dialog_Error.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Error.h"

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

int g_Error;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error dialog


CDialog_Error::CDialog_Error(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Error::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Error)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CDialog_Error::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Error)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Error, CDialog)
	//{{AFX_MSG_MAP(CDialog_Error)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CDialog_Error message handlers

BOOL CDialog_Error::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Gestion de los códigos de Error
        CString str;

        switch (g_Error)        {
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
                str="Fallo en DirectDraw";
                break;
        case 30:
        case 31:
        case 32:
        case 33:
        case 34:
        case 35:
        case 36:
        case 37:
        case 38:
        case 39:
        case 40:
        case 41:
                str="Fallo en DirectSound";
                break;
        case 70:
                str="Error al introducir datos";
                break;
        case 90:
                str="No se puede acceder al disco";
                break;
        case 91:
                str="No existe el archivo solicitado";
                break;
        case 100:
                str="Tienes unas DirectX antiguas, debes actualizar a versión 6.0 o
                    superior Comprueba ejecutando DxDiag";
                break;
        default:
```

```
                str="Esto no debería salir nunca";   // Para tenerlo todo bajo control
                break;
        }

        SetDlgItemText(IDC_STATIC_ERROR, str);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

void CDialog_Error::OnOK()
{
        // Solo se sale de la aplicación si no existe la versión correcta de DirectX
        if (g_Error == 100)
                // Si hay una versión antigua de DirectX, hay que salir automáticamente
                PostMessage(WM_CLOSE, 0 ,0 );

        CDialog::OnOK();
}
```

3.2.5 Dialog_Filtrado

Dialog_Filtrado.h

```
#if !defined(AFX_DIALOG_FILTRADO_H__89DF5607_534D_11D5_871D_F662CFBE2B7A__INCLUDED_)
#define AFX_DIALOG_FILTRADO_H__89DF5607_534D_11D5_871D_F662CFBE2B7A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Filtrado.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Filtrado dialog

#include "Dialog_Personal.h"

class CDialog_Filtrado : public CDialog
{
// Construction
public:
        CDialog_Filtrado(CWnd* pParent = NULL);   // standard constructor
        Inicializacion();
        void Serialize(CFile *File, bool Salvar, bool Configuracion);
        int Configurar();
// Dialog Data
        //{{AFX_DATA(CDialog_Filtrado)
```

```
        enum { IDD = IDD_DIALOG_FILTRADO };
        int             m_Cortes;
        int             m_Gcir;
        int             m_Disco;
        int             m_Escala;
        int             m_Personalizar;
        BOOL    m_BandasStereo;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Filtrado)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
public:
        int *m_frecuencias, *m_Agrupacion, *m_Frec;             // Cálculos
        int m_Cont;
        CDialog_Personal Dlg_p;         // Para pesonalizar los cortes
protected:
        bool m_Filas;           // Filas de potenciómetros
        // Generated message map functions
        //{{AFX_MSG(CDialog_Filtrado)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_FILTRADO_H__89DF5607_534D_11D5_871D_F662CFBE2B7A__INCLUDED_)
```

Dialog_Filtrado.cpp

```
// Dialog_Filtrado.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Filtrado.h"
#include "Global.h"
```

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

// Filtro para la selección de archivos
static char g_FiltroEcu[]="Archivos Ecu (*.ecu) | *.ecu||";

/////////////////////////////////////////////////////////////////////////////
// CDialog_Filtrado dialog

CDialog_Filtrado::CDialog_Filtrado(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Filtrado::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Filtrado)
	m_Cortes = 0;
	m_Disco = FALSE;
	m_Personalizar = FALSE;
	m_Gcir = 0;
	m_Escala = -1;
	m_BandasStereo = FALSE;
	//}}AFX_DATA_INIT
}


void CDialog_Filtrado::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Filtrado)
	DDX_Text(pDX, IDC_CORTES, m_Cortes);
	DDV_MinMaxInt(pDX, m_Cortes, 3, 1000);
	DDX_Check(pDX, IDC_DISCO, m_Disco);
	DDX_Check(pDX, IDC_PERSONAL, m_Personalizar);
	DDX_Text(pDX, IDC_GCIR, m_Gcir);
	DDV_MinMaxInt(pDX, m_Gcir, 1, 10);
	DDX_Radio(pDX, IDC_LIN, m_Escala);
	DDX_Check(pDX, IDC_STEREO, m_BandasStereo);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Filtrado, CDialog)
	//{{AFX_MSG_MAP(CDialog_Filtrado)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CDialog_Filtrado message handlers
CDialog_Filtrado::Inicializacion()
{
        // Coloca unos valores iniciales
        m_Cortes=4;
        m_Gcir=3;
        m_BandasStereo=false;
        m_Disco=false;
        m_Personalizar=false;
        m_Escala=true;

        DoModal();
        return 0;
}

void CDialog_Filtrado::OnOK()
{
        // No se intercambian los datos antes de la llamada a la clase Base

        int result;

        CDialog::OnOK();

        // Al pulsar Ok se procede a asignar los valores del dialogo a las variables
        // miembro con lo que se pueden tomar acciones antes de cerralo definitivamente

        // Se configura como si se fuera a ejecutar en este programa, se configuran
        // los arrays, pero solo se pasan al archivo "FILTRADO.CFG" los que poseen
        // informacion que solo se pueda calcualr aqui, el resto se calcula en
        // filtrado.exe

        result=Configurar();
        if (result != 0)        {
                g_Error=70;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
        }

        return;
}

int CDialog_Filtrado::Configurar()
{
        int Frecuencia_max, Frecuencia_min;
        int Cortes;
        bool Escala, Personalizar, Disco;

        int j,i,k;                              // Cuentas varias
        float tempf, fmin, fmax;                // Almacena resultados intermedios
```

```
int temp, h_min, Fs, CortesMax;
int *PuntosFFT;
char cadena[10];
CString str;
CFile   File;                   // Objeto que accede a disco

// Operaciones para acondicionar todo lo necesario para mostrar por
// pantalla los controles y hacer las cuentas pertinentes con la FFT.
// Una vez solventado los posibles errores en los parámetros de entrada se
// procede a los cálculos de para las bandas en f

Cortes=m_Cortes;
Personalizar=m_Personalizar ? true : false;
Escala=m_Escala ? true : false;
Disco=m_Disco ? true : false;

// Calculos que hay que realizar siempre
// Comprobacion de la entrada de datos
if (Dlg_m.m_Fs.IsEmpty())
        Dlg_m.m_Fs="44100 Hz";

for (i=0; i<Dlg_m.m_Fs.GetLength() && Dlg_m.m_Fs[i] !=' ' ; i++)        {
        cadena[i]=Dlg_m.m_Fs[i];
}
cadena[i]='\0';
Fs=atoi(cadena);

// Discretizar los valores posibles de Frecuencias de muestreo a los 4
// que están en el dialogo: 44100, 22050, 11025, 8820
if (Fs > 44100)
        Fs=44100;
else if ((Fs<44100) && (Fs>22050))      {
        if ((44100-Fs)<(Fs-22050))
                Fs=44100;
        else
                Fs=22050;
}
else if ((Fs<22050) && (Fs>11025))      {
        if ((22050-Fs)<(Fs-11025))
                Fs=22050;
        else
                Fs=11025;
}
else if ((Fs<11025) && (Fs>8820))       {
        if ((11025-Fs)<(Fs-8820))
                Fs=11025;
        else
                Fs=8820;
```

```
}
if (Fs<8820)
        Fs=8820;

// Comprobacion de que los datos introducidos por el usuario son correctos

h_min=Fs/PUNTOSFFT;     // Espaciado mínimo en frecuencia, depende de
                        // Frecuencia de muestreo y puntos de la FFT
if (Fs == 44100)
        temp=20000;
else
        temp=Fs>>1;     // Cálculo del Bw analizable
// Ajuste de los límites de las frecuencais Max y Min
Frecuencia_max=temp;
Frecuencia_min=20;      // Simpre, la superior varia en funcia de Fs

CortesMax=temp/h_min;   // Todo el espectro

// Máximo N de cortes que se pueden representar según el espaciado mínimo

temp=(Frecuencia_max-Frecuencia_min)/Cortes;    // Espaciado mínimo pedido
// Se ajusta a lo máximo que se puede
if (Cortes >= CortesMax)
        m_Cortes=CortesMax;
else if (temp < h_min)
        m_Cortes=(Frecuencia_max-Frecuencia_min)/h_min;
else
        m_Cortes=Cortes;

PuntosFFT=(int *) malloc(PUNTOSFFT * sizeof (int));

for (i=0; i<CortesMax; i++)
        *(PuntosFFT+i)=h_min*(i+1);     // Frecuencias que obtiene la FFT
// No tiene encuenta la DC

// Estructura que gestiona el reparto de las bandas según la selección del
// usuario

if (Disco)              {
        // Para abrir del disco una configuracion personal previamente introducida
        // y revisada
        CFileDialog dlg(true, "ecu", "*.ecu", OFN_HIDEREADONLY | OFN_NOCHANGEDIR,
                g_FiltroEcu, NULL);
        if (dlg.DoModal() == IDOK)      {
                CFile Arch;
                str=dlg.GetPathName();
                if (!Arch.Open(str, CFile::modeRead))   {
                        g_Error=91;
```

```
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR,
                              0, 0);
                return -1;
        }
        Arch.Close();
        // Lectura de los datos del archivo .ecu en cuestión
        if (!File.Open(str, CFile::modeRead))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                Dlg_e.DoModal();
                return 1;                       // Salir y seguir
        }
        else    {
                // Lectura del archivo
                Serialize(&File, false, false);
                File.Close();
        }
}
// Si se lee del Disco no hace falta nada mas

m_Agrupacion=(int *) malloc(m_Cortes * sizeof(int));
m_Frec=(int *) malloc(m_Cortes * sizeof(int));

for (i=0; i<m_Cortes; i++)
        *(m_Frec+i)=*(m_frecuencias+i);

// Hay que agrupar los puntos que calcula FFT con los cortes que controla
// el usuario
j=0;
for (i=0; i<m_Cortes-1; i++)    {
        temp=(*(m_Frec+i) + *(m_Frec+(i+1)))>>1;
        j++;
        while (temp > *(PuntosFFT+j))
        j++;
        *(m_Agrupacion+i)=j;
}
*(m_Agrupacion+i)=(PUNTOSFFT/2);     // Para el filtrado pasa todo
// El punto más alto es PUNTOSFFT/2 porque hasta aqui llega la parte real,
// la parte imaginaria es desde PUNTOSFFT/2 hasta PUNTOSFFT, que valesn
// las mismas constantes para alterar el contenido espectral

// Comprobar si hay que cambiar la frecuencia de muestreo por la
// configuracion del ecualizador
if (Dlg_m.m_Fs.IsEmpty())
        Dlg_m.m_Fs="44100 Hz";

for (i=0; i<Dlg_m.m_Fs.GetLength() && Dlg_m.m_Fs[i] !=' ' ; i++)  {
        cadena[i]=Dlg_m.m_Fs[i];
```

```
        }
        cadena[i]='\0';
        temp=atoi(cadena);
        if (temp != Fs)         {
                // Hay que grabar a disco la nueva Fs
                Dlg_m.Grabar();
        }
}
else if (Personalizar)  {
        // Lanza los dialogos adicionales para recoger las frecuencias de las
        // bandas

        m_frecuencias=(int *) malloc(m_Cortes * sizeof(int));
        for (m_Cont=0; m_Cont<m_Cortes; m_Cont++)       {
                Dlg_p.DoModal();
                *(m_frecuencias+m_Cont)=Dlg_p.m_frec;
                if (Dlg_p.m_Cancel)
                        return -1;              // -1 Salir pero no error
        }
        // Ya ha introducido el usuario que bandas quiere controlar, asi que
        // solo hay que revisar que cumplen con las restricciones de espaciado
        // entre bandas contiguas

        // Los cálculos preliminares han obtenido el espaciado mínimo y tambien
        // los cortes máximos que se pueden tener, asi que hay que contrastarlos
        // con la entrada del usuario

        // Busqueda de frecuencias repetidas
        k=0;
        for (i=0; i<m_Cortes; i++)      {
                temp=*(m_frecuencias+i);
                for (j=i+1; j<m_Cortes; j++)    {
                        if (*(m_frecuencias+j) == temp) {
                                *(m_frecuencias+j)=*(m_frecuencias+m_Cortes-k);
                                k++;
                        }
                }
        }
        m_Cortes-=k;

        // Ordenacion de menor a mayor (por si a caso)
        for (i=0; i<m_Cortes; i++)      {
                for(j=i+1; j<m_Cortes ;j++)             {
                        if (*(m_frecuencias+i) > *(m_frecuencias+j))    {
                                temp=*(m_frecuencias+i);
                                *(m_frecuencias+i)=*(m_frecuencias+j);
                                *(m_frecuencias+j)=temp;
                        }
```

```
        }
}
// Busca si hay bandas por encima de la Frecuencia de Nyquist o 20000Hz
for (i=0; i<m_Cortes; i++)      {
        if (*(m_frecuencias+i) > Frecuencia_max)
                break;
}
// Por si hay menos de 3 bandas se aborta la ejecución
if (m_Cortes < 3)
        return -1;

// Copia de las frecuencias para el filtrado, i tiene la posicion más alta
j=1;
m_Frec=(int *) malloc(i * sizeof(int));
*m_Frec=*(m_frecuencias);
for (k=1; k<i; k++)     {
        if ((*(m_frecuencias+k))-(*(m_Frec+k-1)) > h_min)       {
                *(m_Frec+j)=*(m_frecuencias+k);
                *(m_frecuencias+j)=*(m_Frec+j);// Para salvar a disco
                j++;
        }
}
m_Cortes=j;

// Una vez comprobados los datos se copian al objeto Dialog_Filtrado, para
// su posterior intercambio con el disco, luego han de ser correctos

m_Agrupacion=(int *) malloc(m_Cortes * sizeof(int));

// Hay que agrupar los puntos que calcula FFT con los cortes que controla
// el usuario
j=0;
for (i=0; i<m_Cortes-1; i++)    {
        temp=(*(m_Frec+i) + *(m_Frec+(i+1)))>>1;
        j++;
        while (temp > *(PuntosFFT+j))
        j++;
        *(m_Agrupacion+i)=j;
}
*(m_Agrupacion+i)=(PUNTOSFFT/2);        // Para el filtrado pasa todo
// El punto más alto es PUNTOSFFT/2 porque hasta aqui llega la parte real,
// la parte imaginaria es desde PUNTOSFFT/2 hasta PUNTOSFFT, que valen
// las mismas constantes para alterar el contenido espectral

// Preguntar al usuario si quiere salvar la configuracion del filtro
// actual, bueno si se emulan ecualizadores reales

CFileDialog dlg_p(false, "ecu", "*.ecu", OFN_HIDEREADONLY |
```

```
                OFN_NOREADONLYRETURN | OFN_OVERWRITEPROMPT | OFN_NOCHANGEDIR,
                g_FiltroEcu, NULL);
        if (dlg_p.DoModal() == IDOK)    {
                // Al pulsar OK se coge el nombre del archivo y se procede a
                // escribir la configuracion del ecualizador
                str=dlg_p.GetPathName();

                if (!File.Open(str, CFile::modeCreate | CFile::modeWrite))      {
                        TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                        g_Error=90;
                        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR,
                                        0, 0);
                        return -1;
                }
                else    {
                        // Se crea o machaca si existía el archivo
                        Serialize(&File, true, false);
                        File.Close();
                }
        }
}
else    {
        // Hay que calcular las bandas automáticamente
        // Desde aquí se inicializan todos los arrays que se asignarán
        // dinámicamente, según lo que pida el usuario

        m_Agrupacion=(int *) malloc(m_Cortes * sizeof(int));
        m_Frec=(int *) malloc(m_Cortes * sizeof(int));

        // Ajuste de las frecuencias bajas, para comenzar simepre en 20 Hz, aunque
        // el espaciado sea menor, si Frecuencia de Muestreo es baja, es el caso
        j=0;
        while (h_min*(j+1) < 20)
                j++;
        k=0;
        if (m_Cortes != CortesMax)      {
        // Tener encuenta una escala lineal de menos puntos o mayor espaciado
        // que lo máximo permitido (Tramos de frecuencias)
                if (Escala)     {
                        // Implementa una escala Log
                        fmin=(float)log(Frecuencia_min);
                        fmax=(float)log(Frecuencia_max);
                        tempf=(fmax-fmin)/(Cortes-1);
                        for (i=0; i<m_Cortes-1; i++)            {
                                temp=(int) exp(fmin+(tempf*i));
                                if (temp >= *(PuntosFFT+j))     {
                                        *(m_Frec+k)=temp;
                                        k++;
```

```
                                        j++;
                                }
                        }
                        *(m_Frec+k)=Frecuencia_max;
// Quitar cortes con espaciado menor que h_min Hz, minimo que se calcula; estos
// se dan en f bajas por la escala Log
                        m_Cortes=k+1;
                }
                else    {
                        // Escala Lineal
                        temp=(Frecuencia_max-Frecuencia_min)/m_Cortes;
                        for (i=0; i<m_Cortes-1; i++)    {
                                *(m_Frec+i)=Frecuencia_min+(temp*i);
                        }
                        *(m_Frec+i)=Frecuencia_max;
                }

        // Hay que agrupar los puntos que calcula FFT con los cortes que controla
        // el usuario
                j=0;
                for (i=0; i<m_Cortes-1; i++)    {
                        temp=(*(m_Frec+i) + *(m_Frec+(i+1)))>>1;
                        j++;
                        while (temp > *(PuntosFFT+j))
                        j++;
                        *(m_Agrupacion+i)=j;
                }
                *(m_Agrupacion+i)=(PUNTOSFFT/2);   // Para el filtrado pasa todo
        // El punto más alto es PUNTOSFFT/2 porque hasta aqui llega la parte real,
        // la parte imaginaria es desde PUNTOSFFT/2 hasta PUNTOSFFT, que valen
        // las mismas constantes para alterar el contenido espectral
        }
        // El último se rellena con el último m_CortesMax
        else    {
        // No se puede colocar una escala log si se quiere mostrar los componentes
        // tal cual salen de la FFT porque eston están repartidos linealmente
        // sobre el dominio de la f
                for (i=0; i<CortesMax; i++)
                        *(m_Frec+i)=h_min*(i+1);    // Frecuencias de la FFT
        }
}

// Para saber si hay 1 o 2 filas de deslizadores, lo almacena m_Filas
// 0=>1 Fila y 1=> 2 Filas
m_Filas=Dlg_m.m_Modo && !m_BandasStereo;
m_BandasStereo=(Dlg_m.m_Modo == false) ? false : m_BandasStereo;

// Salvar los arrays útiles calculados y preparados para el filtrado, son:
```

```
        // m_Frec y m_Agrupacion

        if (!File.Open("FILTRADO.CFG", CFile::modeCreate | CFile::modeWrite))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return -1;
        }
        else    {
                // No existe y se crea el archivo por primera vez
                Serialize(&File, false, true);
                File.Close();
        }

        // Liberar los arrays utilizados...
        free(m_Agrupacion);
        free(m_Frec);
        free(PuntosFFT);

        return 0;
}

void CDialog_Filtrado::Serialize(CFile *File, bool Salvar, bool Configuracion)
{
        // Para Guardar o Cargar del disco el contenido de algunas variables miembro

        int i;

        // Salvar=true Salva los datos al disco y Salvar=false carga datos del disco
        // Configuracion distingue entre los datos necesarios para ejecutar filtrado
        // y las configuraciones personalizadas del usuario
        if (!Configuracion)     {
                if (Salvar)     {
                        // Se escribe en el disco...
                        CArchive Archivo(File, CArchive::store, 512);
                        CObject::Serialize(Archivo);
                        if (Archivo.IsStoring())        {
                                Archivo << m_Cortes << m_Gcir << m_BandasStereo;
                                for (int i=0; i<m_Cortes; i++)
                                        Archivo << *(m_frecuencias+i);
                                Archivo << Dlg_m.m_Fs << Dlg_m.m_Modo;
                        }
                }
                else    {
                        // Se lee del disco...
                        CArchive Archivo(File, CArchive::load, 512);
                        CObject::Serialize(Archivo);     // Llamada a la clase base
                        if (!Archivo.IsStoring())       {
```

```
                                Archivo >> m_Cortes >> m_Gcir >> m_BandasStereo;
                                m_frecuencias=(int *) malloc(m_Cortes * sizeof(int));
                                for (i=0; i<m_Cortes; i++)
                                        Archivo >> *(m_frecuencias+i);
                                Archivo >> Dlg_m.m_Fs >> Dlg_m.m_Modo;
                        }
                }
        }
        else    {
                // Para escribir en el disco el archivo "FILTRADO.CFG"
                CArchive Archivo(File, CArchive::store, 512);
                CObject::Serialize(Archivo);
                if (Archivo.IsStoring())        {
                        Archivo << m_Cortes << m_Filas;
                        for (int i=0; i<m_Cortes; i++)
                                Archivo << *(m_Frec+i);
                        for (i=0; i<m_Cortes; i++)
                                Archivo << *(m_Agrupacion+i);
                        Archivo << m_BandasStereo << m_Gcir;
                }
        }
        return;
}
```

3.2.6 Dialog_Graficos

Dialog_Graficos.h

```
#if !defined(AFX_DIALOG_GRAFICOS_H__104E5534_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_DIALOG_GRAFICOS_H__104E5534_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Graficos.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Graficos dialog

class CDialog_Graficos : public CDialog
{
// Construction
public:
        CDialog_Graficos(CWnd* pParent = NULL);   // standard constructor
        void Inicializacion();
// Dialog Data
        //{{AFX_DATA(CDialog_Graficos)
        enum { IDD = IDD_DIALOG_GRAFICOS };
```

```
        int             m_Cortes;
        BOOL    m_Escala;
        int             m_Escalones;
        int             m_Escalones_max;
        int             m_Escalones_min;
        int             m_Frecuencia_max;
        int             m_Frecuencia_min;
        BOOL    m_RuidoRosa;
        BOOL    m_Archivo;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Graficos)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
        void Serialize(CFile *File, bool Salvar);

// Implementation
protected:
        DialogoAbrir();
        // Generated message map functions
        //{{AFX_MSG(CDialog_Graficos)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
public:
        CString m_NArchivo;                     // Sabe el nombre del archivo


};

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

#endif // !defined(AFX_DIALOG_GRAFICOS_H__104E5534_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

Dialog_Graficos.cpp

```
// Dialog_Graficos.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Graficos.h"
```

```
#include "Global.h"

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

extern CDirectS Ds;

// Filtro para el dialogo de selección de archivo
static char g_FiltroArch[]="Archivos Wav (*.wav) | *.wav||";

/////////////////////////////////////////////////////////////////////////////
// CDialog_Graficos dialog


CDialog_Graficos::CDialog_Graficos(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Graficos::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Graficos)
	m_Cortes = 0;
	m_Escala = FALSE;
	m_Escalones = 0;
	m_Escalones_max = 0;
	m_Escalones_min = 0;
	m_Frecuencia_max = 0;
	m_Frecuencia_min = 0;
	m_RuidoRosa = FALSE;
	m_Archivo = FALSE;
	//}}AFX_DATA_INIT
}


void CDialog_Graficos::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Graficos)
	DDX_Text(pDX, IDC_CORTES, m_Cortes);
	DDV_MinMaxInt(pDX, m_Cortes, 3, 1000);
	DDX_Check(pDX, IDC_ESCALA, m_Escala);
	DDX_Text(pDX, IDC_ESCALONES, m_Escalones);
	DDV_MinMaxInt(pDX, m_Escalones, 3, 99);
	DDX_Text(pDX, IDC_ESCALONES_MAX, m_Escalones_max);
	DDV_MinMaxInt(pDX, m_Escalones_max, -99, 0);
	DDX_Text(pDX, IDC_ESCALONES_MIN, m_Escalones_min);
	DDV_MinMaxInt(pDX, m_Escalones_min, -99, 0);
	DDX_Text(pDX, IDC_FRECUENCIA_MAX, m_Frecuencia_max);
```

```
        DDV_MinMaxInt(pDX, m_Frecuencia_max, 20, 20000);
        DDX_Text(pDX, IDC_FRECUENCIA_MIN, m_Frecuencia_min);
        DDV_MinMaxInt(pDX, m_Frecuencia_min, 20, 20000);
        DDX_Check(pDX, IDC_RUIDOROSA, m_RuidoRosa);
        DDX_Check(pDX, IDC_ARCHIVO, m_Archivo);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Graficos, CDialog)
        //{{AFX_MSG_MAP(CDialog_Graficos)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Graficos message handlers
void CDialog_Graficos::Inicializacion()
{

        // Comprueba la existencia del archivo "OPCIONES.CFG", antes de mostrar
        // el dialogo, con los valores que allí estén y si no existen una
        // configuración inicial

        CFile File;

        if (!File.Open("opciones.cfg", CFile::modeRead))        {
                if (!File.Open("OPCIONES.CFG", CFile::modeCreate | CFile::modeWrite))   {
                        TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                        g_Error=90;
                        Dlg_e.DoModal();
                }
                else    {
                        // Se creara el archivo opciones.cfg por primera vez
                        Dlg_m.m_Fs="44100 Hz";
                        Dlg_m.m_Bits=1;
                        Dlg_m.m_Modo=1;

                        m_Cortes=40;
                        m_Escala=0;
                        m_Escalones=20;
                        m_Escalones_max=-10;
                        m_Escalones_min=-60;
                        m_Frecuencia_max=20000;
                        m_Frecuencia_min=20;

                        m_Archivo=false;
                        m_RuidoRosa=false;
                        m_NArchivo.Empty();
```

```
                        Serialize(&File, true);
                        File.Close();                   // Cierra el fichero
                }
        }
        else    {
                // Existe un archivo de configuracion => se lee
                Serialize(&File, false);
                File.Close();
        }

        // Pone el dialogo en pantalla
        DoModal();
        if (g_Error != 0)       {
                // Restauración de valores por defecto para evitar errores
                m_Archivo=false;
                m_RuidoRosa=false;
                m_NArchivo.Empty();
        }
        return;
}

void CDialog_Graficos::OnOK()
{
        // Para guardar los cambios hechos en el dialogo

        CFile File;
        CString str;
        int resultado;

        CDialog::OnOK();

        // Se pasa la orden de aceptar al armazón para que intercambie los datos a las
        // variables y se procede a tratar los resultados.

        if (m_RuidoRosa)        {
                // tiene preferencia el RuidoRosa por estar el primero en la estructura
                // de selcción
                m_Archivo=false;
        }
        else if (m_Archivo)     {
                // Se lanza el dialogo de Güindous para que el usuario de el nombre del
                // archivo a leer, se intenta abrir, que es lo único que se puede hacer,
                // si está corrupto no es posible saberlo hasta que no se lea

                resultado=DialogoAbrir();
                m_RuidoRosa=false;
```

```
        }
        if (!resultado)         {
                // Se ha cancelado desde el dialogo de selección de archivo
                g_Error=70;
                Dlg_e.DoModal();
                return;
        }
        if (!File.Open("OPCIONES.CFG", CFile::modeCreate | CFile::modeWrite))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                Dlg_e.DoModal();
        }
        else    {
                Serialize(&File, true);
                File.Close();
        }

        return;
}

int CDialog_Graficos::DialogoAbrir()
{
        char *temp;

        CFileDialog dlg(true, "wav", "*.wav", OFN_HIDEREADONLY | OFN_NOCHANGEDIR,
                g_FiltroArch, NULL);
        if (dlg.DoModal() == IDOK)      {
                CFile Arch;
                m_NArchivo=dlg.GetPathName();
                temp=(char *) malloc((m_NArchivo.GetLength()+1) * sizeof(char));
                _tcscpy(temp, m_NArchivo);
                if (!Arch.Open(temp, CFile::modeRead))  {
                        g_Error=91;
                        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                        free (temp);
                        return -1;
                }
                Arch.Close();
                free (temp);
                return 1;
        }

        return 0;
}

void CDialog_Graficos::Serialize(CFile *File, bool Salvar)
{
        // Gestion de la escritura o lectura al disco
```

```
int i;

if (Salvar)     {
        CArchive Archivo(File, CArchive::store, 512);
        CObject::Serialize(Archivo);            // Llamada a la clase base
        if (Archivo.IsStoring())        {
                Archivo << Dlg_m.m_Fs << Dlg_m.m_Bits << Dlg_m.m_Modo;
                // Los descriptores de los dispositivos
                Archivo << Ds.guidDs.Data1 << Ds.guidDs.Data2 << Ds.guidDs.Data3;
                for (i=0; i<8; i++)
                        Archivo << Ds.guidDs.Data4[i];
                Archivo << Ds.guidDsC.Data1 << Ds.guidDsC.Data2
                        << Ds.guidDsC.Data3;
                for (i=0; i<8; i++)
                        Archivo << Ds.guidDsC.Data4[i];

                Archivo << m_Cortes << m_Escalones << m_Escalones_max;
                Archivo << m_Escalones_min << m_Frecuencia_max;
                Archivo << m_Frecuencia_min << m_Escala;

                // Opciones adicionales
                Archivo << m_RuidoRosa << m_Archivo;
                if (m_Archivo)
                        // Almacenar el nombre y la ruta del archivo a reproduccir
                        Archivo << m_NArchivo;
        }
}
else {
        CArchive Archivo(File, CArchive::load, 512);
        CObject::Serialize(Archivo);            // Llamada a la clase base
        if (!Archivo.IsStoring())       {
                Archivo >> Dlg_m.m_Fs >> Dlg_m.m_Bits >> Dlg_m.m_Modo;
                // Meter los descriptores de dispositivo
                Archivo >> Ds.guidDs.Data1 >> Ds.guidDs.Data2 >> Ds.guidDs.Data3;
                for (i=0; i<8; i++)
                        Archivo >> Ds.guidDs.Data4[i];
                Archivo >> Ds.guidDsC.Data1 >> Ds.guidDsC.Data2
                        >> Ds.guidDsC.Data3;
                for (i=0; i<8; i++)
                        Archivo >> Ds.guidDsC.Data4[i];

                Archivo >> m_Cortes >> m_Escalones >> m_Escalones_max;
                Archivo >> m_Escalones_min >> m_Frecuencia_max;
                Archivo >> m_Frecuencia_min >> m_Escala;

                // Opciones adidionales, no se leen del disco se resetean
                m_RuidoRosa=m_Archivo=false;
```

```
                }
        }
}
```

3.2.7 Dialog_Muestreo

Dialog_Muestreo.h

```
#if !defined(AFX_DIALOG_MUESTREO_H__104E5533_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_DIALOG_MUESTREO_H__104E5533_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Muestreo.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Muestreo dialog

#include "resource.h"

class CDialog_Muestreo : public CDialog
{
// Construction
public:
        CDialog_Muestreo(CWnd* pParent = NULL);   // standard constructor
        void Inicializacion();
        void Grabar();
// Dialog Data
        //{{AFX_DATA(CDialog_Muestreo)
        enum { IDD = IDD_DIALOG_MUESTREO };
        int             m_Bits;
        CString m_Fs;
        int             m_Modo;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Muestreo)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        void Serialize(CFile *File, bool Salvar);
        // Generated message map functions
```

```
        //{{AFX_MSG(CDialog_Muestreo)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_MUESTREO_H__104E5533_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

Dialog_Muestreo.cpp

```
// Dialog_Muestreo.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Muestreo.h"
#include "Global.h"

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

extern CDialog_Filtrado Dlg_f;
extern CDirectS Ds;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Muestreo dialog


CDialog_Muestreo::CDialog_Muestreo(CWnd* pParent /*=NULL*/)
        : CDialog(CDialog_Muestreo::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDialog_Muestreo)
        m_Bits = -1;
        m_Fs = _T("");
        m_Modo = -1;
        //}}AFX_DATA_INIT
}


void CDialog_Muestreo::DoDataExchange(CDataExchange* pDX)
```

```
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDialog_Muestreo)
        DDX_Radio(pDX, IDC_BITS, m_Bits);
        DDX_CBString(pDX, IDC_FS, m_Fs);
        DDV_MaxChars(pDX, m_Fs, 10);
        DDX_Radio(pDX, IDC_MODO, m_Modo);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Muestreo, CDialog)
        //{{AFX_MSG_MAP(CDialog_Muestreo)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Muestreo message handlers
void CDialog_Muestreo::Inicializacion()
{
        // Comprueba la existencia del archivo "OPCIONES.CFG", antes de mostrar
        // el dialogo, con los valores que allí estén y si no existen una
        // configuración inicial

        CFile File;

        if (!File.Open("OPCIONES.CFG", CFile::modeRead))        {
                if (!File.Open("OPCIONES.CFG", CFile::modeCreate | CFile::modeWrite))   {
                        TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                        g_Error=90;
                        Dlg_e.DoModal();
                }
                else    {
                        // Se creara el archivo opciones.cfg por primera vez
                        m_Fs="44100 Hz"; // 44100 quitar 22050
                        m_Bits=1;
                        m_Modo=1;

                        Dlg_g.m_Cortes=40;
                        Dlg_g.m_Escala=0;
                        Dlg_g.m_Escalones=20;
                        Dlg_g.m_Escalones_max=-10;
                        Dlg_g.m_Escalones_min=-60;
                        Dlg_g.m_Frecuencia_max=20000;
                        Dlg_g.m_Frecuencia_min=20;

                        Dlg_g.m_Archivo=false;
                        Dlg_g.m_RuidoRosa=false;
```

```
                        Dlg_g.m_NArchivo.Empty();

                        Serialize(&File, true);
                        File.Close();                   // Cierra el fichero
                }
        }
        else    {
                // Existe un archivo de configuracion => se lee
                Serialize(&File, false);
                File.Close();
        }

        // Pone el dialogo en pantalla
        DoModal();

        return;
}


void CDialog_Muestreo::OnOK()
{
        // Para almacenar los nuevos valores

        CFile File;
        RECT rc;

        CDialog::OnOK();

        if (!File.Open("OPCIONES.CFG", CFile::modeCreate | CFile::modeWrite))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                Dlg_e.DoModal();
        }
        else    {
                Serialize(&File, true);
                File.Close();
        }
        // Si cambia el modo de captura, es decir, entre mono y stereo es posible que
        // cambie m_Filas, o si modificamos la frecuencia de muestreo; por lo tanto hay
        // que obligar a recalcular los arrays de la configuracion del filtro. y
        // escribir en "FILTRADO.CFG" los nuevos valores

        if (g_primeraFiltrado)
                Dlg_f.Configurar();

        // Antes de salir refrescar el dialogo para mostrar por pantalla las opciones
        // seleccionadas de muestreo
        rc.top=233;
```

```
        rc.left=485;
        rc.bottom=288;
        rc.right=601;   // Para solo repintar este trozo, y evetar parpadeos
        CWnd* pWnd=AfxGetMainWnd();
        pWnd->InvalidateRect(&rc, false);               // Fuerza el repintado del dialogo
}



void CDialog_Muestreo::Serialize(CFile *File, bool Salvar)
{
        // Gestion de la escritura o lectura al disco

        int i;

        if (Salvar)     {
                CArchive Archivo(File, CArchive::store, 512);
                CObject::Serialize(Archivo);            // Llamada a la clase base
                if (Archivo.IsStoring())        {
                        Archivo << m_Fs << m_Bits << m_Modo;
                        // Hay que escribir la estructura por elementos separados
                        Archivo << Ds.guidDs.Data1 << Ds.guidDs.Data2 << Ds.guidDs.Data3;
                        for (i=0; i<8; i++)
                                Archivo << Ds.guidDs.Data4[i];
                        Archivo << Ds.guidDsC.Data1 << Ds.guidDsC.Data2
                                << Ds.guidDsC.Data3;
                        for (i=0; i<8; i++)
                                Archivo << Ds.guidDsC.Data4[i];

                        Archivo << Dlg_g.m_Cortes << Dlg_g.m_Escalones
                                << Dlg_g.m_Escalones_max;
                        Archivo << Dlg_g.m_Escalones_min << Dlg_g.m_Frecuencia_max;
                        Archivo << Dlg_g.m_Frecuencia_min << Dlg_g.m_Escala;

                        // Opciones del analisis adicionales
                        Archivo << Dlg_g.m_RuidoRosa << Dlg_g.m_Archivo;
                        if (Dlg_g.m_Archivo)
                                // Almacenar el nombre y la ruta del archivo a reproduccir
                                Archivo << Dlg_g.m_NArchivo;
                }
        }
        else {
                CArchive Archivo(File, CArchive::load, 512);
                CObject::Serialize(Archivo);            // Llamada a la clase base
                if (!Archivo.IsStoring())       {
                        Archivo >> m_Fs >> m_Bits >> m_Modo;
                        // Hay que rescatar la estreucturas elemento por elemento
                        Archivo >> Ds.guidDs.Data1 >> Ds.guidDs.Data2 >> Ds.guidDs.Data3;
```

```
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDs.Data4[i];
                        Archivo >> Ds.guidDsC.Data1 >> Ds.guidDsC.Data2
                                >> Ds.guidDsC.Data3;
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDsC.Data4[i];

                        Archivo >> Dlg_g.m_Cortes >> Dlg_g.m_Escalones
                                >> Dlg_g.m_Escalones_max;
                        Archivo >> Dlg_g.m_Escalones_min >> Dlg_g.m_Frecuencia_max;
                        Archivo >> Dlg_g.m_Frecuencia_min >> Dlg_g.m_Escala;

                        // Opciones adidionales, no se cargan se resetean
                        Dlg_g.m_RuidoRosa=Dlg_g.m_Archivo=false;
                }
        }
}

void CDialog_Muestreo::Grabar()
{
        // Para grabar a disco directamente...

        CFile File;

        if (!File.Open("OPCIONES.CFG", CFile::modeCreate | CFile::modeWrite))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                Dlg_e.DoModal();
        }
        else    {
                Dlg_m.Serialize(&File, true);
                File.Close();
        }
        return;
}
```

3.2.8 Dialog_Personal

Dialog_Personal.h

```
#if !defined(AFX_DIALOG_PERSONAL_H__89DF5609_534D_11D5_871D_F662CFBE2B7A__INCLUDED_)
#define AFX_DIALOG_PERSONAL_H__89DF5609_534D_11D5_871D_F662CFBE2B7A__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Personal.h : header file
//
```

```
/////////////////////////////////////////////////////////////////////////////
// CDialog_Personal dialog

class CDialog_Personal : public CDialog
{
// Construction
public:
	CDialog_Personal(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CDialog_Personal)
	enum { IDD = IDD_DIALOG_PERSONAL };
	int		m_frec;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDialog_Personal)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CDialog_Personal)
	virtual BOOL OnInitDialog();
	virtual void OnCancel();
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

public:
	bool m_Cancel;

};

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

#endif // !defined(AFX_DIALOG_PERSONAL_H__89DF5609_534D_11D5_871D_F662CFBE2B7A__INCLUDED_)
```

Dialog_Personal.cpp

```
// Dialog_Personal.cpp : implementation file
```

```
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "Dialog_Personal.h"
#include "Global.h"

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

extern CDialog_Filtrado Dlg_f;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Personal dialog


CDialog_Personal::CDialog_Personal(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Personal::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Personal)
	m_frec = 0;
	//}}AFX_DATA_INIT
	m_Cancel=false;
}


void CDialog_Personal::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Personal)
	DDX_Text(pDX, IDC_FRECBANDA, m_frec);
	DDV_MinMaxInt(pDX, m_frec, 20, 20000);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Personal, CDialog)
	//{{AFX_MSG_MAP(CDialog_Personal)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Personal message handlers

BOOL CDialog_Personal::OnInitDialog()
```

```
{
        CDialog::OnInitDialog();

        // Según se van pidiendo hay que cambiar el rotulo fijo

        CString str;

        str.Format("Frecuencia Nž %d", Dlg_f.m_Cont+1);
        SetDlgItemText(IDC_TEXTOFIJO, str);

        // Gestion de la lista de frecuencias anteriormente introducidas, para que el
        // usuario sepa por donde va...

        CListBox* pLista=(CListBox*) GetDlgItem(IDC_LISTADO_FRECUENCIAS);
        if (Dlg_f.m_Cont > 0)   {
                for (int i=0; i<Dlg_f.m_Cont; i++)      {
                        str.Format("%dž Banda en %d", i+1, *(Dlg_f.m_frecuencias+i));
                        pLista->AddString(str);
                }
        }

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

void CDialog_Personal::OnCancel()
{
        // Por si el usuario quiere abortar la operacion

        m_Cancel=true;
        CDialog::OnCancel();
}

void CDialog_Personal::OnOK()
{
        // Gestion de la cancelacion

        m_Cancel=false;
        CDialog::OnOK();
}
```

3.2.9 DirectS

DirectS.h

```
// DirectS.h: interface for the CDirectS class.
//
//////////////////////////////////////////////////////////////////////
```

```
#if !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
#define AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_

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

#include "Global.h"
#include "Estructuras.h"

class CDirectS
{
        // Variables
private:
        int m_Fs;               // Almacena la Frecuencia de muestreo com entero
        int m_Modo;             // Canales a tratar, Stereo(2) o Mono(1)
        int m_Bits;             // Bits por muestra 8 o 16

        int m_SubBuffer;        // Almacena cuanto mide el subbuffer
        int m_j;                // Contador para FFT

        LPDIRECTSOUND                   lpds;
        LPDIRECTSOUNDBUFFER             lpdsbPrimary, lpdsb;
        LPDIRECTSOUNDCAPTURE            lpdsc;
        LPDIRECTSOUNDCAPTUREBUFFER      lpdscb;
        LPDIRECTSOUNDNOTIFY             lpdsNotify;
        HANDLE                          rghEvent[NUMEVENTOS];
        DSBPOSITIONNOTIFY               rgdsbpn[NUMEVENTOS];
        DWORD                           m_TamBuffer;

        HMMIO                           hmmio;
        MMCKINFO                        mmckinfoData, mmckinfoParent;
        MMIOINFO                        mmioinfo;

        DWORD CursorLectura, CursorEscritura;   // Apuntan a los Buffers

public:
        HWND            m_Hwnd;           // Puntero a la ventana principal
        DSCAPS          dscaps;
        DSCCAPS         dsccaps;          // Almacena propiedades de Objetos Ds y DsC
        PDSDEVICEDESC   lpdsddOut;        // Apunta al primero de la lista Repro
        PDSDEVICEDESC   lpdsddIn;         // Apunta al primero de la lista Captura
        PDSDEVICEDESC   lpdsddanterior;

        GUID guidDs, guidDsC;             // Estructuras con info de los Dispositivos
        bool m_Objetos;                   // Objetos creados o no
        char *m_NArchivo;                 // Array con el nombre del archivo a grabar
```

```
        UINT TotalBytesEscritos, TotalBytesLeidos;

private:
        ProcesarEntrada(DWORD Posicion);
        TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total);
        TransferenciaFFT(LPBYTE Orig, DWORD Total);

        // Funciones de acceso a archivos Multimedia
        int WaveCreateFile(char *, HMMIO *, WAVEFORMATEX *, MMCKINFO *, MMCKINFO *);
        int WaveStartDataWrite(HMMIO *, MMCKINFO *, MMIOINFO *);
        int WaveCloseWriteFile(HMMIO *, MMCKINFO *, MMCKINFO *, MMIOINFO *, DWORD);
        int WaveWriteFile(HMMIO, UINT, BYTE *, MMCKINFO *, UINT *, MMIOINFO *);

public:
        CDirectS();
        virtual ~CDirectS();
        SeleccionDispositivo();
        CrearObjetos();
        Configurar();
        InicializarBuffers(bool Captura);
        LanzarCaptura();
//      ReproducirArchivo();            // Coherencia del objeto
        Parar();
        Liberar(bool Todo);
        void LiberarLista();
};

#endif // !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
```

DirectS.cpp

```
// DirectS.cpp: implementation of the CDirectS class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Wave.h"
#include "DirectS.h"
#include "Dialog_Dispositivo.h"

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

CEvent g_TerminarHilo;
```

```
/////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////

CDirectS::CDirectS()
{
        // Inicializaciones adicionales; valores inicialies para punteros
        lpds=NULL;
        lpdsbPrimary=NULL;
        lpdsb=NULL;
        lpdsc=NULL;
        lpdscb=NULL;
        lpdsNotify=NULL;
        // Punteros de la lista de dispositivos
        lpdsddanterior=NULL;
        lpdsddIn=NULL;
        lpdsddOut=NULL;
        // Al iniciar evidentemente aun no se han creado los objetos DirectSound
        // ni DirectSoundCapture
        m_Objetos=false;
}

CDirectS::~CDirectS()
{
}

CDirectS::Configurar()
{
        // Extrae los valores proporcionados por el usuario en el Dialogo_Muestreo y
        // los convierte a formatos deacuerdo con el Objeto Ds.
        // Pasa la Fs de CString a int, el Modo y Bits de bool a int

        char temp[10];
        int i=0;

        // Comprobacion de la entrada de datos
        if (Dlg_m.m_Fs.IsEmpty())
                Dlg_m.m_Fs="44100 Hz";

        for (i=0; i<Dlg_m.m_Fs.GetLength() && Dlg_m.m_Fs[i] !=' ' ; i++)        {
                temp[i]=Dlg_m.m_Fs[i];
        }
        temp[i]='\0';
        m_Fs=atoi(temp);
        Dlg_m.m_Bits==0 ? m_Bits=8 : m_Bits=16;
        Dlg_m.m_Modo==0 ? m_Modo=1 : m_Modo=2;

        return 0;
```

```
}

CDirectS::CrearObjetos()
{
        // Creacion de los objetos con sus interfaces para su posterior uso
        // Solo se crean una vez por ejecucion de la aplicacion, al fianlizar se liberan

        if (!m_Objetos)         {
                m_Hwnd=AfxGetMainWnd()->GetSafeHwnd();
                if (lpds != NULL)       {
                        lpds->Release();
                        lpds=NULL;
                }
                // Creacion objeto DirectSound
                if (DS_OK != DirectSoundCreate(&guidDs, &lpds, NULL))   {
                        // No se ha podido crear
                        TRACE("Error en la creacion del DirectSound\n");
                        return -1;
                }
                // No permite que suene nada más que esta aplicación EXCLUSIVE
                // Ecualizador.exe solo va a grabar, la reproduccion solo que no estorbe
                // cuando estén los otros funcionando
                if (DS_OK != lpds->SetCooperativeLevel(m_Hwnd, DSSCL_NORMAL))   {
                        // No se ha podido establecer la prioridad
                        TRACE("ERROR: al establecer la prioridad\n");
                        return -1;
                }

                if (lpdsc != NULL)      {
                        lpdsc->Release();
                        lpdsc=NULL;
                }
                // Creacion del objeto de Captura
                if (DS_OK != DirectSoundCaptureCreate(&guidDsC, &lpdsc, NULL))  {
                        TRACE("Fue mal la creación de CDirectSoundCapture\n");
                        return -1;
                }

                // Peticion de informacion sobre los dispositivos seleccionados, tanto
                // Reproducción como de Captura
                dscaps.dwSize=sizeof(DSCAPS);
                if (DS_OK != lpds->GetCaps(&dscaps))    {
                       TRACE("La tarjeta no soporta peticion de parametros de captura\n");
                }
                dsccaps.dwSize=sizeof(DSCCAPS);
                if (DS_OK != lpdsc->GetCaps(&dsccaps))  {
                       TRACE("La tarjeta no soporta peticion de parametros de captura\n");
                }
```

```
                m_Objetos=true;
        }
        return 0;
}

CDirectS::InicializarBuffers(bool Captura)
{
        // Esta funcion crea los buffers necesarios para la aplicacion, es decir, se
        // crea lo necesario para un funcionamiento en FULL-DUPLEX, puesto que filtrado
        // lo requiere y Analisis si se quiere RuidoRosa tambien lo ha de implementar.
        // Así se crean 2 buffers para Reproduccion y 1 para captura:
        // Reproduccion: Primario, marca el formato en que se oirá y mezcla todos los
        // secundarios que se estén reproduciendo, el secundario recibira los datos del
        // de captura tratados o generados por el Ruido Rosa
        // Captura: Unicamente recibe datos de la Tarjeta para que se puedan leer

        WAVEFORMATEX            wfx;
        DSBUFFERDESC            dsbd;
        DSCBUFFERDESC           dscbd;

        LPBYTE LpByte, LpByte2;
        DWORD CuenByte2, CuenByte;

        // Todos los buffers tendrán el mismo formato, puesto que evita sobrecarga de
        // CPU y no es el propósito de la aplicacion convertir éstos
        ZeroMemory(&wfx, sizeof(WAVEFORMATEX));
        wfx.wFormatTag=WAVE_FORMAT_PCM;
        wfx.nChannels=m_Modo;
        wfx.nSamplesPerSec=m_Fs;
        wfx.wBitsPerSample=m_Bits;
        wfx.nBlockAlign=wfx.wBitsPerSample/8*wfx.nChannels;
        wfx.nAvgBytesPerSec=wfx.nSamplesPerSec*wfx.nBlockAlign;

        // Creacion del Buffer primario
        ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
        dsbd.dwSize=sizeof(DSBUFFERDESC);
        dsbd.dwFlags=DSBCAPS_PRIMARYBUFFER;
        if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsbPrimary, NULL))       {
                TRACE("Error:Creacion del buffer Primario\n");
                return -1;
        }

        // Ajuste del formato deseado para B.Primario, no hace falta aquí, solo captura
//      if (DS_OK != lpdsbPrimary->SetFormat(&wfx))     {
//              TRACE("ERROR:Ajuste del formato al buffer primario\n");
//              return -1;
//      }                               Coherencia del objeto
```

```
// Creacion del buffer de captura
// Cuentas para tamaño de los buffers
m_SubBuffer=max(4096, wfx.nAvgBytesPerSec/8);
m_SubBuffer -= m_SubBuffer % wfx.nBlockAlign;   // Quede N entero de muestras
                                                // en los buffers

ZeroMemory(&dscbd, sizeof(DSCBUFFERDESC));
dscbd.dwSize=sizeof(DSCBUFFERDESC);
dscbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
dscbd.lpwfxFormat=&wfx;

if (DS_OK != lpdsc->CreateCaptureBuffer(&dscbd, &lpdscb, NULL)) {
        TRACE("Error:Creacion del buffer de Captura\n");
        return -1;
}

// Creacion del buffer Secundario
// Debe ser exactamente igual que el de captura, porque entre ellos se pasarán
// los datos con o sin procesamiento
ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize=sizeof(DSBUFFERDESC);
dsbd.dwFlags=DSBCAPS_GETCURRENTPOSITION2
                        | DSBCAPS_GLOBALFOCUS;
dsbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
dsbd.lpwfxFormat=&wfx;
if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsb, NULL))      {
        TRACE("Error:Creacion Buffer Secundario\n");
        return -1;
}
// Por último llenar el buffer secundario de reproduccion con silencio, evita
// chispazos y picos indeseados
if (DS_OK != lpdsb->Lock(0, 0, (LPVOID*)&LpByte, &CuenByte,
        (LPVOID*)&LpByte2, &CuenByte2, DSBLOCK_ENTIREBUFFER))   //
        TRACE("Error al bloquear el buffer entero\n");
else    {
        if (m_Bits == 8)        {
                FillMemory(LpByte, CuenByte, 128);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 128);
        }
        else    {
                // Muestras de 16 Bits
                FillMemory(LpByte, CuenByte, 0);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 0);
        }
}
if (DS_OK != lpdsb->Unlock((LPVOID)LpByte, CuenByte, (LPVOID)LpByte2,
```

```
        CuenByte2))
        TRACE("Error al desbloquear el buffer entero\n");

m_TamBuffer=dsbd.dwBufferBytes;

// Creacion y configuracion de los eventos
// Tener encuenta quien pedirá eventos, captura (Analisis, Filtrado y Grabacion
// a archivo) o reproduccion (Lectura de archivo)
if (Captura)    {
        // Los Eventos se asociarán a la Captura

        for (int i=0; i<NUMEVENTOS; i++)        {
        rghEvent[i]=CreateEvent(NULL, false, false, NULL);
        if (NULL==rghEvent[i])
                return false;
        }
        for (i=0; i<NUMEVENTOS; i++)    {
                rgdsbpn[i].dwOffset=((m_SubBuffer*i)+m_SubBuffer-1);
                rgdsbpn[i].hEventNotify=rghEvent[i];
        }

        //Asegura que lpdscNotify esta libre
        if (lpdsNotify != NULL) {
                lpdsNotify->Release();
                lpdsNotify=NULL;
        }
        if (DS_OK != lpdscb->QueryInterface(IID_IDirectSoundNotify,
                (void **)&lpdsNotify))  {
                g_Error=35;
                return -1;
        }
        if (DS_OK != lpdsNotify->SetNotificationPositions(NUMEVENTOS, rgdsbpn)) {
                lpdsNotify->Release();
                g_Error=35;
                return -1;
        }
        TRACE("SE HAN CREADO LOS EVENTOS\n");
        // No hace falta la inicilizacion de la FFT que suele venir aquí
        // Por otro lado se procederá a abrir el archivo que contendrá la grabación
        // se prepara el acceso a disco

        if (g_Grabar)   {
                if (WaveCreateFile(m_NArchivo, &hmmio, &wfx, &mmckinfoData,
                                &mmckinfoParent))   {
                        g_Error=35;
                        return -1;
                }
                if (WaveStartDataWrite(&hmmio, &mmckinfoData, &mmioinfo))       {
```

```
                                // Ha ido mal se procede a cerrar y eliminar el archivo
                                WaveCloseWriteFile(&hmmio, &mmckinfoData, &mmckinfoParent,
                                                   &mmioinfo, 0);

                                DeleteFile(m_NArchivo);
                                g_Error=35;
                                return -1;
                        }
                }
        }
        else    {
                // Aqui se configuran los eventos sobre el buffer secundario, cuando se
                // lean archivos WAV
                for (int i=0; i<NUMEVENTOS; i++)        {
                rghEvent[i]=CreateEvent(NULL, false, false, NULL);
                if (NULL==rghEvent[i])
                return false;
                }
                for (i=0; i<NUMEVENTOS; i++)    {
                        rgdsbpn[i].dwOffset=((m_SubBuffer*i)+m_SubBuffer-1);
                        rgdsbpn[i].hEventNotify=rghEvent[i];
                }

                //Asegura que lpdscNotify esta libre
                if (lpdsNotify != NULL) {
                        lpdsNotify->Release();
                        lpdsNotify=NULL;
                }
        }
        return 0;
}

CDirectS::LanzarCaptura()
{
        // Se centraliza en esta funcion, tanto la grabacion a disco, como el
        // funcionamiento en FULL-DUPLEX de la aplicacion, puesto que los dos modos
        // se basan en la captura y será esta la que comande todo la ejecucion en la
        // gestion de los eventos según se van produciendo y atendiendo

        CWnd* pWnd;
        RECT rc;

        if (g_Grabar)   {
                // En caso de que haya que grabar audio a un archivo, se inicializa aqui
                // todo lo necesario para acceder al disco, preparacion del archivo WAV

                if (DS_OK != lpdscb->Start(DSCBSTART_LOOPING))  {
                        TRACE("Error: Arranque de la Captura\n");
```

```
                // Tambien hay que cerrar y eliminar lo poco que se escribió del
                // archivo wav
                WaveCloseWriteFile(&hmmio, &mmckinfoData, &mmckinfoParent,
                                   &mmioinfo, 0);
                DeleteFile(m_NArchivo);
                g_Error=36;
                return -1;
        }
        TotalBytesEscritos=0; // Contador total de Bytes al archivo, son 32 bits
                              // sin signo=>4294967296 a 44100Hz 16bits y stereo
                              // el tiempo de grabación es... 6h 45' 47"

}
else    {
        // Para Analisis y Filtrado lanzar la Captura, y para evitar los chispazos
        // picos o chasquidos de paro-marcha, se coloca el buffer primario en
        // reproduccion continua, así, esta mezclando continuamente.
        if (DS_OK != lpdsbPrimary->Play(0, 0, DSBPLAY_LOOPING)) {
                TRACE("Error:Arranque del buffer Primario\n");
                g_Error=37;
                return -1;
        }
        if (DS_OK != lpdscb->Start(DSCBSTART_LOOPING))  {
                TRACE("Error: Arranque de la Captura\n");
                g_Error=36;
                return -1;
        }
        if (g_Filtrado)         {
                if (DS_OK != lpdsb->SetCurrentPosition(0))      {
                        TRACE("Error:Posicionamiento antes de reproducir\n");
                        g_Error=37;
                        return -1;
                }
                if (DS_OK != lpdsb->Play(0, 0, DSBPLAY_LOOPING))        {
                        TRACE("Error:Arranque del buffer Secundario\n");
                        g_Error=37;
                        return -1;
                }
        }
}

// Gestion de los eventos
// Inicializa el cursor escribir datos en B.Secundario
CursorEscritura=0xFFFFFFFF;

int result;
BOOL Done=false;
while(!Done)    {
```

```
                DWORD dwEvt=MsgWaitForMultipleObjects(
                        NUMEVENTOS,             // Numero de Eventos
                        rghEvent,               // Array donde están los Handles
                        FALSE,                  // £Esperar a todos?
                        INFINITE,               // £Cuanto espero?
                        QS_ALLINPUT);           // £Cualquier Mensaje es suceso?
                // Se normalizan los eventos; WAIT_OBJECT_0=0;
                dwEvt -= WAIT_OBJECT_0;
TRACE("                                                         dwEvt=%d\n",dwEvt);
                if (dwEvt < (DWORD) NUMEVENTOS) {
                        result=ProcesarEntrada(dwEvt);  // Copia datos al fichero o buffer
                        if (result != 0)
                                Done=true;
                }
                // Comprueba el evento de salida
                if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                        Done=true;              // Sale del bucle
                        pWnd=AfxGetMainWnd();
                        rc.top=318;
                        rc.left=194;
                        rc.bottom=343;
                        rc.right=582;   // Para solo repintar este trozo, evita parpadeos
                        // Fuerza el repintado antes de salir
                        pWnd->InvalidateRect(&rc, false);
                }
                // Puesta en pantalla de la cantidad de bytes grabados
                // Actualizar la variable TotalBytesEscritos para mostrarlo por el
                // dialogo, DWORD => 6.76h y 4 GBytes en H.D, creo que es suficiente

                if (dwEvt == 0 || dwEvt == 4 || dwEvt == 8 || dwEvt == 12)      {
                        // Para que refresque cada segundo, no hace falta tanta rapidez
                        pWnd=AfxGetMainWnd();
                        rc.top=318;
                        rc.left=194;
                        rc.bottom=343;
                        rc.right=582;   // Solo repinta este trozo, y evita parpadeos
                        // Fuerza el repintado del dialogo
                        pWnd->InvalidateRect(&rc, false);
                }
        }               // Final del procesamiento de Eventos
        return 0;
}

CDirectS::Parar()
{
        // Centraliza las órdenes de para de los buffers
        if (lpdscb != NULL)
                lpdscb->Stop();         // Para la Captura
```

```
        if (lpdsbPrimary != NULL)
                lpdsbPrimary->Stop();
        if (g_Filtrado || (g_Analisis && g_RuidoRosa))  {
                if (lpdsb != NULL)
                        lpdsb->Stop();
        }
        // Para los últimos datos capturados, al terminar la grabacion, no es necesario
        // forzar otroa escritura a disco, porque queda poco sin llevar al disco, puesto
        // que los subbuffers son pequeños y se escribe cada poco.

        if (g_Grabar)   {
                WaveCloseWriteFile(&hmmio, &mmckinfoData, &mmckinfoParent, &mmioinfo,
                        TotalBytesEscritos/(m_Bits/8));         // Se cierra el archivo
        }

        return 0;
}

CDirectS::Liberar(bool Todo)
{
        // LLeva a cabo la tarea de la liberacion de todo; tanto los buffers solamente,
        // si se cambia el formato, u objetos y buffers si se procede a salir de la
        // aplicacion

        if ((lpdscb !=NULL))    {
                lpdscb->Release();
                lpdscb=NULL;
        }
        if (lpdsbPrimary != NULL)       {
                lpdsbPrimary->Release();
                lpdsbPrimary=NULL;
        }
        if (lpdsb != NULL)              {
                lpdsb->Release();
                lpdsb=NULL;
        }
        if (lpdsNotify != NULL) {
                lpdsNotify->Release();
                lpdsNotify=NULL;
        }

        if ((lpdsc != NULL) && Todo)    {
                lpdsc->Release();
                lpdsc=NULL;
        }
        if ((lpds != NULL) && Todo)             {
                lpds->Release();
                lpds=NULL;
```

```
        }
        if (Todo)
                m_Objetos=false;

        return 0;
}

void CDirectS::LiberarLista()
{
        // Desvincular las estructuras que enumeraron los dispositivos

        PDSDEVICEDESC lpdsddtemp, lpdsddant;

        if (lpdsddOut != NULL)  {
                lpdsddant=lpdsddOut;
                do      {
                        if (lpdsddant->pNext == NULL)   {
                                // Solo queda un elemento en la lista
                                delete(lpdsddant->lpDeviceDesc);
                                delete(lpdsddant->lpDriverDesc);
                                delete(lpdsddant);
                                lpdsddOut=NULL;
                        }
                        else    {
                                lpdsddtemp=lpdsddant->pNext;
                                if (lpdsddtemp->pNext == NULL)  {
                                        // Liberar antes el array char
                                        delete(lpdsddtemp->lpDeviceDesc);
                                        delete(lpdsddtemp->lpDriverDesc);
                                        delete(lpdsddtemp);
                                        lpdsddant->pNext=NULL;
                                        lpdsddant=lpdsddOut;
                                }
                                else    {
                                        lpdsddant=lpdsddant->pNext;
                                }
                        }
                } while (lpdsddOut != NULL);
        }
        lpdsddOut=NULL;         // indica queda vacio

        if (lpdsddIn != NULL)   {
                lpdsddant=lpdsddIn;
                do      {
                        if (lpdsddant->pNext == NULL)   {
                                // Solo queda un elemento en la lista
                                delete(lpdsddant->lpDeviceDesc);
                                delete(lpdsddant->lpDriverDesc);
```

```
                                delete(lpdsddant);
                                lpdsddIn=NULL;
                        }
                        else    {
                                lpdsddtemp=lpdsddant->pNext;
                                if (lpdsddtemp->pNext == NULL)  {
                                        // Liberar antes el array char
                                        delete(lpdsddtemp->lpDeviceDesc);
                                        delete(lpdsddtemp->lpDriverDesc);
                                        delete(lpdsddtemp);
                                        lpdsddant->pNext=NULL;
                                        lpdsddant=lpdsddIn;
                                }
                                else    {
                                        lpdsddant=lpdsddant->pNext;
                                }
                        }
                } while (lpdsddIn != NULL);
        }
        lpdsddIn=NULL;          // indica queda vacio

        return;
}


CDirectS::ProcesarEntrada(DWORD Posicion)
{
        // Gestion de los bloqueos de los buffers tanto de Captura como de reproduccion,
        // para pasar los datos de uno a otro, o bien directamente copiarlo a un archivo
        // .WAV, puesto que los eventos en ambos casos son del Buffer de Captura

        DWORD                   dwNumBytes;
        LPBYTE                  pbInput1, pbInput2, pbOutput1, pbOutput2;
        DWORD                   cbInput1, cbInput2, cbOutput1, cbOutput2;
        UINT                    BytesEscritos;          // Bytes ya escritos

        if (Posicion == 0)      {
                // Primer SubBuffer del Buffer
                CursorLectura=0;
                dwNumBytes=(rgdsbpn[Posicion].dwOffset)+1;
        }
        else    {
                // Resto de subdivisiones, en las que está compuesto el buffer
                CursorLectura=(rgdsbpn[Posicion-1].dwOffset)+1;
                dwNumBytes=m_SubBuffer;
                // rgdsbpn[Posicion].dwOffset - rgdsbpn[Posicion-1].dwOffset;
                // Otra opcion pero como como sabe el tamaño de los subbuffers
        }
```

```
        // Bloqueo del buffer de Captura

// Nunca hay Wraparound porque las peticiones de bloqueo se hacen ajustandose
// al buffer de Captura, esta subdividido en 16 bloques y se seleccionan cada
// uno de éstos, por lo que pbInput2 y cbInput2 no sirven para nada

        if (DS_OK != lpdscb->Lock(CursorLectura, dwNumBytes,
                        (LPVOID *)&pbInput1, &cbInput1,
                        (LPVOID *)&pbInput2, &cbInput2, 0))     {
                TRACE("Error:Bloqueo Buffer de Captura\n");
                g_Error=38;
                return -1;
        }
        else    {
                if (g_Grabar)   {
                      // Solo tener en cuenta el buffer de captura
                      if ((TotalBytesEscritos+cbInput1) < 4294967296) {
                              if (WaveWriteFile(hmmio, cbInput1, pbInput1, &mmckinfoData,
                                        &BytesEscritos, &mmioinfo))     {
                                        g_Error=36;
                                        TRACE("Error: Al escribir en el archivo\n");
                                        return -1;
                                }
                                TotalBytesEscritos+=BytesEscritos;
                        }
                        else
                                // Por si se supera el límite del registro de 32 Bits,
                                g_TerminarHilo.SetEvent();
                }
                else if (g_Analisis)    {
                        // Calcular la FFT a partir de las muestras capturadas, asi que
                        // solo atender buffer de captura
                        TransferenciaFFT(pbInput1, cbInput1);
                }
                else    {
                        // Aquí entrará en modo Filtrado, lo cual implica una atencion a
                        // ambos buffers, leer del de Captura y escribir en el secundario
                        // Preparar el bloqueo del buffer secundario
                        if (CursorEscritura == 0xFFFFFFFF)      {
                      // La primera escritura, hay que saber por donde va reproducciendo
                      // y adelantarse un poco para que no nos pille poniendo datos
                                if (DS_OK != lpdsb->GetCurrentPosition(NULL,
                                           &CursorEscritura)) {
                                        TRACE("Error:Peticion de posicion en Buffer
                                               Secundario\n");
                                        g_Error=39;
                                }
                                else    {
```

```
                                        // Adelanta 2, 3 o 4 SubBuffers
                                        CursorEscritura+=(4*m_SubBuffer);
                                        while (CursorEscritura >= m_TamBuffer)
                                                CursorEscritura-=m_TamBuffer;
                                }
                        }
                        // Bloqueo del buffer Secundario (Reproducción)

// El hecho de escribir por primera vez tomando como referencia donde se encuantra
// el cursor de lectura hace que no concuerden los bloqueos del buffer secundario
// con el tamaño del buffer => Wraparound, tener encuenta pbOutput2 y cbOutput2

                        if (DS_OK != lpdsb->Lock(CursorEscritura, dwNumBytes,
                                        (LPVOID *)&pbOutput1, &cbOutput1,
                                        (LPVOID *)&pbOutput2, &cbOutput2, 0))   {
                                TRACE("Error:Bloqueo del buffer Secundario\n");
                                g_Error=39;
                        }
                        else    {
                        // Aquí los dos Buffers tienen una zona bloqueada cada uno, con lo
                        // cual es el momento de llevar los datos de Captura al Secundario,
                        // copiandolos o haciendo cosas con ellos por el camino
                                TransferenciaBuffers(pbOutput1, pbInput1, cbOutput1);
                                if (pbOutput2 != NULL)  {
                                        TRACE("Hay pbOutput2\n");
                                        TransferenciaBuffers(pbOutput2, pbInput1+cbOutput1,
                                                                cbOutput2);
                                }
                        }
                        // Desbloqueo del Buffer Secundario
                        if (DS_OK != lpdsb->Unlock(pbOutput1, cbOutput1, pbOutput2,
                                                cbOutput2)){
                                TRACE("Error:Desbloqueo del Buffer Secundario\n");
                                g_Error=41;
                        }
                        CursorEscritura+=dwNumBytes;    // cbInput1=dwNumBytes
                        while (CursorEscritura >= m_TamBuffer)
                                CursorEscritura-=m_TamBuffer;
                }
                if (DS_OK != lpdscb->Unlock(pbInput1, cbInput1, pbInput2, cbInput2))    {
                        TRACE("Error:Desbloqueo del Buffer de Captura\n");
                        g_Error=40;
                        return -1;
                }
                if (g_Error != 0)
                // Problemas al desbloquear el buffer secundario, pero se comprueba aqui
                // para desbloquear el de captura y ya salir
                        return -1;
```

```
        }
        // Puesta en orden del Cursor de Lectura
        CursorLectura+=dwNumBytes;
        while (CursorLectura >= m_TamBuffer)
                CursorLectura-=m_TamBuffer;

        return 0;
}

CDirectS::TransferenciaFFT(LPBYTE Orig, DWORD Total)
{
        // Realiza las transferencias entre el buffer de captura y prepara los datos
        // para pasarlos al objeto que calcula la FFT.

        // En este programa Ecualizador.exe la única gestion que se hace de DirectSound
        // es la grabacion a disco de archivos Wav por eso se ha eliminado esta parte
        // en este programa


        return 0;
}

CDirectS::TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total)
{
        // Realiza las transferencias entre buffers, con dos transformadas FFT y
        // filtrado entre medias.
        // Se realiza la FFT del T a F, se modifican las componentes en la frecuencia
        // como corresponde a un filtro y por último se calcula FFT de F a T y se
        // llevan los datos al buffer de salida

        // Se ha mantenido la estructura por la facilidad que brinda, de poder manipular
        // los datos en el trasiego entre buffers, aunque no venga nunca el flujo del
        // programa hasta aquí.

        register DWORD i;               // Contador
        static BYTE temp;       // Almacena el byte sobrante en transfer impares

        //      Comprueba como van colocados los datos en los buffers
        int flag=0;
        for (i=0; i<Total; i++) {
                switch (flag)   {
                        case 0:
                                *Dest=*Orig;
                                break;
                        case 1:
                                *Dest=*Orig;
                                break;
                        case 2:
```

```
                                *Dest=*Orig;
                                break;
                        case 3:
                                *Dest=*Orig;
                                break;
                }
                Dest++;
                Orig++;
                flag++;
                if (flag >3) flag=0;
//              *Dest++=*Orig++;        // Transferencia directa
        }
        if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                m_j=0;                  // Cuando se termina se puede cambiar de formato
                                        // luego se inicializa todo
                g_TerminarHilo.SetEvent();      // Evita Resets automaticos por el chequeo
        }

        return 0;
}

CDirectS::SeleccionDispositivo()
{
        // Desde aquí se centraliza toda la gestion del escaneo de todos los
        // dispositivos de audio que haya en el sistema, tanto para DirectSound como
        // para DirectSoundCapture, y mediante el correspondiente dialogo el usuario
        // podrá seleccionar el que prefiere utilizar.

        // Se crea una sola vez la lista y luego solo se lee y se selecciona con el
        // dialogo correspondiente. Hasta ahora no se pueden pinchar en caliente
        // tarjetas, por eso solo se rastrea una vez. No para PCMCIA

        PDSDEVICEDESC lpdsddtemp;
        CString str;

        lpdsddanterior=NULL;
        // Enumeracion de Dispositivos....
        if (DS_OK != DirectSoundEnumerate(EnumCallBackProc, &lpdsddOut))        {
                TRACE("Error: En la enumeracion de dispositivos Ds\n");
                return -1;
        }

        lpdsddanterior=NULL;
        if (DS_OK != DirectSoundCaptureEnumerate(EnumCallBackProc, &lpdsddIn))  {
                TRACE("Error: En la enumeracion de dispositivos DsCapture\n");
                return -1;
        }
```

```
        CDialog_Dispositivo Dlg_d;
        Dlg_d.m_DispRepro=lpdsddOut->lpDeviceDesc;              // Inicializacion
        Dlg_d.m_DispCaptura=lpdsddIn->lpDeviceDesc;
        Dlg_d.DoModal();

        lpdsddtemp=lpdsddOut;
        do      {
                str=lpdsddtemp->lpDeviceDesc;
                if (str == Dlg_d.m_DispRepro)
                        guidDs=lpdsddtemp->guid;
                lpdsddtemp=lpdsddtemp->pNext;
        } while (lpdsddtemp != NULL);

        lpdsddtemp=lpdsddIn;
        do      {
                str=lpdsddtemp->lpDeviceDesc;
                if (str == Dlg_d.m_DispCaptura)
                        guidDsC=lpdsddtemp->guid;
                lpdsddtemp=lpdsddtemp->pNext;
        } while (lpdsddtemp != NULL);

        // Antes de salir sería conveniente obligar a escribir en "OPCIONES.CFG" la
        // nueva información, puesto que es posible que se haya modificado
        if (g_primeraMuestreo)          {
                // Esperar a que haya una  primera configuracion de muestreo
                Dlg_m.Grabar();
        }

        return 0;
}

////////////////////////////////////////////////////////////////////////////////
//
// Funciones Globales

BOOL CALLBACK EnumCallBackProc( GUID * lpguidDevice, LPCSTR lpszDesc,
                                LPCSTR lpszDrvName, LPVOID lpContext )
{
    // Entra tantas veces como dispositivos haya en la máquina
        // Lo normal 2 veces, el asignado por Güindous y luego el de la tarjeta que
        // se tenga instalada.

        // El puntero que se manda en DirectSoundEnumerate, en este caso, lpdsddOut,
        // se recibe en lpContext y con formato LPVOID, en vez de PDSDEVICEDESC.

        PDSDEVICEDESC *lpdsdd=(PDSDEVICEDESC *) lpContext;      // Copia
        PDSDEVICEDESC lpdsddnew;                // Para hacerlo dinámico
        extern CDirectS Ds;                     // Funcion ajena al Objeto DirectS aunque
```

```
                                                //  está en el mismo archivo

        // Creacion de una estructura por cada Dispositivo
        if (Ds.lpdsddanterior == NULL)          {
                // El primero
                lpdsddnew= new DSDEVICEDESC;
                if( lpguidDevice != NULL )
                        lpdsddnew->guid= *lpguidDevice;
                else
                        lpdsddnew->guid=GUID_NULL;
                lpdsddnew->lpDeviceDesc=new char[lstrlen(lpszDesc)+1];
                lstrcpy( lpdsddnew->lpDeviceDesc, lpszDesc );
                lpdsddnew->lpDriverDesc=new char[lstrlen(lpszDrvName)+1];
                lstrcpy( lpdsddnew->lpDriverDesc, lpszDrvName);
                lpdsddnew->pNext=NULL;
                Ds.lpdsddanterior=lpdsddnew;
                *lpdsdd=lpdsddnew;              // Puntero al primero, se ve desde fuera
        }
        else    {
                // Ya hay al menos un elemento en la lista, enlazarlo con ella
                lpdsddnew= new DSDEVICEDESC;
                if( lpguidDevice != NULL )
                        lpdsddnew->guid= *lpguidDevice;
                else
                        lpdsddnew->guid=GUID_NULL;
                lpdsddnew->lpDeviceDesc=new char[lstrlen(lpszDesc)+1];
                lstrcpy( lpdsddnew->lpDeviceDesc, lpszDesc );
                lpdsddnew->lpDriverDesc=new char[lstrlen(lpszDrvName)+1];
                lstrcpy( lpdsddnew->lpDriverDesc, lpszDrvName);
                lpdsddnew->pNext=NULL;
                // Enlace del anterior con este
                Ds.lpdsddanterior->pNext=lpdsddnew;
                Ds.lpdsddanterior=lpdsddnew;
        }


    return TRUE;
}



////////////////////////////////////////////////////////////////////////////////
//
// Funciones pertenecientes a WAVE.C que viene con la SDK DirectX para el acceso a
// disco. Las he incluido como funciones miembro del objetos DirectS pero el
// origen es el ya descrito, por eso ocupan este lugar detrás de la función global
// Originalmente eran funciones C pero se ha traido al obejto C++
// Va unida a Wave.h con las cabeceras necesarias para compilarlo
```

```
/*==========================================================================
 *
 *  Copyright (C) 1995-1997 Microsoft Corporation. All Rights Reserved.
 *
 *  File:               wave.c
 *  Content:    Wave library routines.
 *      This file is used for loading/saving waves, and reading and
 *      writing waves in smaller blocks.
 *      Uses WaveOpenFile, WaveReadFile and WaveCloseReadFile for
 *      single block access to reading wave files.
 *      Uses WaveCreateFile, WaveWriteFile, WaveCloseWriteFile for
 *      single block access for writing  wave files.
 *      Uses WaveLoadFile to load a whole wave file into memory.
 *      Uses WaveSaveFile to save a whole wave file into memory.
 *
 ***************************************************************************/


/* ROUTINES */
/* -------------------------------------------------------*/

/*  This routine will create a wave file for writing.  This will automatically over-
        write any existing file with the same name, so be careful and check before hand!!!
        pszFileName     - Pointer to filename to write.
        phmmioOut               - Pointer to HMMIO handle that is used for further writes
        pwfxDest                - Valid waveformatex destination structure.
        pckOut                  - Pointer to be set with the MMCKINFO.
        pckOutRIFF              - Pointer to be set with the RIFF info.

*/
CDirectS::WaveCreateFile(
                        char *pszFileName,              // (IN)
                        HMMIO *phmmioOut,               // (OUT)
                        WAVEFORMATEX *pwfxDest,         // (IN)
                        MMCKINFO *pckOut,               // (OUT)
                        MMCKINFO *pckOutRIFF            // (OUT)

                        )


/*  This routine has to be called before any data is written to the wave output file, via
    wavewritefile. This sets up the data to write, and creates the data chunk.
*/

CDirectS::WaveStartDataWrite(
                                HMMIO *phmmioOut,           // (IN)
                                MMCKINFO *pckOut,           // (IN)
```

```
                                MMIOINFO *pmmioinfoOut      // (OUT)
                                )

/*  This routine will close a wave file used for writing.  Returns 0 if successful,
        else the error code.
        phmmioOut       - Pointer to mmio handle for saving.
        pckOut          - Pointer to the MMCKINFO for saving.
        pckOutRiff      - Pointer to the riff MMCKINFO for saving.
        pmmioinfoOut- Pointer to mmioinfo for saving.
        cSamples        - # of samples saved, for the fact chunk.  For PCM, this isn't
                          used but will be written anyway, so this can be zero as long
                          as programs ignore this field when they load PCM formats.
*/
CDirectS::WaveCloseWriteFile(
                        HMMIO *phmmioOut,               // (IN)
                        MMCKINFO *pckOut,               // (IN)
                        MMCKINFO *pckOutRIFF,           // (IN)
                        MMIOINFO *pmmioinfoOut,         // (IN)
                        DWORD cSamples                  // (IN)
                        )

/* This routine will write out data to a wave file.
        hmmioOut                - Handle to hmmioOut filled by WaveCreateFile
        cbWrite                 - # of bytes to write out.
        pbSrc                   - Pointer to source.
        pckOut                  - pointer to ckOut filled by WaveCreateFile
        cbActualWrite   - # of actual bytes written.
        pmmioinfoOut    - Pointer to mmioinfoOut filled by WaveCreateFile.

        Returns 0 if successful, else the error code.

 */


CDirectS::WaveWriteFile(
                HMMIO hmmioOut,                 // (IN)
                UINT cbWrite,                   // (IN)
                BYTE *pbSrc,                    // (IN)
                MMCKINFO *pckOut,               // (IN)
                UINT *cbActualWrite,            // (OUT)
                MMIOINFO *pmmioinfoOut          // (IN)
                )
```

3.2.10 EcualizadorDlg

EcualizadorDlg.h

```
// EcualizadorDlg.h : header file
```

```
//

#if !defined(AFX_ECUALIZADORDLG_H__104E5527_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_ECUALIZADORDLG_H__104E5527_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_

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

/////////////////////////////////////////////////////////////////////////////
// CEcualizadorDlg dialog

class CEcualizadorDlg : public CDialog
{
// Construction
public:
	CEcualizadorDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CEcualizadorDlg)
	enum { IDD = IDD_ECUALIZADOR_DIALOG };
	CString m_NArchivo;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEcualizadorDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;
	CBitmap m_BmpAnalisis, m_BmpFiltrado, m_BmpGrabar, m_BmpPararGrab;
	CString m_DisplayArchivo;
	bool m_Dispositivos;				// Para lanzarlo la primera vez
	// Generated message map functions
	//{{AFX_MSG(CEcualizadorDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnDestroy();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnAnalisis();
	afx_msg void OnOpcionesGraficos();
	afx_msg void OnOpcionesMuestreo();
	afx_msg void OnFiltrado();
	afx_msg void OnOpcionesFiltrado();
```

```
        afx_msg void OnOpcionesDispositivos();
        afx_msg void OnCapsCaptura();
        afx_msg void OnCapsReproduccion();
        afx_msg void OnClose();
        virtual void OnOK();
        afx_msg void OnGrabar();
        afx_msg void OnParar();
        afx_msg void OnHelp();
        //}}AFX_MSG
        afx_msg void OnError();
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_ECUALIZADORDLG_H__104E5527_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

EcualizadorDlg.cpp

```
// EcualizadorDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Ecualizador.h"
#include "EcualizadorDlg.h"
#include "Global.h"

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

// Objetos Globales
CDialog_Muestreo Dlg_m;
CDialog_Graficos Dlg_g;
CDialog_Error    Dlg_e;
CDialog_Filtrado Dlg_f;

CDirectS Ds;

// Varibales Globales
bool g_primeraMuestreo;
bool g_primeraFiltrado;
bool g_primeraGraficos; // Lanzamiento la primera vez para tener una opciones
bool g_Grabar;                  // Estado de grabación
```

```
bool g_Filtrado;                //    "          Filtrado
bool g_Analisis;                //    "          Analisis
bool g_RuidoRosa;               //    "          Ruido Rosa
bool g_DispositivoOk;   // Para saber si se ha seleccionado un dispositivo

// Filtro para las extensiones en los diálogos de selección de archivo
static char g_FiltroArch[]="Archivos Wav (*.wav) | *.wav||";

/////////////////////////////////////////////////////////////////////////////
// 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)
        //}}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()

/////////////////////////////////////////////////////////////////////////////
// CEcualizadorDlg dialog

CEcualizadorDlg::CEcualizadorDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CEcualizadorDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CEcualizadorDlg)
        m_NArchivo = _T("");
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
        g_primeraMuestreo=false;
        g_primeraGraficos=false;
        g_primeraFiltrado=false;
        m_Dispositivos=false;
        g_Grabar=false;
        g_Filtrado=false;
        g_Analisis=false;
        g_RuidoRosa=false;
        g_DispositivoOk=false;          // Inicializaciones varias

        // El nombre de archivo se vacía para evitar desastres
        m_NArchivo.Empty();

        // Comprobacion de la Version de DirectX puesto que sin esto no hay nada que
        // hacer.

        DWORD   pdwVersion, pdwRevision;

        // Inspeccion de los DirectX, al menos version 6.0
        int result=DirectXSetupGetVersion(&pdwVersion, &pdwRevision);
        if (result =! 0)        {
                // La funcion devuelve un valor correcto
                TRACE("Version de DirectX instalada %d.%d\n",HIWORD(pdwVersion),
                        LOWORD(pdwVersion));
                if (LOWORD(pdwVersion) < 6)     {
                        // Ha de haber instaladas DirectX 6.0 o superior
                        g_Error=100;
                }
        }
}

void CEcualizadorDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CEcualizadorDlg)

        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CEcualizadorDlg, CDialog)
        //{{AFX_MSG_MAP(CEcualizadorDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_DESTROY()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_ANALISIS, OnAnalisis)
        ON_BN_CLICKED(IDC_OPCIONES_GRAFICOS, OnOpcionesGraficos)
        ON_BN_CLICKED(IDC_OPCIONES_MUESTREO, OnOpcionesMuestreo)
        ON_BN_CLICKED(IDC_FILTRADO, OnFiltrado)
        ON_BN_CLICKED(IDC_OPCIONES_FILTRADO, OnOpcionesFiltrado)
        ON_BN_CLICKED(IDC_OPCIONES_DISPOSITIVOS, OnOpcionesDispositivos)
        ON_BN_CLICKED(IDC_CAPS_CAPTURA, OnCapsCaptura)
        ON_BN_CLICKED(IDC_CAPS_REPRODUCCION, OnCapsReproduccion)
        ON_WM_CLOSE()
        ON_BN_CLICKED(IDC_GRABAR, OnGrabar)
        ON_BN_CLICKED(IDC_PARAR, OnParar)
        ON_WM_KEYUP()
        ON_BN_CLICKED(ID_HELP, OnHelp)
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_ERROR, OnError)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEcualizadorDlg message handlers

BOOL CEcualizadorDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
```

```
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon

        // Inicializaciones adicionales, poner los bmp en los botones
        m_BmpAnalisis.LoadBitmap(IDB_COM_ANALISIS);
        HBITMAP hBitmap=(HBITMAP) m_BmpAnalisis.GetSafeHandle();
        ((CButton*) GetDlgItem(IDC_ANALISIS))->SetBitmap(hBitmap);

        m_BmpFiltrado.LoadBitmap(IDB_COM_FILTRADO);
        hBitmap=(HBITMAP) m_BmpFiltrado.GetSafeHandle();
        ((CButton*) GetDlgItem(IDC_FILTRADO))->SetBitmap(hBitmap);

        m_BmpGrabar.LoadBitmap(IDB_COM_GRABAR);
        hBitmap=(HBITMAP) m_BmpGrabar.GetSafeHandle();
        ((CButton*) GetDlgItem(IDC_GRABAR))->SetBitmap(hBitmap);

        m_BmpPararGrab.LoadBitmap(IDB_COM_PARARGRAB);
        hBitmap=(HBITMAP) m_BmpPararGrab.GetSafeHandle();
        ((CButton*) GetDlgItem(IDC_PARAR))->SetBitmap(hBitmap);


        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CEcualizadorDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

void CEcualizadorDlg::OnDestroy()
{
```

```
        WinHelp(0L, HELP_QUIT);
        CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CEcualizadorDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }

        // Para refrescar algunos controles de dialogo en tiempo real, las ventanas
        // adicionales que se han añadido a la sección de grabación

        CString str;
        float tempf;
        CWnd* pWnd=GetDlgItem(IDC_BYTES);       // Obtencion de puntero al recurso
        CDC* pControlDC=pWnd->GetDC();

        // Pone en pantalla los bytes que va ocupando la grabación
        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SetTextColor(RGB(0,153,0));
        pControlDC->SetBkColor(RGB(191, 191, 191));
        tempf=(float)Ds.TotalBytesEscritos;
        if (Ds.TotalBytesEscritos == 0)
                str.Empty();
```

```
        else if (Ds.TotalBytesEscritos < 1048576)
                // No ha llegado al primer MegaByte
                str.Format("%.3f KBytes",(tempf/1024)); // Division 1024
        else
                str.Format("%.3f MBytes",(tempf/1048576));      // Division 1024^2
        pControlDC->TextOut(10, 0, str);
        pWnd->ReleaseDC(pControlDC);

        // Pone el nombre dle archivo que se graba
        pWnd=GetDlgItem(IDC_DISPLAYNOMBRE);
        pControlDC=pWnd->GetDC();
        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SetTextColor(RGB(0,153,0));
        pControlDC->SetBkColor(RGB(191, 191, 191));
        pControlDC->TextOut(0,0,m_DisplayArchivo);
        pWnd->ReleaseDC(pControlDC);

        // Pone la configuracion de captura actual
        pWnd=GetDlgItem(IDC_DISPLAY_FS);
        pControlDC=pWnd->GetDC();
        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SetTextColor(RGB(0,153,0));
        pControlDC->SetBkColor(RGB(191, 191, 191));
        pControlDC->TextOut(25,0,Dlg_m.m_Fs);
        pWnd->ReleaseDC(pControlDC);

        pWnd=GetDlgItem(IDC_DISPLAY_BITS);
        pControlDC=pWnd->GetDC();
        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SetTextColor(RGB(0,153,0));
        pControlDC->SetBkColor(RGB(191, 191, 191));
        pControlDC->TextOut(0, 0, Dlg_m.m_Bits ? "16 Bits" : "8 Bits");
        pWnd->ReleaseDC(pControlDC);

        pWnd=GetDlgItem(IDC_DISPLAY_MODO);
        pControlDC=pWnd->GetDC();
        pWnd->Invalidate();
        pWnd->UpdateWindow();
        pControlDC->SetTextColor(RGB(0,153,0));
        pControlDC->SetBkColor(RGB(191, 191, 191));
        pControlDC->TextOut(0,0, Dlg_m.m_Modo ? "Stereo" : "Mono");
        pWnd->ReleaseDC(pControlDC);
}

// The system calls this to obtain the cursor to display while the user drags
```

```
//  the minimized window.
HCURSOR CEcualizadorDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CEcualizadorDlg::OnAnalisis()
{
        // Antes de lanzar el programa que realmente realiza el analisis, hay que
        // pedirle al usuario que introduzca sus preferencias en los parámetro que
        // son configurables

        g_Error=0;

        // Lanzamiento de la selección de dispositivo
        if (!g_Grabar)  {
                // Evitar que se esté en proceso de grabación
                if (!m_Dispositivos)    {
                        OnOpcionesDispositivos();
                        m_Dispositivos=true;
                }

                if (g_DispositivoOk)    {
                        if (!g_primeraMuestreo)         {
                                Dlg_m.Inicializacion();
                                g_primeraMuestreo=true;
                        }
                        if (!g_primeraGraficos)         {
                                Dlg_g.Inicializacion();
                                g_primeraGraficos=true;
                        }

                // Ahora ya se ha creado un "OPCIONES.CFG", con lo que ya se puede lanzar
                // el analisis..
                        if (g_Error == 0)       {
                                // Evita conflictos con Analisis.exe y filtrado.exe
                                Ds.Liberar(true);
                                // Llamada al programa
                                _spawnl( _P_WAIT, "Analisis.exe", "uno", NULL );
                        }
                }
        }

        return;

}

void CEcualizadorDlg::OnOpcionesGraficos()
```

```
{
        // Se usan estas funciones para cambiar las configuracion de los parámetros
        // que se utillizarán para la representacion grafica. Por lo tanto se lanza
        // el dialogo de las opciones graficas, con todo lo que conlleva...

        if (g_Grabar)
                return;

        Dlg_g.Inicializacion();
        g_primeraGraficos=true;

        return;
}

void CEcualizadorDlg::OnOpcionesMuestreo()
{
        // En esta función se lanzará el dialogo para modificar las opciones de la
        // captura, asi que se lanza el dialogo correspondiente

        if (g_Grabar)
                return;

        Dlg_m.Inicializacion();
        g_primeraMuestreo=true;

        return;
}

void CEcualizadorDlg::OnFiltrado()
{
        // En esta funcion se gestiona el lanzamiento y los preparativos previos del
        // filtrado

        g_Error=0;

        if (!g_Grabar)  {

                // Lanzamiento de la selección de dispositivo
                if (!m_Dispositivos)    {
                        OnOpcionesDispositivos();
                        m_Dispositivos=true;
                }

                if (g_DispositivoOk)    {
                        if (!g_primeraMuestreo)         {
                                Dlg_m.Inicializacion();
                                g_primeraMuestreo=true;
                        }
```

```
                        if (!g_primeraFiltrado) {
                                Dlg_f.Inicializacion();
                                if (g_Error == 0)
                                        g_primeraFiltrado=true;
                        }
                        if (g_Error == 0)       {
                                // Evita conflictos con Analisis.exe y filtrado.exe
                                Ds.Liberar(true);
                                // Lanzamiento del programa que realiza el filtrado
                                _spawnl( _P_WAIT, "Filtrado.exe", "uno", NULL );
                        }
                }
        }

        return;
}

void CEcualizadorDlg::OnOpcionesFiltrado()
{
        // Se encarga de lanzar cuando el usuario lo deseé el dialogo apropiado para
        // modificar las condiciones del modo filtrado

        if (g_Grabar)
                return;         // Aborta la invocación si se está grabando

        if (!g_primeraFiltrado)         {
                Dlg_f.Inicializacion();
                g_primeraFiltrado=true;
        }
        else
                Dlg_f.DoModal();

        return;
}


void CEcualizadorDlg::OnError()
{
        // Gestiona el alnzamiento del dialogo para los mensajes de error.
        // Deja todo tal como estaba según donde se haya producido

        switch (g_Error)        {
        case 15:
        case 14:
        case 13:
        case 12:
        case 11:
//              Dd.Restaurar();
```

```
        case 10:
                ShowCursor(true);
//              Dd.Destruir(true);   // Compatibilidad con el objeto usado en
                                     // Analisis y filtrado
        case 41:
        case 40:
        case 39:
        case 38:
        case 37:
        case 36:
                Ds.Parar();
        case 35:
        case 34:
        case 33:
        case 32:
        case 31:
        case 30:
                Ds.Liberar(true);
                break;
        default:
                break;
        }

        if (g_Error != 0)       {
                Dlg_e.DoModal();
                g_Grabar=false;
                g_TerminarHilo.ResetEvent();
        }

        return;
}



void CEcualizadorDlg::OnOpcionesDispositivos()
{
        // Permite lanzar el dialogo para cambiar los dispositivos que se usan tanto
        // para la reproduccion como la captura

        static bool primera=false;
        PDSDEVICEDESC lpdsddtemp;
        CString str;

        if (g_Grabar)
                return;

        if (!primera)   {
                // Se crea la lista y se hace todo lo necesario para que el usuario
```

```
                // pueda seleccionar el que prefiera
                Ds.SeleccionDispositivo();
                primera=true;
                m_Dispositivos=true;
        }
        else    {
                // Como ya está la lista creada solo hay que leerla
                CDialog_Dispositivo Dlg_d;

                Dlg_d.m_DispRepro=Ds.lpdsddOut->lpDeviceDesc;    // Inicializacion
                Dlg_d.m_DispCaptura=Ds.lpdsddIn->lpDeviceDesc;
                Dlg_d.DoModal();

                lpdsddtemp=Ds.lpdsddOut;
                do      {
                        str=lpdsddtemp->lpDeviceDesc;
                        if (str == Dlg_d.m_DispRepro)
                                Ds.guidDs=lpdsddtemp->guid;
                        lpdsddtemp=lpdsddtemp->pNext;
                } while (lpdsddtemp != NULL);

                lpdsddtemp=Ds.lpdsddIn;
                do      {
                        str=lpdsddtemp->lpDeviceDesc;
                        if (str == Dlg_d.m_DispCaptura)
                                Ds.guidDsC=lpdsddtemp->guid;
                        lpdsddtemp=lpdsddtemp->pNext;
                } while (lpdsddtemp != NULL);

                // Antes de salir sería conveniente obligar a escribir en "OPCIONES.CFG" la
                // nueva información, puesto que es posible que se haya modificado
                if (g_primeraMuestreo)          {
                        // Esperar a que haya una  primera configuracion de muestreo
                        Dlg_m.Grabar();
                }
        }
        return;
}

void CEcualizadorDlg::OnCapsCaptura()
{
        // Cuando el usuario desé ver las capacidades de su tarjeta en cuanto a
        // captura pulsará este botón. En principio este programa Ecualizador.exe, solo
        // usa la tarjeta de sonido para capturar y llevarlo al disco, por lo tanto
        // para no interferir con los otros 2 programas Filtrado.exe o Analisis.exe
        // que hacen un uso exclusivo del dispositivo de sonido se crean los objetos
        // DirectSound y DirectSoundCapture se le pide info acerca de sus capacidades,
        // no es necesario crear buffers, puesto que se busca info del dispositivo y
```

```
        // se liberan.


        if (!g_Grabar)  {
                if (!g_DispositivoOk)   {
                        // No se ha seleccionado aún
                        OnOpcionesDispositivos();
                        g_DispositivoOk=true;
                }

                Ds.CrearObjetos();

                // Los resultados están en Ds.dsccaps
                // La estructura para analizar los resultados se ubicará en el código del
                // dialogo que mostrará los resultados fáclimente legibles
                CDialog_CapsCaptura Dlg;
                Dlg.DoModal();

                // Una vez visto el contenido del dialogo, para no interferir con
                // Analisis.exe o con Filtrado.exe se liberan los objetos DirectSound
                // y DirectSoundCapture
                Ds.Liberar(true);
        }

        return;
}

void CEcualizadorDlg::OnCapsReproduccion()
{
        // Lo mismo que OnCapsCaptura pero para el dispositivo de reproducción de
        // DirectSound

        if (!g_Grabar)  {
                if (!g_DispositivoOk)   {
                        // Por si aún no se ha selccionado dispositivo
                        OnOpcionesDispositivos();
                        g_DispositivoOk=true;
                }
                Ds.CrearObjetos();

        // Los resultados están el la estructura Ds.dscaps
        // La estructura para analizar los resultados se ubicará en el código del
        // dialogo que mostrará los resultados fáclimente legibles

                CDialog_CapsReproduccion Dlg;
                Dlg.DoModal();

        // Una vez visto el contenido del dialogo, para no interferir con Analisis.exe
```

```
        // o con Filtrado.exe se liberan los objetos DirectSound y DirectSoundCapture
                Ds.Liberar(true);
        }

        return;
}

void CEcualizadorDlg::OnClose()
{
        // Antes de salir se libera aquí toda la memoria asignada dinámicamente, para
        // dejar todo como estaba...

        if (g_Grabar)   {
                // Si se está en proceso de grabación se termina
                g_TerminarHilo.SetEvent();
                Sleep(500);             // Para que pare la captura...
        }
        Ds.LiberarLista();
        Ds.Liberar(true);

        CDialog::OnClose();
}

void CEcualizadorDlg::OnOK()
{
        // La pulsacíón de Salir se corresponde con el mensaje de Ok, luego es paralela
        // a Close, luego gestionar aqui también la salida

        if (g_Grabar)   {
                // Si se está en proceso de grabación se termina
                g_TerminarHilo.SetEvent();
                Sleep(500);             // Para que pare la captura...
        }
        Ds.LiberarLista();
        Ds.Liberar(true);

        CDialog::OnOK();
}


void CEcualizadorDlg::OnGrabar()
{
        // Lanza e iniciliza todo lo necesario para poder capturar a disco la fuente
        // sonora seleccionada en la mesa de mezclas del sistema en un archivo con
        // nombre dado por el usuario en la casilla correspondiente

        int result;
        RECT rc;
```

```
// COMPROBAR QUE NO ESTA GRABANDO...
if (!g_Grabar)  {
        g_Error=0;

// Lanzamiento del diálogo predefinido de Güindous para que el usuario
// seleccione o proporcione el nombre del archivo a guardar y donde lo
// quiere hacer
        CFileDialog dlg(false, "wav", "*.wav", OFN_HIDEREADONLY
                | OFN_NOREADONLYRETURN | OFN_OVERWRITEPROMPT
                | OFN_NOCHANGEDIR, g_FiltroArch, NULL);

        if (dlg.DoModal() == IDOK)      {
                // Extraer el nombre y pasarselo al Objeto DirectSound para que
                // capture el audio y lo deje allí
                m_DisplayArchivo=dlg.GetPathName();
                Ds.m_NArchivo=(char *) malloc((m_DisplayArchivo.GetLength()+1) *
                                sizeof(char));
                for (int i=0; i<m_DisplayArchivo.GetLength(); i++)      {
                        *(Ds.m_NArchivo+i)=m_DisplayArchivo[i];
                }
                *(Ds.m_NArchivo+i)='\0';
                // Poner el nombre del archivo en el dialogo
                rc.top=318;
                rc.left=194;
                rc.bottom=343;
                rc.right=582;   // Solo repinta este trozo, y evitar parpadeos
                InvalidateRect(&rc, false);  // Fuerza el repintado del dialogo
        }
        else    {
                return;         // No hay nombre de archivo se sale
        }

        // Las opciones necesarias antes de proseguir

        // Lanzamiento de la selección de dispositivo
        if (!m_Dispositivos)    {
                OnOpcionesDispositivos();
                m_Dispositivos=true;
        }

        if (g_DispositivoOk)    {
                if (!g_primeraMuestreo)         {
                        Dlg_m.Inicializacion();
                        g_primeraMuestreo=true;
                }
        }
        else
```

```
                        return;     // Termina si no se introducen los datos pedidos

                g_TerminarHilo.ResetEvent();
                g_Grabar=true;

        // Preparacion y lanzamiento de la grabacion y de los objetos DirectSound que
        // conlleva
                result=Ds.CrearObjetos();
                if (result != 0)        {
                        g_Error=30;
                        PostMessage(WM_ERROR, 0, 0);
                        return;
                }
        // Comprobacion de la instalacion de los DirectX
                if (g_Error != 0)       {
                        PostMessage(WM_ERROR, 0, 0);
                        return;
                }
                // Inicializacion del Objeto DirectSound
                result=Ds.Configurar();
                if (result != 0)        {
                        g_Error=31;
                        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                        return;
                }

        // Segun el parametro que se le pase a InicializarBuffers, el Objeto Ds, creara
        // los eventos en el buffer de Captura (Analiss, Filtrado y Grabacion) o en el
        // secundario (Lectura de archivo WAV)

                result=Ds.InicializarBuffers(true);
                if (result != 0)        {
                        g_Error=32;
                        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                        return;
                }

                // Por último lanzar el audio para que comienze la captura y reproduccion,
                // cuando sea necesario
                AfxBeginThread(Hilo_LanzarAudio, Ds.m_Hwnd,
                               THREAD_PRIORITY_TIME_CRITICAL);
        }
        return;
}

void CEcualizadorDlg::OnParar()
{
        // Para detener la grabación
```

```
        g_TerminarHilo.SetEvent();
        return;
}

////////////////////////////////////////////////////////////////////////////////
//      Función global para ejecutar hilos, por requerimientos de Güindous
//
//
//

UINT Hilo_LanzarAudio(LPVOID pParam)
{
        // Funcion externa que envuelve a la funcion que realmente filtra
        // por requerimeintos de los hilos

        int result;

        result=Ds.LanzarCaptura();
        if (result != 0)
                PostMessage((HWND) pParam, WM_ERROR, 0, 0);
        // Si hay error tambien hay que parralo todo, asi que dejo que siga
        // la parada normal...
        // g_Error se asigna donde se produzca el error

        // Deja el Ojeto Ds preparado para lo siguiente
        Ds.Parar();
        Ds.Liberar(true);         // Evitar acaparar los recursos DirectSoundCapture

        g_TerminarHilo.ResetEvent();    // Resetea para la siguiente vez


        // Preparacion para siguiente lanzamiento, actuliza el estado de la aplicacion
        free(Ds.m_NArchivo);
        g_Grabar=false;

        return 0;
}



void CEcualizadorDlg::OnHelp()
{
        // Lanzamiento de la ayuda, lanza el menu general, para navegar desde él hasta
        // lo que buesque el usuario

        AfxGetApp()->WinHelp(0L, HELP_FINDER);
```

```
}
```

3.2.11 Estructuras.h

```
// Estructuras adicionales
// Almacena el GUID y la descripcion en estructuras de este tipo creando
// una lista simplemente enlazada
typedef struct tag_dsdevdesc
{
    GUID    guid;              // GUID del Dispositivo
    PSTR    lpDeviceDesc;      // Texto de Descripcion del dispositivo
    PSTR    lpDriverDesc;      // Texto del Driver del dispositivo
    struct tag_dsdevdesc *pNext;
} DSDEVICEDESC, *PDSDEVICEDESC;
```

3.3 Filtrado

Se procede a listar los archivos principales que forman en ejecutable Filtrado.exe.

3.3.1 Dialog_Error

Dialog_Error.h

```
#if !defined(AFX_DIALOG_ERROR_H__4EDC1AC5_4F60_11D5_871D_9C5ACA65E879__INCLUDED_)
#define AFX_DIALOG_ERROR_H__4EDC1AC5_4F60_11D5_871D_9C5ACA65E879__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Error.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error dialog

class CDialog_Error : public CDialog
{
// Construction
public:
        CDialog_Error(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialog_Error)
        enum { IDD = IDD_DIALOG_ERROR };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
```

```
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Error)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_Error)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_ERROR_H__4EDC1AC5_4F60_11D5_871D_9C5ACA65E879__INCLUDED_)
```

Dialog_Error.cpp

```
// Dialog_Error.cpp : implementation file
//

#include "stdafx.h"
#include "Filtrado.h"
#include "Dialog_Error.h"

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

int g_Error;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error dialog


CDialog_Error::CDialog_Error(CWnd* pParent /*=NULL*/)
        : CDialog(CDialog_Error::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDialog_Error)
```

```
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CDialog_Error::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDialog_Error)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialog_Error, CDialog)
        //{{AFX_MSG_MAP(CDialog_Error)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Error message handlers

BOOL CDialog_Error::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Acceso al texto del dialogo

        CString str;

        switch (g_Error)        {
        case 30:
        case 31:
        case 32:
        case 33:
        case 34:
        case 35:
        case 36:
        case 37:
        case 38:
        case 39:
        case 40:
        case 41:
                str="Fallo en DirectSound";
                break;
        case 10:
        case 11:
        case 12:
```

```
        case 13:
        case 14:
        case 15:
                str="Fallo en DirectDraw";
                break;
        case 61:
                str="Fallo en la configuración del filtro";
                break;
        case 90:
                str="No se puede leer la configuración. Debes lanzarlo desde
                     Ecuallizador.exe";
                break;
        case 100:
                str="Tienes unas DirectX antiguas, debes actualizar a versión 6.0 o
                     superior";
                break;
        default:
                str="Esto no debería salir nunca";
                break;
        }


        SetDlgItemText(IDC_STATIC_ERROR, str);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

3.3.2 Dialog_Filtrado

Dialog_Filtrado.h

```
#if !defined(AFX_DIALOG_FILTRADO_H__5D790323_4F99_11D5_871D_B3323415CA79__INCLUDED_)
#define AFX_DIALOG_FILTRADO_H__5D790323_4F99_11D5_871D_B3323415CA79__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Dialog_Filtrado.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialog_Filtrado dialog

class CDialog_Filtrado : public CDialog
{
// Construction
public:
```

```
        CDialog_Filtrado(CWnd* pParent = NULL);   // standard constructor
        Inicializacion();
        void Serialize(CFile *File, bool Todo);
// Dialog Data
        //{{AFX_DATA(CDialog_Filtrado)
        enum { IDD = IDD_DIALOG_FILTRADO };
        int             m_Cortes;
        int             m_Disco;
        int             m_Gcir;
        int             m_Escala;
        int             m_Personalizar;
        BOOL            m_BandasStereo;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialog_Filtrado)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
public:
        int  m_Cont;
        int  m_Filas;

protected:

        // Generated message map functions
        //{{AFX_MSG(CDialog_Filtrado)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

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

#endif // !defined(AFX_DIALOG_FILTRADO_H__5D790323_4F99_11D5_871D_B3323415CA79__INCLUDED_)
```

Dialog_Filtrado.cpp

```
// Dialog_Filtrado.cpp : implementation file
//

#include "stdafx.h"
#include "Filtrado.h"
```

```
#include "Dialog_Filtrado.h"
#include "Global.h"

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

extern CDirectS Ds;
extern CFiltro Filtro;

/////////////////////////////////////////////////////////////////////////////
// CDialog_Filtrado dialog
// En este programa solo se usa el objeto que controla el dialogo de las opciones
// de filtrado para acceder al disco y leer lo que se ha escrito el programa
// Ecualizador.exe, en el archivo correspondiente

CDialog_Filtrado::CDialog_Filtrado(CWnd* pParent /*=NULL*/)
	: CDialog(CDialog_Filtrado::IDD, pParent)
{
	//{{AFX_DATA_INIT(CDialog_Filtrado)
	m_Cortes = 0;
	m_Disco = FALSE;
	m_Personalizar = FALSE;
	m_Gcir = 0;
	m_Escala = -1;
	m_BandasStereo = FALSE;
	//}}AFX_DATA_INIT
}


void CDialog_Filtrado::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDialog_Filtrado)
	DDX_Text(pDX, IDC_CORTES, m_Cortes);
	DDV_MinMaxInt(pDX, m_Cortes, 3, 1000);
	DDX_Check(pDX, IDC_DISCO, m_Disco);
	DDX_Check(pDX, IDC_PERSONAL, m_Personalizar);
	DDX_Text(pDX, IDC_GCIR, m_Gcir);
	DDV_MinMaxInt(pDX, m_Gcir, 1, 10);
	DDX_Radio(pDX, IDC_LIN, m_Escala);
	DDX_Check(pDX, IDC_STEREO, m_BandasStereo);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CDialog_Filtrado, CDialog)
        //{{AFX_MSG_MAP(CDialog_Filtrado)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialog_Filtrado message handlers

CDialog_Filtrado::Inicializacion()
{
        // Coloca unos valores iniciales, para pruebas, y evitar la lectura de archivos
/*      m_Cortes=4;
        m_Gcir=3;
        m_BandasStereo=false;
        m_Disco=false;
        m_Personalizar=false;
        m_Escala=true;
*/
        return 0;
}

void CDialog_Filtrado::OnOK()
{
        // No se intercambian los datos antes de la llamada a la clase Base

        CDialog::OnOK();

        // Esta clase solo se usa para recoger los datos que ha generado Ecualizador.exe

        return;
}


void CDialog_Filtrado::Serialize(CFile *File, bool Todo)
{
        // Se limita a leer el archivo "FILTRADO.CFG" para obtener la información
        // necesaria que le permitirá ejecutar el modo filtrado con seguridad

        // Todo distingue el acceso que se hace desde el constructor para el tamaño
        // de la ventana y la lectura completa del archivo en la configuracion

        int i;

        CArchive Archivo(File, CArchive::load, 512);
        CObject::Serialize(Archivo);            // Llamada a la clase base
        if (!Archivo.IsStoring())       {
                Archivo >> m_Cortes >> m_Filas;
                if (Todo)       {
```

```
                        Filtro.m_Agrupacion=(int *) malloc(m_Cortes * sizeof(int));
                        Filtro.m_Frecuencias=(int *) malloc(m_Cortes * sizeof(int));
                        for (i=0; i<m_Cortes; i++)
                                Archivo >> *(Filtro.m_Frecuencias+i);
                        for (i=0; i<m_Cortes; i++)
                                Archivo >> *(Filtro.m_Agrupacion+i);
                        Archivo >> m_BandasStereo >> m_Gcir;
                }
        }

        return;
}
```

3.3.3 DirectS

DirectS.h

```
// DirectS.h: interface for the CDirectS class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
#define AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_

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

#include "Global.h"
#include "Estructuras.h"

class CDirectS
{
        // Variables
private:
        int m_Fs;               // Almacena la Frecuencia de muestreo com entero
        int m_Modo;             // Canales a tratar, Stereo(2) o Mono(1)
        int m_Bits;             // Bits por muestra 8 o 16

        int m_SubBuffer;         // Almacena cuanto mide el subbuffer
        int m_j, m_k, m_h, m_i;  // Contadores para FFT
        bool m_llenado;          // Para compensar los wraparound en bloqueo buffers

        LPDIRECTSOUND                   lpds;
        LPDIRECTSOUNDBUFFER             lpdsbPrimary, lpdsb;
        LPDIRECTSOUNDCAPTURE            lpdsc;
        LPDIRECTSOUNDCAPTUREBUFFER      lpdscb;
        LPDIRECTSOUNDNOTIFY             lpdsNotify;
```

```
        HANDLE                          rghEvent[NUMEVENTOS];
        DSBPOSITIONNOTIFY               rgdsbpn[NUMEVENTOS];
        DWORD                           m_TamBuffer;

        DWORD CursorLectura, CursorEscritura;   // Apuntan a los Buffers

public:
        HWND            m_Hwnd;             // Puntero a la ventana principal
        Opc_Muestreo    Muestreo;           // Recoje las opciones de muestreo
        GUID            guidDs, guidDsC;    // Estructuras con info de los Dispositivos

private:
        ProcesarEntrada(DWORD Posicion);
        TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total);
        TransferenciaFFT(LPBYTE Orig, DWORD Total);

public:
        CDirectS();
        virtual ~CDirectS();
        CrearObjetos();
        Configurar();
        InicializarBuffers(bool Captura);
        LanzarCaptura();
//      ReproducirArchivo();            // Coherencia del objeto
        Parar();
        Liberar(bool Todo);

};

#endif // !defined(AFX_DIRECTS_H__F8A65BD5_4924_11D5_871D_96D68E438279__INCLUDED_)
```

DirectS.cpp

```
// DirectS.cpp: implementation of the CDirectS class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Filtrado.h"
#include "DirectS.h"
#include "Global.h"

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

```
CEvent g_TerminarHilo;
extern CFiltro Filtro;

extern CFft Fft;
extern Opc_Muestreo Muestreo;
const double pi = 3.1415926535;

/////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////

CDirectS::CDirectS()
{
        // Inicializaciones adicionales; valores inicialies para punteros
        lpds=NULL;
        lpdsbPrimary=NULL;
        lpdsb=NULL;
        lpdsc=NULL;
        lpdscb=NULL;
        lpdsNotify=NULL;
}


CDirectS::~CDirectS()
{
}


CDirectS::Configurar()
{
        // Extrae los valores proporcionados por el usuario en el Dialogo_Muestreo y
        // los convierte a formatos deacuerdo con el Objeto Ds.
        // Pasa la Fs de CString a int, el Modo y Bits de bool a int

        char temp[10];
        int i=0;

        // Comprobacion de la entrada de datos
        if (Muestreo.m_Fs.IsEmpty())
                Muestreo.m_Fs="44100 Hz";

        for (i=0; i<Muestreo.m_Fs.GetLength() && Muestreo.m_Fs[i] !=' ' ; i++)  {
                temp[i]=Muestreo.m_Fs[i];
        }
        temp[i]='\0';
        m_Fs=atoi(temp);
        Muestreo.m_Bits==0 ? m_Bits=8 : m_Bits=16;
```

```
        Muestreo.m_Modo==0 ? m_Modo=1 : m_Modo=2;

        return 0;
}


CDirectS::CrearObjetos()
{
        // Creacion de los objetos con sus interfaces para su posterior uso
        // Solo se crean una vez por ejecucion de la aplicacion, al fianlizar se liberan

        m_Hwnd=AfxGetMainWnd()->GetSafeHwnd();
        if (lpds != NULL)       {
                lpds->Release();
                lpds=NULL;
        }
        // Creacion objeto DirectSound
        if (DS_OK != DirectSoundCreate(&guidDs, &lpds, NULL))   {
                // No se ha podido crear
                TRACE("Error en la creacion del DirectSound\n");
                return -1;
        }
        // No permite que suene nada más que esta aplicación EXCLUSIVE
        if (DS_OK != lpds->SetCooperativeLevel(m_Hwnd, DSSCL_EXCLUSIVE))        {
                // No se ha podido establecer la prioridad
                TRACE("ERROR: al establecer la prioridad\n");
                return -1;
        }

        if (lpdsc != NULL)      {
                lpdsc->Release();
                lpdsc=NULL;
        }
        // Creacion del objeto de Captura
        if (DS_OK != DirectSoundCaptureCreate(&guidDsC, &lpdsc, NULL)  {
                TRACE("Fue mal la creación de CDirectSoundCapture\n");
                return -1;
        }

        return 0;
}


CDirectS::InicializarBuffers(bool Captura)
{
        // Esta funcion crea los buffers necesarios para la aplicacion, es decir, se
        // crea lo necesario para un funcionamiento en FULL-DUPLEX, puesto que filtrado
        // lo requiere y Analisis si se quiere RuidoRosa tambien lo ha de implementar.
```

```
// Así se crean 2 buffers para Reproduccion y 1 para captura:
// Reproduccion: Primario, marca el formato en que se oirá y mezcla todos los
// secundarios que se estén reproduciendo, el secundario recibira los datos del
// de captura tratados o generados por el Ruido Rosa
// Captura: Unicamente recibe datos de la Tarjeta para que se puedan leer

WAVEFORMATEX            wfx;
DSBUFFERDESC            dsbd;
DSCBUFFERDESC           dscbd;

LPBYTE LpByte, LpByte2;
DWORD CuenByte2, CuenByte;

// Todos los buffers tendrán el mismo formato, puesto que evita sobrecarga de
// CPU y no es el propósito de la aplicacion convertir éstos
ZeroMemory(&wfx, sizeof(WAVEFORMATEX));
wfx.wFormatTag=WAVE_FORMAT_PCM;
wfx.nChannels=m_Modo;
wfx.nSamplesPerSec=m_Fs;
wfx.wBitsPerSample=m_Bits;
wfx.nBlockAlign=wfx.wBitsPerSample/8*wfx.nChannels;
wfx.nAvgBytesPerSec=wfx.nSamplesPerSec*wfx.nBlockAlign;

// Creacion del Buffer primario
ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize=sizeof(DSBUFFERDESC);
dsbd.dwFlags=DSBCAPS_PRIMARYBUFFER;
if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsbPrimary, NULL))       {
        TRACE("Error:Creacion del buffer Primario\n");
        return -1;
}
// Ajuste del formato deseado para B.Primario
if (DS_OK != lpdsbPrimary->SetFormat(&wfx))     {
        TRACE("ERROR:Ajuste del formato al buffer primario\n");
        return -1;
}
// Creacion del buffer de captura
// Cuentas para tamaño de los buffers
m_SubBuffer=max(4096, wfx.nAvgBytesPerSec/8);
m_SubBuffer -= m_SubBuffer % wfx.nBlockAlign;   // Quede Nž entero de mustras
                                                // en los buffers

ZeroMemory(&dscbd, sizeof(DSCBUFFERDESC));
dscbd.dwSize=sizeof(DSCBUFFERDESC);
dscbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
dscbd.lpwfxFormat=&wfx;

if (DS_OK != lpdsc->CreateCaptureBuffer(&dscbd, &lpdscb, NULL)) {
```

```
        TRACE("Error:Creacion del buffer de Captura\n");
        return -1;
}

// Creacion del buffer Secundario
// Debe ser exactamente igual que el de captura, porque entre ellos se pasarán
// los datos con o sin procesamiento
ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize=sizeof(DSBUFFERDESC);
dsbd.dwFlags=DSBCAPS_GETCURRENTPOSITION2        // Control sobre cursor
             | DSBCAPS_GLOBALFOCUS;
dsbd.dwBufferBytes=m_SubBuffer*NUMSUBBUFFERS;
dsbd.lpwfxFormat=&wfx;
if (DS_OK != lpds->CreateSoundBuffer(&dsbd, &lpdsb, NULL))      {
        TRACE("Error:Creacion Buffer Secundario\n");
        return -1;
}
// Por último llenar el buffer secundario de reproduccion con silencio, evita
// chispazos y picos indeseados
if (DS_OK != lpdsb->Lock(0, 0, (LPVOID*)&LpByte, &CuenByte,
        (LPVOID*)&LpByte2, &CuenByte2, DSBLOCK_ENTIREBUFFER))
        TRACE("Error al bloquear el buffer entero\n");
else    {
        if (m_Bits == 8)        {
                FillMemory(LpByte, CuenByte, 128);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 128);
        }
        else    {
                // Muestras de 16 Bits
                FillMemory(LpByte, CuenByte, 0);
                if (LpByte2 != NULL)
                        FillMemory(LpByte2, CuenByte2, 0);
        }
}
if (DS_OK != lpdsb->Unlock((LPVOID)LpByte, CuenByte, (LPVOID)LpByte2,
        CuenByte2))
        TRACE("Error al desbloquear el buffer entero\n");

m_TamBuffer=dsbd.dwBufferBytes;

// Creacion y configuracion de los eventos
// Tener encuenta quien pedirá eventos, captura (Analisis, Filtrado y Grabacion
// a archivo) o reproduccion (Lectura de archivo)
if (Captura)    {
        // Los Eventos se asociarán a la Captura
        for (int i=0; i<NUMEVENTOS; i++)        {
        rghEvent[i]=CreateEvent(NULL, false, false, NULL);
```

```
                if (NULL==rghEvent[i])
                        return false;
                }
                for (i=0; i<NUMEVENTOS; i++)    {
                        rgdsbpn[i].dwOffset=((m_SubBuffer*i)+m_SubBuffer-1);
                        rgdsbpn[i].hEventNotify=rghEvent[i];
                }

                //Asegura que lpdscNotify esta libre
                if (lpdsNotify != NULL) {
                        lpdsNotify->Release();
                        lpdsNotify=NULL;
                }
                if (DS_OK != lpdscb->QueryInterface(IID_IDirectSoundNotify,
                        (void **)&lpdsNotify))  {
                        g_Error=35;
                        return -1;
                }
                if (DS_OK != lpdsNotify->SetNotificationPositions(NUMEVENTOS, rgdsbpn)) {
                        lpdsNotify->Release();
                        g_Error=35;
                        return -1;
                }
                TRACE("SE HAN CREADO LOS EVENTOS\n");
                m_j=0;
                m_k=1653;
                m_h=1102;
                m_i=551;                // Para controlar los datos a FFT
                m_llenado=false;
                do      {                       // Espera a que este preparado
                                                // el objeto para la Fft
                }while (WaitForSingleObject(g_FftOk, 0) != WAIT_OBJECT_0);
                // Para que no lo reseteé despues de Chequearlo
                g_FftOk.SetEvent();
        }
        else    {
                // Aqui se configuran los eventos sobre el buffer secundario, cuando se
                // lean archivos WAV, para Análisis...

        }

        return 0;
}


CDirectS::LanzarCaptura()
{
        // Se centraliza en esta funcion, tanto la grabacion a disco, como el
```

```
// funcionamiento en FULL-DUPLEX de la aplicacion, puesto que los dos modos
// se basan en la captura y será esta la que comande todo la ejecucion en la
// gestion de los eventos según se van produciendo y atendiendo

if (g_Grabar)   {
        // En caso de que haya que grabar audio a un archivo, se inicializa aqui
        // todo lo necesario para acceder al disco, preparacion del archivo WAV

}
else    {
        // Para Analisis y Filtrado lanzar la Captura, y para evitar los chispazos
        // picos o chasquidos de paro-marcha, se coloca el buffer primario en
        // reproduccion continua, así, esta mezclando continuamente. Por arranque-
        // paro en el DMA del sistema
        if (DS_OK != lpdsbPrimary->Play(0, 0, DSBPLAY_LOOPING)) {
                TRACE("Error:Arranque del buffer Primario\n");
                g_Error=37;
                return -1;
        }
        if (DS_OK != lpdscb->Start(DSCBSTART_LOOPING))  {
                TRACE("Error: Arranque de la Captura\n");
                g_Error=36;
                return -1;
        }
        if (g_Filtrado)         {
                if (DS_OK != lpdsb->SetCurrentPosition(0))      {
                        TRACE("Error:Posicionamiento antes de reproducir\n");
                        g_Error=37;
                        return -1;
                }
                if (DS_OK != lpdsb->Play(0, 0, DSBPLAY_LOOPING))        {
                        TRACE("Error:Arranque del buffer Secundario\n");
                        g_Error=37;
                        return -1;
                }
        }
}

// Gestion de los eventos
// Inicializa el cursor escribir datos en B.Secundario
CursorEscritura=0xFFFFFFFF;

BOOL Done=false;
while(!Done)    {
        DWORD dwEvt=MsgWaitForMultipleObjects(
                NUMEVENTOS,             // Numero de Eventos
                rghEvent,               // Array donde están los Handles
                FALSE,                  // £Esperar a todos?
```

```
                        INFINITE,               // £Cuanto espero?
                        QS_ALLINPUT);           // £Cualquier Mensage es suceso?
                // Se normalizan los eventos; WAIT_OBJECT_0=0;
                dwEvt -= WAIT_OBJECT_0;
TRACE("                                                         dwEvt=%d\n",dwEvt);
                if (dwEvt < (DWORD) NUMEVENTOS) {
                        ProcesarEntrada(dwEvt);         // Copia datos al fichero o buffer
                }
                // Comprueba el evento de salida
                if (WaitForSingleObject(g_TerminarHilo, 0) == WAIT_OBJECT_0)    {
                        Done=true;              // Sale del bucle
                }
        }               // Final del procesamiento de Eventos
        return 0;
}

CDirectS::Parar()
{
        // Centraliza las órdenes de para de los buffers
        if (lpdscb != NULL)
                lpdscb->Stop();         // Para la Captura
        if (lpdsbPrimary != NULL)
                lpdsbPrimary->Stop();
        if (g_Filtrado) {
                if (lpdsb != NULL)
                        lpdsb->Stop();
        }
        return 0;
}

CDirectS::Liberar(bool Todo)
{
        // LLeva a cabo la tarea de la liberacion de todo; tanto los buffers solamente,
        // si se cambia el formato, u objetos y buffers si se procede a salir de la
        // aplicacion

        if ((lpdscb !=NULL))    {
                lpdscb->Release();
                lpdscb=NULL;
        }
        if (lpdsbPrimary != NULL)       {
                lpdsbPrimary->Release();
                lpdsbPrimary=NULL;
        }
        if (lpdsb != NULL)              {
                lpdsb->Release();
                lpdsb=NULL;
        }
```

```
        if (lpdsNotify != NULL) {
                lpdsNotify->Release();
                lpdsNotify=NULL;
        }

        if ((lpdsc != NULL) && Todo)    {
                lpdsc->Release();
                lpdsc=NULL;
        }
        if ((lpds != NULL) && Todo)             {
                lpds->Release();
                lpds=NULL;
        }
        return 0;
}

CDirectS::ProcesarEntrada(DWORD Posicion)
{
        // Gestion de los bloqueos de los buffers tanto de Captura como de reproduccion,
        // para pasar los datos de uno a otro, o bien directamente copiarlo a un archivo
        // .WAV, puesto que los eventos en ambos casos son del Buffer de Captura

        DWORD                   dwNumBytes;
        LPBYTE                  pbInput1, pbInput2, pbOutput1, pbOutput2;
        DWORD                   cbInput1, cbInput2, cbOutput1, cbOutput2;

        if (Posicion == 0)      {
                // Primer SubBuffer del Buffer
                CursorLectura=0;
                dwNumBytes=(rgdsbpn[Posicion].dwOffset)+1;
        }
        else    {
                // Resto de subdivisiones, en las que está compuesto el buffer
                CursorLectura=(rgdsbpn[Posicion-1].dwOffset)+1;
                dwNumBytes=m_SubBuffer;
                // rgdsbpn[Posicion].dwOffset - rgdsbpn[Posicion-1].dwOffset;
                // Otra solucion es esta calcular la diferencia entre subbuffers contiguos
        }
        // Bloqueo del buffer de Captura

// Nunca hay Wraparound porque las peticiones de bloqueo se hacen ajustandose
// al buffer de Captura, esta subdividido en 16 bloques y se seleccionan cada
// uno de éstos, por lo que pbInput2 y cbInput2 no sirven para nada

        if (DS_OK != lpdscb->Lock(CursorLectura, dwNumBytes,
                        (LPVOID *)&pbInput1, &cbInput1,
                        (LPVOID *)&pbInput2, &cbInput2, 0))     {
                TRACE("Error:Bloqueo Buffer de Captura\n");
```

```
                g_Error=38;
                return -1;
        }
        else    {
                if (g_Grabar)   {
                        // Solo tener en cuenta el buffer de captura

                }
                else if (g_Analisis)    {
                        // Calcular la FFT a partir de las muestras capturadas, asi que
                        // solo atender buffer de captura
                        TransferenciaFFT(pbInput1, cbInput1);
                }
                else    {
                        // Aquí entrará en modo Filtrado, lo cual implica una atencion a
                        // ambos buffers, leer del de Captura y escribir en el secundario
                        // Preparar el bloqueo del buffer secundario
                        if (CursorEscritura == 0xFFFFFFFF)      {
                    // La primera escritura, hay que saber por donde va reproducciendo
                    // y adelantarse un poco para que no nos pille poniendo datos
                                if (DS_OK != lpdsb->GetCurrentPosition(NULL,
                                             &CursorEscritura)) {
                                        TRACE("Error: Peticion de posicion en Buffer
                                               Secundario\n");
                                        g_Error=39;
                                }
                                else    {
                                        // Adelanta 2, 3 o 4 SubBuffers
                                        CursorEscritura+=(2*m_SubBuffer);
                                        while (CursorEscritura >= m_TamBuffer)
                                                CursorEscritura-=m_TamBuffer;
                                }
                        }
                        // Bloqueo del buffer Secundario (Reproducción)

// El hecho de escribir por primera vez tomando como referencia donde se encuentra
// el cursor de lectura hace que no concuerden los bloqueos del buffer secundario
// con el tamaño del buffer => Wraparound, tener encuenta pbOutput2 y cbOutput2

                        if (DS_OK != lpdsb->Lock(CursorEscritura, dwNumBytes,
                                        (LPVOID *)&pbOutput1, &cbOutput1,
                                        (LPVOID *)&pbOutput2, &cbOutput2, 0))   {
                                TRACE("Error:Bloqueo del buffer Secundario\n");
                                g_Error=39;
                        }
                        else    {
                    // Aquí los dos Buffers tienen una zona bloqueada cada uno, con lo
                    // cual es el momento de llevar los datos de Captura al Secundario,
```

```
                    // copiandolos o haciendo cosas con ellos por el camino
                                TransferenciaBuffers(pbOutput1, pbInput1, cbOutput1);
                                if (pbOutput2 != NULL)  {
                                        TRACE("Hay pbOutput2\n");
                                        TransferenciaBuffers(pbOutput2, pbInput1+cbOutput1,
                                                                cbOutput2);
                                }
                        }
                        // Desbloqueo del Buffer Secundario
                        if (DS_OK != lpdsb->Unlock(pbOutput1, cbOutput1, pbOutput2,
                                                cbOutput2)){
                                TRACE("Error:Desbloqueo del Buffer Secundario\n");
                                g_Error=41;
                        }
                        CursorEscritura+=dwNumBytes;    // cbInput1=dwNumBytes
                        while (CursorEscritura >= m_TamBuffer)
                                CursorEscritura-=m_TamBuffer;
                }
                if (DS_OK != lpdscb->Unlock(pbInput1, cbInput1, pbInput2, cbInput2))  {
                        TRACE("Error:Desbloqueo del Buffer de Captura\n");
                        g_Error=40;
                        return -1;
                }
                if (g_Error != 0)
                        // Problemas al desbloquear el buffer secundario, pero se
                        // comprueba aquí, para desbloquear el de captura y ya salir
                        return -1;
        }
        // Puesta en orden del Cursor de Lectura
        CursorLectura+=dwNumBytes;
        while (CursorLectura >= m_TamBuffer)
                CursorLectura-=m_TamBuffer;

        return 0;
}


CDirectS::TransferenciaFFT(LPBYTE Orig, DWORD Total)
{
        // Realiza las transferencias entre el buffer de captura y prepara los datos
        // para pasarlos al objeto que calcula la FFT.
        // Aqui no se reliza nunca esta tarea, asi que por eso no aparece el desarrollo
        // de la función, se implementa en Analisis.exe

        return 0;
}

CDirectS::TransferenciaBuffers(LPBYTE Dest, LPBYTE Orig, DWORD Total)
```

```
{
        // Realiza las transferencias entre buffers, con dos transformadas FFT y
        // filtrado entre medias.
        // Se realiza la FFT del T a F, se modifican las componentes en la frecuencia
        // como corresponde a un filtro y por último se calcula FFT de F a T y se
        // llevan los datos al buffer de salida

        // El cast no hace bien la conversión de datos de double a BYTE, asi que se
        // arregla con la funcion ceil(), aproxima un double por el entero menor que sea
        // mayor o igual que ese double. De esta manera lo hace bien

        register DWORD i;               // Contador
        DWORD *DOrig32, *DDest32;       // Para apuntar los buffer en accesos DWORD
        WORD  *WOrig16, *WDest16;       // Barrer la memoria con accesos WORD
        double normalizar=Fft.m_n*4;    // Evita otra multiplicacion en el bucle interno

        if (m_Bits == 8)        {
                if (m_Modo == 1)        {
                        // 8 Bits y Mono
                        for (i=0; i<Total; i++) {
                                // Aqui el ajuste con rfftw ................
                                *((Fft.m_RTCInR)+m_j)=*Orig;
                                *((Fft.m_RTCInR_1)+m_k)=*Orig;
                                *((Fft.m_RTCInR_2)+m_h)=*Orig;
                                *((Fft.m_RTCInR_3)+m_i)=*Orig++;

                                *Dest++=(unsigned char)ceil((((*((Fft.m_CTROutR)+m_j)))+
                                        (*((Fft.m_CTROutR_1)+m_k))+(*((Fft.m_CTROutR_2)+
                                        m_h))+(*((Fft.m_CTROutR_3)+m_i)))/(double)
                                        normalizar);

                                m_k++;
                                m_h++;
                                m_i++;
                                if (m_k == Fft.m_n )    {
                                        if (m_llenado)  {
                                                Fft.CalcularTF(Fft.m_RTCInR_1,
                                                               Fft.m_RTCOutR_1);
                                                Filtro.FiltrarR(Fft.m_RTCOutR_1,
                                                                Fft.m_CTRInR_1);
                                                Fft.CalcularFT(Fft.m_CTRInR_1,
                                                               Fft.m_CTROutR_1);
                                        }
                                        m_k=0;
                                }
                                if (m_h == Fft.m_n)     {
                                        if (m_llenado)          {
                                                Fft.CalcularTF(Fft.m_RTCInR_2,
```

```
                                                Fft.m_RTCOutR_2);
                                        Filtro.FiltrarR(Fft.m_RTCOutR_2,
                                                         Fft.m_CTRInR_2);
                                        Fft.CalcularFT(Fft.m_CTRInR_2,
                                                Fft.m_CTROutR_2);
                                }
                                m_h=0;
                        }
                        if (m_i == Fft.m_n)     {
                                if (m_llenado)          {
                                        Fft.CalcularTF(Fft.m_RTCInR_3,
                                                Fft.m_RTCOutR_3);
                                        Filtro.FiltrarR(Fft.m_RTCOutR_3,
                                                         Fft.m_CTRInR_3);
                                        Fft.CalcularFT(Fft.m_CTRInR_3,
                                                Fft.m_CTROutR_3);
                                }
                                m_i=0;
                        }
                        m_j++;
                        if (m_j == Fft.m_n)     {
                                Fft.CalcularTF(Fft.m_RTCInR,
                                        Fft.m_RTCOutR);
                                Filtro.FiltrarR(Fft.m_RTCOutR,
                                         Fft.m_CTRInR);
                                Fft.CalcularFT(Fft.m_CTRInR,
                                        Fft.m_CTROutR);
                                m_j=0;
                                m_llenado=true;
                        }
                }
        }
        else    {
                // 8 Bits y Stereo
                WDest16=(LPWORD)Dest;
                WOrig16=(LPWORD)Orig;
                Total=Total>>1;                 // Transferencias de 16 bits
                for(i=0; i<Total; i++)  {
                        // Ajuste con rfftw...
                        *((Fft.m_RTCInR)+m_j)=(unsigned char)(*(LPBYTE)WOrig16);
                        *((Fft.m_RTCInL)+m_j)=(unsigned char)(*(LPBYTE)((LPBYTE)
                                        WOrig16+1));
                        *((Fft.m_RTCInR_1)+m_k)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInR_2)+m_h)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInR_3)+m_i)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInL_1)+m_k)=*((Fft.m_RTCInL)+m_j);
                        *((Fft.m_RTCInL_2)+m_h)=*((Fft.m_RTCInL)+m_j);
                        *((Fft.m_RTCInL_3)+m_i)=*((Fft.m_RTCInL)+m_j);
```

```
        WOrig16++;

(*(LPBYTE)WDest16)=(unsigned char)ceil((((*((Fft.m_CTROutR)+
 m_j)))+(*((Fft.m_CTROutR_1)+m_k))+(*((Fft.m_CTROutR_2)+
 m_h))+(*((Fft.m_CTROutR_3)+m_i)))/(double)normalizar);
(*(LPBYTE)((LPBYTE)WDest16+1))=(unsigned char)ceil((((*((Fft.
 m_CTROutL)+m_j)))+(*((Fft.m_CTROutL_1)+m_k))+(*((Fft.
 m_CTROutL_2)+m_h))+(*((Fft.m_CTROutL_3)+m_i)))/(double)
 normalizar);
        WDest16++;

        m_k++;
        m_h++;
        m_i++;
        if (m_k == Fft.m_n)     {
                if (m_llenado)          {
                        Fft.CalcularTF(Fft.m_RTCInR_1,
                                        Fft.m_RTCOutR_1);
                        Fft.CalcularTF(Fft.m_RTCInL_1,
                                        Fft.m_RTCOutL_1);
                        Filtro.FiltrarR(Fft.m_RTCOutR_1,
                                         Fft.m_CTRInR_1);
                        Filtro.FiltrarL(Fft.m_RTCOutL_1,
                                         Fft.m_CTRInL_1);
                        Fft.CalcularFT(Fft.m_CTRInR_1,
                                        Fft.m_CTROutR_1);
                        Fft.CalcularFT(Fft.m_CTRInL_1,
                                        Fft.m_CTROutL_1);
                }
                m_k=0;
        }
        if (m_h == Fft.m_n)     {
                if (m_llenado)          {
                        Fft.CalcularTF(Fft.m_RTCInR_2,
                                        Fft.m_RTCOutR_2);
                        Fft.CalcularTF(Fft.m_RTCInL_2,
                                        Fft.m_RTCOutL_2);
                        Filtro.FiltrarR(Fft.m_RTCOutR_2,
                                         Fft.m_CTRInR_2);
                        Filtro.FiltrarL(Fft.m_RTCOutL_2,
                                         Fft.m_CTRInL_2);
                        Fft.CalcularFT(Fft.m_CTRInR_2,
                                        Fft.m_CTROutR_2);
                        Fft.CalcularFT(Fft.m_CTRInL_2,
                                        Fft.m_CTROutL_2);
                }
                m_h=0;
        }
```

```
                        if (m_i == Fft.m_n)     {
                                if (m_llenado)          {
                                        Fft.CalcularTF(Fft.m_RTCInR_3,
                                                        Fft.m_RTCOutR_3);
                                        Fft.CalcularTF(Fft.m_RTCInL_3,
                                                        Fft.m_RTCOutL_3);
                                        Filtro.FiltrarR(Fft.m_RTCOutR_3,
                                                         Fft.m_CTRInR_3);
                                        Filtro.FiltrarL(Fft.m_RTCOutL_3,
                                                         Fft.m_CTRInL_3);
                                        Fft.CalcularFT(Fft.m_CTRInR_3,
                                                        Fft.m_CTROutR_3);
                                        Fft.CalcularFT(Fft.m_CTRInL_3,
                                                        Fft.m_CTROutL_3);
                                }
                                m_i=0;
                        }
                        m_j++;
                        if (m_j == Fft.m_n)     {
                                Fft.CalcularTF(Fft.m_RTCInR,
                                               Fft.m_RTCOutR);
                                Fft.CalcularTF(Fft.m_RTCInL,
                                               Fft.m_RTCOutL);
                                Filtro.FiltrarR(Fft.m_RTCOutR,
                                                Fft.m_CTRInR);
                                Filtro.FiltrarL(Fft.m_RTCOutL,
                                                Fft.m_CTRInL);
                                Fft.CalcularFT(Fft.m_CTRInR,
                                               Fft.m_CTROutR);
                                Fft.CalcularFT(Fft.m_CTRInL,
                                               Fft.m_CTROutL);
                                m_j=0;
                                m_llenado=true;
                        }
                }
        }
}
else    {
        if (m_Modo == 1)        {
                // 16 Bits y Mono
                WDest16=(LPWORD)Dest;
                WOrig16=(LPWORD)Orig;
                Total=Total>>1;         // Transferencias de 16 bits
                for (i=0; i<Total; i++) {
                        // Aqui el ajuste con rfftw ................

                        *((Fft.m_RTCInR)+m_j)=(short int)((*WOrig16));
                        *((Fft.m_RTCInR_1)+m_k)=(short int)((*WOrig16));
```

```
*((Fft.m_RTCInR_2)+m_h)=(short int)((*WOrig16));
*((Fft.m_RTCInR_3)+m_i)=(short int)((*WOrig16++));

*WDest16++=(short int)ceil((((*((Fft.m_CTROutR)+m_j))+
   (*((Fft.m_CTROutR_1)+m_k))+(*((Fft.m_CTROutR_2)+m_h))
   +(*((Fft.m_CTROutR_3)+m_i))))/(double)normalizar);

m_k++;
m_h++;
m_i++;
if (m_k == Fft.m_n)     {
        if (m_llenado)          {
                Fft.CalcularTF(Fft.m_RTCInR_1,
                               Fft.m_RTCOutR_1);
                Filtro.FiltrarR(Fft.m_RTCOutR_1,
                                Fft.m_CTRInR_1);
                Fft.CalcularFT(Fft.m_CTRInR_1,
                               Fft.m_CTROutR_1);
        }
        m_k=0;
}
if (m_h == Fft.m_n)     {
        if (m_llenado)          {
                Fft.CalcularTF(Fft.m_RTCInR_2,
                               Fft.m_RTCOutR_2);
                Filtro.FiltrarR(Fft.m_RTCOutR_2,
                                Fft.m_CTRInR_2);
                Fft.CalcularFT(Fft.m_CTRInR_2,
                               Fft.m_CTROutR_2);
        }
        m_h=0;
}
if (m_i == Fft.m_n)     {
        if (m_llenado)          {
                Fft.CalcularTF(Fft.m_RTCInR_3,
                               Fft.m_RTCOutR_3);
                Filtro.FiltrarR(Fft.m_RTCOutR_3,
                                Fft.m_CTRInR_3);
                Fft.CalcularFT(Fft.m_CTRInR_3,
                               Fft.m_CTROutR_3);
        }
        m_i=0;
}
m_j++;
if (m_j == Fft.m_n)     {
        Fft.CalcularTF(Fft.m_RTCInR,
                       Fft.m_RTCOutR);
        Filtro.FiltrarR(Fft.m_RTCOutR,
```

```
                                        Fft.m_CTRInR);
                                Fft.CalcularFT(Fft.m_CTRInR,
                                               Fft.m_CTROutR);
                                m_llenado=true;
                                m_j=0;
                        }
                }
        }
        else    {
                // 16 Bits y Stereo
                DDest32=(LPDWORD)Dest;
                DOrig32=(LPDWORD)Orig;
                Total=Total>>2 ;     // Entre 4, para transferencias 32 Bits
                for (i=0; i<Total; i++) {
                        // Ajuste con rfftw
                        *((Fft.m_RTCInR)+m_j)=(short int)(*(LPWORD)DOrig32);
                        *((Fft.m_RTCInL)+m_j)=(short int)(*(LPWORD)((LPWORD)
                                                DOrig32+1));
                        *((Fft.m_RTCInR_1)+m_k)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInR_2)+m_h)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInR_3)+m_i)=*((Fft.m_RTCInR)+m_j);
                        *((Fft.m_RTCInL_1)+m_k)=*((Fft.m_RTCInL)+m_j);
                        *((Fft.m_RTCInL_2)+m_h)=*((Fft.m_RTCInL)+m_j);
                        *((Fft.m_RTCInL_3)+m_i)=*((Fft.m_RTCInL)+m_j);
                        DOrig32++;

                        (*(LPWORD)DDest32)=(short int)ceil((((*((Fft.m_CTROutR)+
                           m_j))+(*((Fft.m_CTROutR_1)+m_k))+(*((Fft.m_CTROutR_2)+
                           m_h))+(*((Fft.m_CTROutR_3)+m_i))))/(double)normalizar);
                        (*(LPWORD)((LPWORD)DDest32+1))=(short int)ceil((((*((Fft.
                           m_CTROutL)+m_j))+(*((Fft.m_CTROutL_1)+m_k))+(*((Fft.
                           m_CTROutL_2)+m_h))+(*((Fft.m_CTROutL_3)+m_i))))/
                           (double)normalizar);
                        DDest32++;

                        m_k++;
                        m_h++;
                        m_i++;
                        if (m_k == Fft.m_n)     {
                                if (m_llenado)          {
                                        Fft.CalcularTF(Fft.m_RTCInR_1,
                                                       Fft.m_RTCOutR_1);
                                        Fft.CalcularTF(Fft.m_RTCInL_1,
                                                       Fft.m_RTCOutL_1);
                                        Filtro.FiltrarR(Fft.m_RTCOutR_1,
                                                        Fft.m_CTRInR_1);
                                        Filtro.FiltrarL(Fft.m_RTCOutL_1,
                                                        Fft.m_CTRInL_1);
```

```
                Fft.CalcularFT(Fft.m_CTRInR_1,
                               Fft.m_CTROutR_1);
                Fft.CalcularFT(Fft.m_CTRInL_1,
                               Fft.m_CTROutL_1);
        }
        m_k=0;
}
if (m_h == Fft.m_n)     {
        if (m_llenado)          {
                Fft.CalcularTF(Fft.m_RTCInR_2,
                               Fft.m_RTCOutR_2);
                Fft.CalcularTF(Fft.m_RTCInL_2,
                               Fft.m_RTCOutL_2);
                Filtro.FiltrarR(Fft.m_RTCOutR_2,
                                Fft.m_CTRInR_2);
                Filtro.FiltrarL(Fft.m_RTCOutL_2,
                                Fft.m_CTRInL_2);
                Fft.CalcularFT(Fft.m_CTRInR_2,
                               Fft.m_CTROutR_2);
                Fft.CalcularFT(Fft.m_CTRInL_2,
                               Fft.m_CTROutL_2);
        }
        m_h=0;
}
if (m_i == Fft.m_n)     {
        if (m_llenado)          {
                Fft.CalcularTF(Fft.m_RTCInR_3,
                               Fft.m_RTCOutR_3);
                Fft.CalcularTF(Fft.m_RTCInL_3,
                               Fft.m_RTCOutL_3);
                Filtro.FiltrarR(Fft.m_RTCOutR_3,
                                Fft.m_CTRInR_3);
                Filtro.FiltrarL(Fft.m_RTCOutL_3,
                                Fft.m_CTRInL_3);
                Fft.CalcularFT(Fft.m_CTRInR_3,
                               Fft.m_CTROutR_3);
                Fft.CalcularFT(Fft.m_CTRInL_3,
                               Fft.m_CTROutL_3);
        }
        m_i=0;
}
m_j++;
if (m_j == Fft.m_n)     {
        Fft.CalcularTF(Fft.m_RTCInR,
                       Fft.m_RTCOutR);
        Fft.CalcularTF(Fft.m_RTCInL,
                       Fft.m_RTCOutL);
        Filtro.FiltrarR(Fft.m_RTCOutR,
```

```
                                                        Fft.m_CTRInR);
                                        Filtro.FiltrarL(Fft.m_RTCOutL,
                                                        Fft.m_CTRInL);
                                        Fft.CalcularFT(Fft.m_CTRInR,
                                                       Fft.m_CTROutR);
                                        Fft.CalcularFT(Fft.m_CTRInL,
                                                       Fft.m_CTROutL);
                                        m_j=0;
                                        m_llenado=true;
                                }
                        }
                }
        }
        return 0;
}
```

3.3.4 Fft

Fft.h

```
// Fft.h: interface for the CFft class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_)
#define AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_

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

#include "Global.h"

class CFft
{
public:
        int DestruirPlan();
        int CalcularTF(fftw_real *In, fftw_real *Out);
        int CalcularFT(fftw_real *In, fftw_real *Out);
        int CrearPlan();
        CFft();
        virtual ~CFft();

private:
        //fftw_real *DatosIn, *DatosOut;
        rfftw_plan m_p, m_q;
        CCriticalSection m_Csfft;

public:
```

```
        fftw_real *m_RTCInR, *m_RTCOutR;        // Para el canal de la Derecha (R)
        fftw_real *m_RTCInL, *m_RTCOutL;        // Para el canal de la Izquierda (L)
        fftw_real *m_CTRInR, *m_CTROutR;
        fftw_real *m_CTRInL, *m_CTROutL;
        fftw_real *m_RTCInR_1, *m_RTCOutR_1;    // Para el solapamiento de las
        fftw_real *m_RTCInL_1, *m_RTCOutL_1;    // distintas FFT
        fftw_real *m_CTRInR_1, *m_CTROutR_1;
        fftw_real *m_CTRInL_1, *m_CTROutL_1;
        fftw_real *m_RTCInR_2, *m_RTCOutR_2;
        fftw_real *m_RTCInL_2, *m_RTCOutL_2;
        fftw_real *m_CTRInR_2, *m_CTROutR_2;
        fftw_real *m_CTRInL_2, *m_CTROutL_2;
        fftw_real *m_RTCInR_3, *m_RTCOutR_3;
        fftw_real *m_RTCInL_3, *m_RTCOutL_3;
        fftw_real *m_CTRInR_3, *m_CTROutR_3;
        fftw_real *m_CTRInL_3, *m_CTROutL_3;    // Solapamiento

        int m_n;                                // La precision de la transformada
        BOOL m_PlanTF, m_PlanFT;

};

#endif // !defined(AFX_FFT_H__9E267C81_E87F_11D4_8F4D_444553540000__INCLUDED_)
```

Fft.cpp

```
// Fft.cpp: implementation of the CFft class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Fft.h"
#include "Global.h"

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

CEvent g_FftOk;

extern CDirectS Ds;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
```

```
CFft::CFft()
{
        m_PlanFT=false;
        m_PlanTF=false;
        m_RTCInR=NULL;
        m_RTCOutR=NULL;
        m_RTCInL=NULL;
        m_RTCOutL=NULL;
        m_CTRInR=NULL;
        m_CTROutR=NULL;
        m_CTRInL=NULL;
        m_CTROutL=NULL;
}

CFft::~CFft()
{
}


int CFft::CrearPlan()
{
        // Antes de realizar el cálculo de la Real FFT hay que preparar las estructuras
        // necesarias antes de realizar la transformada.

        // fftw_real in[N], out[N], power_spectrum[N/2+1];
        // Se crean los arrays que contedran la entrada y salida de datos con el
        // objeto puesto que se lanzará con el hilo que lo ejecuta y si cambia
        // se destruye el hilo y se vuelve a crear con la nueva configuracion

        m_n=2205;               // 2204 componentes y DC

        // Prototipo:
        // rfftw_plan rfftw_create_plan(int n, fftw_direction dir, int flags);

    //m_p = rfftw_create_plan(m_n, FFTW_REAL_TO_COMPLEX, FFTW_MEASURE);

    // Los flags son: FFTW_MEASURE or FFTW_ESTIMATE
        // La direccion: FFTW_REAL_TO_COMPLEX or FFTW_COMPLEX_TO_REAL

        // Prototipo:
        // rfftw_plan rfftw_create_plan_specific(int n, fftw_direction dir, int flags,
        //              fftw_real *in, int istride, fftw_real *out, int ostride);

        if (!m_PlanTF)  {
                g_CreandoPlan=2;
                m_RTCInR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
                m_RTCOutR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
```

```
        m_RTCInL=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutL=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        m_RTCInR_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutR_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCInR_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutR_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCInR_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutR_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        m_RTCInL_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutL_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCInL_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutL_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCInL_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_RTCOutL_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        // Se crea el plan de conversion en la direccion Tiempo -> frecuencia
        m_p=rfftw_create_plan_specific(m_n, FFTW_REAL_TO_COMPLEX,
                FFTW_ESTIMATE, m_RTCInR, 1, m_RTCOutR, 1);
        m_PlanTF=true;
        TRACE("SE HA CREADO PLAN T->F\n");

}
if (!m_PlanFT)          {
        g_CreandoPlan=2;
        m_CTRInR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTRInL=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutR=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutL=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        m_CTRInR_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutR_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTRInR_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutR_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTRInR_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutR_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        m_CTRInL_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutL_1=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTRInL_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutL_2=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTRInL_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));
        m_CTROutL_3=(fftw_real*) malloc(m_n * sizeof(fftw_real));

        // Se crea el plan de conversion de Frecuencia -> Tiempo
        m_q=rfftw_create_plan_specific(m_n, FFTW_COMPLEX_TO_REAL,
                FFTW_ESTIMATE, m_RTCInR, 1, m_RTCOutR, 1);
```

```
        m_PlanFT=true;
        TRACE("SE HA CREADO PLAN F->T\n");

}

// Para evitar posibles chispazos en el buffer de salida, puesto que los
// buffers se rellenan con silencios, los primeros 2205 se copian antes de
// calcular ninguna FFT, es necesario poner silencio, pero deacuerdo con
// el buffer, el tipo de datos que lleve; 8 ó 16 bits

for (int i=0; i<m_n; i++)       {
        if (Ds.Muestreo.m_Bits) {
                // Datos de 16 bits=> Ca2 luego 0
                *(m_CTROutR+i)=0;
                *(m_CTROutL+i)=0;
                *(m_CTROutR_1+i)=0;
                *(m_CTROutL_1+i)=0;
                *(m_CTROutR_2+i)=0;
                *(m_CTROutL_2+i)=0;
                *(m_CTROutR_3+i)=0;
                *(m_CTROutL_3+i)=0;

        }
        else    {
                // Datos de 8 bits, poner 128 que esta el 0
                *(m_CTROutR+i)=128*m_n;
                *(m_CTROutL+i)=128*m_n;
                *(m_CTROutR_1+i)=128*m_n;
                *(m_CTROutL_1+i)=128*m_n;
                *(m_CTROutR_2+i)=128*m_n;
                *(m_CTROutL_2+i)=128*m_n;
                *(m_CTROutR_3+i)=128*m_n;
                *(m_CTROutL_3+i)=128*m_n;
        }
        // Entrada a (FFT)^-1 inicializada a 0's para evitar los picos hasta
        // que se llenen todas ellas
        *(m_CTRInR_1+i)=0;
        *(m_CTRInR_2+i)=0;
        *(m_CTRInR_3+i)=0;
        *(m_CTRInL_1+i)=0;
        *(m_CTRInL_2+i)=0;
        *(m_CTRInL_3+i)=0;
}

g_FftOk.SetEvent();
g_CreandoPlan=false;

g_Error=0;
```

```
        return 0;
}


int CFft::CalcularTF(fftw_real *In, fftw_real *Out)
{
        // Esta funcion lanza la rutina encargada de transformar los datos
        // Realiza el calculo del dominio del tiempo a frecuencia
        // Prototipo:
        // void rfftw_one(rfftw_plan plan, fftw_real *in, fftw_real *out);

        // *In, *Out;   // Punteros a los arrays

        m_Csfft.Lock();                         // Entra en seccion crítica
        rfftw_one(m_p, In, Out);
        m_Csfft.Unlock();                       // Sale de la seccion critica

        return 0;
}

int CFft::CalcularFT(fftw_real *In, fftw_real *Out)
{
        // En esta funcion se calcula la FFt del dominio de la frecuencia al tiempo

        m_Csfft.Lock();
        rfftw_one(m_q, In, Out);
        m_Csfft.Unlock();

        return 0;
}

int CFft::DestruirPlan()
{
        // Antes de terminar hay que destruir el plan creado
        rfftw_destroy_plan(m_p);
        rfftw_destroy_plan(m_q);

        // Liberacion de la memoria asignada dinámicamente

        free(m_RTCInR);
        free(m_RTCOutR);
        free(m_RTCInL);
        free(m_RTCOutL);
        free(m_CTRInR);
        free(m_CTROutR);
        free(m_CTRInL);
        free(m_CTROutL);
```

```
        free(m_RTCInR_1);
        free(m_RTCInR_2);
        free(m_RTCInR_3);
        free(m_RTCInL_1);
        free(m_RTCInL_2);
        free(m_RTCInL_3);
        free(m_RTCOutR_1);
        free(m_RTCOutR_2);
        free(m_RTCOutR_3);
        free(m_RTCOutL_1);
        free(m_RTCOutL_2);
        free(m_RTCOutL_3);

        free(m_CTRInR_1);
        free(m_CTRInR_2);
        free(m_CTRInR_3);
        free(m_CTRInL_1);
        free(m_CTRInL_2);
        free(m_CTRInL_3);
        free(m_CTROutR_1);
        free(m_CTROutR_2);
        free(m_CTROutR_3);
        free(m_CTROutL_1);
        free(m_CTROutL_2);
        free(m_CTROutL_3);

        return 0;
}
```

3.3.5 FiltradoView

FiltradoView.h

```
// FiltradoView.h : interface of the CFiltradoView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTRADOVIEW_H__104E5544_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_FILTRADOVIEW_H__104E5544_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_

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

class CFiltradoView : public CScrollView
{
protected: // create from serialization only
        CFiltradoView();
```

```
        DECLARE_DYNCREATE(CFiltradoView)

// Attributes
public:
        CFiltradoDoc* GetDocument();

private:
        CBitmap *m_BmpDeslizador, *m_BmpJog, *m_BmpMute, *m_BmpSolo;    // Cargar BMP's
        CDC     *m_dcMemDes, *m_dcMemJog, *m_dcMemMute, *m_dcMemSolo;   // BMP's en memoria
        CFont Fuente;
        bool m_Captura, m_FilaSup;

        const CSize m_sizeJog;          // Tamaño del Jog (Lógicas)
        CPoint m_pointTopLeft;          // Esquina sup. Izq del Jog (Lógicas)
        CSize m_Offset;                 // Offset con el raton en captura (Dispositivo)
        HCURSOR m_hCursor;              // Para nuevo puntero de ratón

        int m_Iden, m_IdenL;            // Almacena que barra ha sido movida
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFiltradoView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual void OnInitialUpdate(); // called first time after construct
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CFiltradoView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CFiltradoView)
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

```
};

#ifndef _DEBUG  // debug version in FiltradoView.cpp
inline CFiltradoDoc* CFiltradoView::GetDocument()
   { return (CFiltradoDoc*)m_pDocument; }
#endif

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

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

#endif // !defined(AFX_FILTRADOVIEW_H__104E5544_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

FiltradoView.cpp

```
// FiltradoView.cpp : implementation of the CFiltradoView class
//

#include "stdafx.h"
#include "Filtrado.h"
#include "FiltradoDoc.h"
#include "FiltradoView.h"
#include "Global.h"

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

// Objetos Globales
CFiltro Filtro;
extern CDirectS Ds;
extern bool g_AccesoGDI;

static char Cursor_str[]="ID_ARRASTRE";         // Para manipular el puntero del ratón
                                                // que se ha añadido
/////////////////////////////////////////////////////////////////////////////
// CFiltradoView

IMPLEMENT_DYNCREATE(CFiltradoView, CScrollView)

BEGIN_MESSAGE_MAP(CFiltradoView, CScrollView)
        //{{AFX_MSG_MAP(CFiltradoView)
        ON_WM_LBUTTONDOWN()
        ON_WM_LBUTTONUP()
```

```
        ON_WM_MOUSEMOVE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFiltradoView construction/destruction

CFiltradoView::CFiltradoView(): m_Offset(0, 0),
                                                m_sizeJog(18,28),
                                                m_pointTopLeft(10,60)
// Es necesaria esta inicializacion, para comenzar el rastreo de los Jogs, llevado
// a cabo en la funcion OnLButtonDown(), pues ha de comenzar por algún sitio, así
// que se le asigna el primero.
{
        // Inicializaciones adicionales, los objetos se crean todos dinamicamente
        // y duarán lo mismo que el objeto que controla la vista, se asocia su
        // ciclo de vida con el constructor y destructor del objeto

        m_BmpDeslizador=new CBitmap;
        m_BmpJog=new CBitmap;
        m_dcMemDes=new CDC;
        m_dcMemJog=new CDC;
        m_Captura=false;

        m_BmpMute=new CBitmap;
        m_BmpSolo=new CBitmap;
        m_dcMemMute=new CDC;
        m_dcMemSolo=new CDC;

        m_Iden=2000;
        m_IdenL=2000;   // Para que la primera vez que los pinte sea en reposo, y use
                        // las coordenadas de los arrays

        m_hCursor=AfxGetApp()->LoadCursor(IDC_MANO);
}

CFiltradoView::~CFiltradoView()
{
        // Limpiar los objetos adicionales
        delete m_BmpDeslizador;
        delete m_BmpJog;
        delete m_dcMemDes;
        delete m_dcMemJog;
        delete m_BmpMute;
        delete m_BmpSolo;
        delete m_dcMemMute;
        delete m_dcMemSolo;
}
```

```
BOOL CFiltradoView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs
        // Pone el fondo negro

        cs.lpszClass=AfxRegisterWndClass(CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW,
                NULL,::CreateSolidBrush(RGB(0,0,0)));

        return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CFiltradoView drawing

void CFiltradoView::OnDraw(CDC* pDC)
{
        // Puesta en pantalla de los deslizadores y todo lo que lleva asociado para el
        // equalizador. Güindous cuando quiere repintar la vista viene aquí...

        LONG x, y;
        CString str;                      // Para escribir el valor de la Frecuencia
        CPoint offset=GetDeviceScrollPosition();

        x=0;
        y=0;    // Contadores para almacenar offsets

        if (g_AccesoGDI)                {
                for (int i=0; i<Filtro.m_Cortes+1; i++) {
                        m_dcMemDes->SelectObject(m_BmpDeslizador);
                        pDC->BitBlt(x, y, 58, 175, m_dcMemDes, 0, 0, SRCCOPY);
                        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                                // Si hay tambien Deslizadores abajo, Hay que pintarlos
                                pDC->BitBlt(x, y+175, 58, 175, m_dcMemDes, 0, 0, SRCCOPY);
                        }

                        m_dcMemJog->SelectObject(m_BmpJog);
                        pDC->BitBlt(i != m_Iden ? (Filtro.m_RefJog+i)->x : m_pointTopLeft.
                                x, i != m_Iden ? (Filtro.m_RefJog+i)->y : m_pointTopLeft.y,
                                18, 28, m_dcMemJog, 0, 0, SRCCOPY);
                        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                                // Si hay tambien Deslizadores abajo, Hay que pintarlos
                                pDC->BitBlt(i != m_IdenL ? (Filtro.m_RefJogL+i)->x :
                                        m_pointTopLeft.x, i != m_IdenL ? (Filtro.
                                        m_RefJogL+i)->y : m_pointTopLeft.y, 18, 28,
                                        m_dcMemJog, 0, 0, SRCCOPY);
                        }
```

```
// Con esas estructuras de decision ?, se comprueba cual se mueve o si esta
// parada para que se repinten siempre correctamente

                if (i == 0)     {
                        // Para pintar el/los masters
                        str.Format("Vol.Principal");
                        pDC->SelectObject(&Fuente);
                        pDC->SetBkColor(RGB(191,191,191));
                        pDC->TextOut(x+5, y+158, str);
                        // Nueva serigrafía
                        pDC->TextOut(32, 102, "Res  ");
                        pDC->TextOut(32, 127, "Direc");
                        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                           // Si hay tambien Deslizadores abajo, Hay que pintarlos
                                pDC->TextOut(x+5, y+333, str);
                                // Nueva serigrafía
                                pDC->TextOut(32, 277, "Res  ");
                                pDC->TextOut(32, 301, "Direc");
                        }
                }
                else    {
                        str.Format("%d",*(Filtro.m_Frecuencias+i-1));
                        pDC->SelectObject(&Fuente);
                        pDC->SetBkColor(RGB(191,191,191));
                        pDC->TextOut(x+5, y+158, str);
                        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                           // Si hay tambien Deslizadores abajo, Hay que pintarlos
                                pDC->TextOut(x+5, y+333, str);
                        }
                }

            // Colocacion de los niveles de Ganacia-Atenuacion en los Deslizadores
                if (Filtro.Dlg_f.m_Gcir == 10)
                        str.Format("+%d", Filtro.Dlg_f.m_Gcir);
                else
                        str.Format("+%ddB", Filtro.Dlg_f.m_Gcir);
                pDC->TextOut(x+34, y+12, str);
                if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                        // Si hay tambien Deslizadores abajo, Hay que pintarlos
                        pDC->TextOut(x+34, y+187, str);
                }
                if (Filtro.Dlg_f.m_Gcir == 10)
                        str.Format("-%d", Filtro.Dlg_f.m_Gcir);
                else
                        str.Format("-%ddB", Filtro.Dlg_f.m_Gcir);
                pDC->TextOut(x+34, y+140, str);
                if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                        // Si hay tambien Deslizadores abajo, Hay que pintarlos
```

```
        pDC->TextOut(x+34, y+315, str);
}
// Colocación del valor numérico que se esta aplicando a cada
// banda en el recuadro habilitado a tal efecto

if (i == 0)     {
        // Esta en los masters
        if (Filtro.m_MasterR == 1)
                str.Format(" 0dB");
        else    {
                if (Filtro.Dlg_f.m_Gcir == 10)
                        str.Format("%.1f",10*log10(Filtro.
                                    m_MasterR));
                else
                        str.Format("%.2f",10*log10(Filtro.
                                    m_MasterR));
        }
}
else    {
        // Resto de deslizadores
        if (*(Filtro.m_Cte+i-1) == 1)
                str.Format(" 0dB");
        else    {
                if (Filtro.Dlg_f.m_Gcir == 10)
                        str.Format("%.1f",10*log10(*(Filtro.
                                    m_Cte+i-1)));
                else
                        str.Format("%.2f",10*log10(*(Filtro.
                                    m_Cte+i-1)));
        }
}
pDC->TextOut(x+34, 45, str);
if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
        // Si hay Deslizadores abajo, Hay que pintarlos
        if (i == 0)     {
                // Esta en los masters
                if (Filtro.m_MasterL == 1)
                        str.Format(" 0dB");
                else    {
                        if (Filtro.Dlg_f.m_Gcir == 10)
                                str.Format("%.1f",10*log10(Filtro.
                                            m_MasterR));
                        else
                                str.Format("%.2f",10*log10(Filtro.
                                            m_MasterL));
                }
        }
        else    {
```

```
                        // Resto de deslizadores
                        if (*(Filtro.m_CteL+i-1) == 1)
                                str.Format(" 0dB");
                        else    {
                                if (Filtro.Dlg_f.m_Gcir == 10)
                                        str.Format("%.1f",10*log10(
                                            *(Filtro.m_CteL+i-1)));
                                else
                                        str.Format("%.2f",10*log10(
                                            *(Filtro.m_CteL+i-1)));
                        }
                }
                pDC->TextOut(x+34, 220, str);
        }
        // Deseleciona la fuente
        pDC->SelectStockObject(BLACK_BRUSH);


// Para que valga siempre sacar las coordenadas del array que conoce
// todas las posiciones de los Jogs, m_RefJog

        x+=58;

        // Una vez que se han pintado o repintado los Jog's se actualizan
        // los botones
        // Si no esta activado no pinta el botón pulsado, deja el fondo
        // Mutes
        if (*(Filtro.m_MutesR+i))       {
                m_dcMemMute->SelectObject(m_BmpMute);
                pDC->BitBlt(37+(i*58), 88, 13, 13, m_dcMemMute, 0, 0,
                            SRCCOPY);
        }
        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo &&
                        *(Filtro.m_MutesL+i))   {
                m_dcMemMute->SelectObject(m_BmpMute);
                pDC->BitBlt(37+(i*58), 263, 13, 13, m_dcMemMute, 0, 0,
                            SRCCOPY);
        }
        // Solos
        if (*(Filtro.m_SoloR+i))        {
                m_dcMemSolo->SelectObject(m_BmpSolo);
                pDC->BitBlt(37+(i*58), 113, 13, 13, m_dcMemSolo, 0, 0,
                            SRCCOPY);
        }
        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo &&
                        *(Filtro.m_SoloL+i))    {
                m_dcMemSolo->SelectObject(m_BmpSolo);
                pDC->BitBlt(37+(i*58), 288, 13, 13, m_dcMemSolo, 0, 0,
```

```
                                        SRCCOPY);
                        }
                }
        }
        else    {
                // No se accede a la GDI, prepara los datos inicialies para rastrear los
                // Jog's en la siguiente vez
                // Prepara los datos para la siguiente ejecucion

                m_Iden=2000;
                m_IdenL=2000;   // Para que nunca coincida con ningún control, es decir
                                // que la primera vez que los pinte sea en reposo
        }
        return;
}


void CFiltradoView::OnInitialUpdate()
{
        // Se calcula en funcion del tamaño de la vista...
        int x,y;                // Cálculos sobre las dimensiones de la ventana

        CScrollView::OnInitialUpdate();

        // Con GetClientRect puedo saber las dimensiones de la vista, pero se quieren
        // definir las dimensiones

        x=58*(Filtro.m_Cortes+1);
        y=175*(Filtro.Dlg_f.m_Filas ? 2 : 1);   // Se leen en el constructor para
                                                // dimensionar correctamente la ventana
        CSize sizeTotal(x, y);
        // Cálculos del tamaño de la vista
        CSize sizePage(sizeTotal.cx/2, sizeTotal.cy/2);
        CSize sizeLine(sizeTotal.cx/50, sizeTotal.cy/50);

        SetScrollSizes(MM_TEXT, sizeTotal, sizePage, sizeLine);

        // Creacion de los Contextos de dispositivo con los BMP
        Fuente.CreateFont(12, 0, 0, 0, 400, false, false, 0, ANSI_CHARSET,
                OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_SWISS, "Arial");
        m_BmpDeslizador->LoadBitmap(IDB_DESLIZADOR);
        m_dcMemDes->CreateCompatibleDC(NULL);
        m_BmpJog->LoadBitmap(IDB_JOG);
        m_dcMemJog->CreateCompatibleDC(NULL);

        m_BmpMute->LoadBitmap(IDB_MUTE_PULSADO);
        m_dcMemMute->CreateCompatibleDC(NULL);
```

```
        m_BmpSolo->LoadBitmap(IDB_SOLO_PULSADO);
        m_dcMemSolo->CreateCompatibleDC(NULL);
}

/////////////////////////////////////////////////////////////////////////////
// CFiltradoView diagnostics

#ifdef _DEBUG
void CFiltradoView::AssertValid() const
{
        CScrollView::AssertValid();
}

void CFiltradoView::Dump(CDumpContext& dc) const
{
        CScrollView::Dump(dc);
}

CFiltradoDoc* CFiltradoView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFiltradoDoc)));
        return (CFiltradoDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFiltradoView message handlers

void CFiltradoView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // Al pulsar con el raton en la vista hay que comprobar si se está encima de
        // uno de los Jogs (mandos) para desplazarlo y hacer lo que corresponde con él.
        // Moverlo en la vista y extraer el valor para el filtro.
        // Hay que rastrear todos los Jogs uno por uno, al igual que los botones.
        // El primer potenciometro, que es o son los master, tienen una gestion especial.

        CRect Jog(m_pointTopLeft, m_sizeJog);   // Rastreo Jog's
        CRect Boton(37,88,49,100);              // Rastreo de los botones
        CRgn Region;
        CPoint offset;                          // Almacena el Scroll


        if (g_AccesoGDI)        {
                CClientDC dc(this);
                OnPrepareDC(&dc);

                offset=GetDeviceScrollPosition();       // Offset por si hay Scroll
                Region.CreateRectRgnIndirect(Jog);      // inicializacion de la region con
```

```
                                        // cualquier valor
for (int i=0; i<Filtro.m_Cortes+1; i++) {
        Jog.top=(Filtro.m_RefJog+i)->y;
        Jog.left=(Filtro.m_RefJog+i)->x;
        Jog.bottom=((Filtro.m_RefJog+i)->y)+28;
        Jog.right=((Filtro.m_RefJog+i)->x)+18;

        Region.SetRectRgn(Jog);                 // Se van rastreando las
                                                // regiones de los Jogs
        if (Region.PtInRegion(point+offset))    {
                SetCapture();
                m_Captura=true;
        // Ya esta identificado quien, luego colocar sus coordenadas para
        // hacer los cálculos del arrastre
                m_pointTopLeft.x=((Filtro.m_RefJog+i)->x);
                m_pointTopLeft.y=(Filtro.m_RefJog+i)->y;
                CPoint pointTopLeft(m_pointTopLeft);
                dc.LPtoDP(&pointTopLeft);
                m_Offset=point-pointTopLeft;
                ::SetCursor(m_hCursor);     // Cambia el puntero del raton
        // Para los límites del Jog dentro del Deslizador
                m_FilaSup=true;
        // Almacenar la el identificador del deslizador seleccionado
                m_Iden=i;
                break;  // Solo puede activarse uno, solo hay un raton
        }
        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                // Si hay bandas separadas, una para cada canal
                Jog.top=(Filtro.m_RefJogL+i)->y;
                Jog.left=(Filtro.m_RefJogL+i)->x;
                Jog.bottom=((Filtro.m_RefJogL+i)->y)+28;
                Jog.right=((Filtro.m_RefJogL+i)->x)+18;

                Region.SetRectRgn(Jog);          // Se van rastreando las
                                                 // regiones de los Jogs
                if (Region.PtInRegion(point+offset))    {
                        SetCapture();
                        m_Captura=true;
        // Ya esta identificado quien luego colocar sus coordenadas para
        // hacer los cálculos del arrastre
                        m_pointTopLeft.x=(Filtro.m_RefJogL+i)->x;
                        m_pointTopLeft.y=(Filtro.m_RefJogL+i)->y;
                        CPoint pointTopLeft(m_pointTopLeft);
                        dc.LPtoDP(&pointTopLeft);
                        m_Offset=point-pointTopLeft;
                        ::SetCursor(m_hCursor);   // puntero del raton
                        m_FilaSup=false;
        // Almacenar la el identificador del deslizador seleccionado
```

```
                        m_IdenL=i;
                        break;  // Solo puede activarse uno, hay un raton
                }
        }

}
// Ahora rastrear los botones asociados con los potenciómetro, Mute y Solo
// Otro rastreo por todos ellos
// A diferencia de los Jog's eston no cambian su posición con lo cual, no
// es necesario un array de CPoint, conociendo las coordenadas relativas
// y rastreando desde (0,0), es suficiente
for (i=0; i<Filtro.m_Cortes+1; i++)     {
        // Botones de Mute
        Boton.top=88;
        Boton.left=37+(i*58);   // Barrer en eje x
        Boton.bottom=Boton.top+13;
        Boton.right=Boton.left+13;

        Region.SetRectRgn(Boton);           // Se rastrean los botones
        if (Region.PtInRegion(point+offset))    {
                if (i != 0)     {
                        // Cuando se pulsa Mute se silencia la banda
                        if (!(*(Filtro.m_MutesR+i)))    {
                    // Se ha pulsado; estado 1, pero valor 0.0, es mute...
                                // Cambia el estado del boton
                                (*(Filtro.m_MutesR+i))=true;
                                *(Filtro.m_MascaraR+i)=0.0;
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                    Ds.Muestreo.m_Modo)
                                        *(Filtro.m_MascaraL+i)=0.0;
                        }
                        else    {
                     // Se ha liberado; estado 0, pero se deja pasar 1.0
                                // Cambia el estado del boton
                                (*(Filtro.m_MutesR+i))=false;
                                // si lo deja Solo
                                if (Filtro.m_ISoloR == 0 || *(Filtro.
                                    m_SoloR+i))        {
                                        *(Filtro.m_MascaraR+i)=1.0;
                                        if (Filtro.Dlg_f.m_BandasStereo &&
                                            Ds.Muestreo.m_Modo)
                                               *(Filtro.m_MascaraL+i)=1.0;
                                }
                        }
                }
                else    {
                   // Se ha pulsado el mute del Master, se implementa
                   // un significado especial; Reset de los controles.
```

```
            // Será un pulsador, es decir no se quedará pulsado
            // resetea y sale
                if (!(*(Filtro.m_MutesR)))      {
                    // Cambia el estado del boton
                        (*(Filtro.m_MutesR))=true;
                }
        }
        Boton.left-=offset.x;           // Repintar el boton
        InvalidateRect(&Boton, false);
        break;          // Solo se pudede pulsar una cosa cada vez
}
// Botones de Solo
Boton.top=113;
Boton.left=37+(i*58);   // Barrer en eje x
Boton.bottom=Boton.top+13;
Boton.right=Boton.left+13;

Region.SetRectRgn(Boton);    // Se van rastreando las regiones de
                             // los Jogs
if (Region.PtInRegion(point+offset))    {
        // Se juega con la máscara pero hay que comprobar si hay
        // alguno ya pulsado (semaforo) y silenciar todos menos
        // ese si es el primero, o si ya hay más,  escuchar
        // solo los marcados
        if (i != 0)             {
                if (Filtro.m_ISoloR == 0)       {
                        // La primera pulsacion de un boton Solo
                        for(int j=1; j<i; j++)  {
                                *(Filtro.m_MascaraR+j)=0.0;
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                     Ds.Muestreo.m_Modo)
                                        *(Filtro.m_MascaraL+j)=0.0;
                        }
                        *(Filtro.m_MascaraR+j)=1.0;
                        if (Filtro.Dlg_f.m_BandasStereo &&
                            Ds.Muestreo.m_Modo)
                                        *(Filtro.m_MascaraL+j)=1.0;
                        for (j=i+1;j<Filtro.m_Cortes+1; j++)    {
                                *(Filtro.m_MascaraR+j)=0.0;
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                    Ds.Muestreo.m_Modo)
                                        *(Filtro.m_MascaraL+j)=0.0;
                        }
                        *(Filtro.m_SoloR+i)=true;
                        Filtro.m_ISoloR++;  // Cuantos pulsados
                }
                else    {
                        // Ya hay algún solo pulsado anteriormente
```

```
                        if (!(*(Filtro.m_SoloR+i)))     {
                           // Se añade otro canal, se da paso, a
                           // no ser que tenga el mute pulsado
                                if (!(*(Filtro.m_MutesR+i)))    {
                                       *(Filtro.m_MascaraR+i)=1.0;
                                       if (Filtro.Dlg_f.
                                           m_BandasStereo && Ds.
                                           Muestreo.m_Modo)
                                       *(Filtro.m_MascaraL+i)=1.0;
                                }
                                *(Filtro.m_SoloR+i)=true;
                                Filtro.m_ISoloR++;    // Contador
                        }
                        else    {
                                // Es la liberación de algun solo
                                Filtro.m_ISoloR--;
                                if (Filtro.m_ISoloR == 0)  {
                             // Es el último, dejar como estaba,
                             // teniendo  en cuenta solo los
                             //  mutes pulsados
                        for (int j=0; j<Filtro.m_Cortes+1; j++) {
                              if (!(*(Filtro.m_MutesR+j)))    {
                                 *(Filtro.m_MascaraR+j)=1.0;
                                  if (Filtro.Dlg_f.m_BandasStereo
                                              && Ds.Muestreo.m_Modo)
                                        *(Filtro.m_MascaraL+j)=1.0;
                                            }
                                      }
                                }
                                else    {
                                // Quedan botones Solo pulsados
                                 *(Filtro.m_MascaraR+i)=0.0;
                                 if (Filtro.Dlg_f.m_BandasStereo
                                      && Ds.Muestreo.m_Modo)
                                       *(Filtro.m_MascaraL+i)=0.0;
                                }
                                *(Filtro.m_SoloR+i)=false;
                        }
                }
        }
        else    {
            // Se ha pulsado el Solo del Master, también se le da
            // otro significado, en este caso para ignorar la
            // ecualización, un bypass y poder contrastar...
                if (*(Filtro.m_SoloR) == false)         {
                        *(Filtro.m_SoloR)=true;         // Estado
                        if (Filtro.Dlg_f.m_BandasStereo &&
                            Ds.Muestreo.m_Modo)
```

```
                                        *(Filtro.m_SoloL)=true;
                    // En la funcion de Filtrar se testea este componente
                    // del array y se opera en consecuencia.
                    // Todas las modificaciones hechas se almacenan en los
                    // arrays de siempre con lo que siempre hay coherencia
                    // entre lo mostrado por pantalla y el procesamiento
                        }
                        else    {
                                // Vuelve a procesar el ecualizaor
                                *(Filtro.m_SoloR)=false;        // Estado
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                    Ds.Muestreo.m_Modo)
                                        *(Filtro.m_SoloL)=false;
                        }
                }
                Boton.left-=offset.x;           // Repintar el boton
                InvalidateRect(&Boton, false);
                break;     // Solo se pudede pulsar una cosa cada vez
        }

        // Para los de abajo...
        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                // Si hay bandas separadas, una para cada canal
                // Botones de Mute
                Boton.top=263;
                Boton.left=37+(i*58);           // Barrer en eje x
                Boton.bottom=Boton.top+13;
                Boton.right=Boton.left+13;

                Region.SetRectRgn(Boton);     // Se van rastreando los Jogs
                if (Region.PtInRegion(point+offset))    {
                        if ( i != 0)    {
                                if (!(*(Filtro.m_MutesL+i)))    {
             // Se ha pulsado; estado 1, pero cte 0.0, es mute...
                                        // Cambia el estado del boton
                                        (*(Filtro.m_MutesL+i))=true;
                                        *(Filtro.m_MascaraL+i)=0.0;
                                }
                                else    {
             // Se ha liberado; estado 0, pero cte 1.0
                                        // Cambia el estado del boton
                                        (*(Filtro.m_MutesL+i))=false;
                                        *(Filtro.m_MascaraL+i)=1.0;
                                }
                        }
                        else    {
                                // Como es el Master se reiniciliza todo
                                if (!(*(Filtro.m_MutesL)))      {
```

```
                                        // Cambia el estado del boton
                                        (*(Filtro.m_MutesL))=true;
                                }
                        }
                        // Repintar correctamente el boton
                        Boton.left-=offset.x;
                        InvalidateRect(&Boton, false);
                        break;   // Solo se pudede pulsar una cosa
                }
                // Rastreo de Solo
                Boton.top=288;
                Boton.left=37+(i*58);   // Barrer en eje x
                Boton.bottom=Boton.top+13;
                Boton.right=Boton.left+13;

                Region.SetRectRgn(Boton);    // Se rastrean los botones
                if (Region.PtInRegion(point+offset))    {
        // Se juega con la mascara pero hay que comprobar si alguno ya
        // pulsado (semaforo) y silenciar todos menos ese si es el
        // primero o si ya hay escuchar solo los marcados
                        if ( i != 0)    {
                              if (Filtro.m_ISoloL == 0)       {
                                 // La primera pulsacion de un boton Solo
                                      for(int j=1; j<i; j++)  {
                                           *(Filtro.m_MascaraL+j)=0.0;
                                      }
                                      *(Filtro.m_MascaraL+j)=1.0;
                                      for (j=i+1;j<Filtro.m_Cortes+1;
                                           j++)    {
                                             *(Filtro.m_MascaraL+j)=0.0;
                                      }
                                      *(Filtro.m_SoloL+i)=true;
                                      // cuantos hay pulsados
                                      Filtro.m_ISoloL++;
                              }
                              else    {
                             // Ya hay algún solo pulsado anteriormente
                                 if (!(*(Filtro.m_SoloL+i)))     {
                             // Se añade otro canal para escucharlo solo,
                             // se da paso, a no ser que tenga el mute
                             // pulsado
                                       if (!(*(Filtro.m_MutesL+i)))    {
                                            (Filtro.m_MascaraL+i)=1.0;
                                           }
                                           *(Filtro.m_SoloL+i)=true;
                                           Filtro.m_ISoloL++; // Contador
                                      }
                                      else    {
```

```
                                                        // Es la liberación de algun boton solo
                                                           Filtro.m_ISoloL--;
                                                           if (Filtro.m_ISoloL == 0)    {
                                                        // Es el último, asi que dejar como
                                                        // estaba, teniendo en cuenta solo
                                                        // los mutes pulsados
                                                          for (int j=0; j<Filtro.m_Cortes+1;
                                                               j++) {
                                                             if (!(*(Filtro.m_MutesL+j))) {
                                                               *(Filtro.m_MascaraL+j)=1.0;
                                                                  }
                                                               }
                                                            }
                                                            else    {
                                                            // Quedan botones Solo pulsados
                                                               *(Filtro.m_MascaraL+i)=0.0;
                                                            }
                                                            *(Filtro.m_SoloL+i)=false;
                                                        }
                                                }
                                        }
                                        else    {
                                        // Se pulso Solo de abajo del Master, igual
                                        // que en los potenciómetros de arriba
                                                if (*(Filtro.m_SoloL) == false) {
                                                        *(Filtro.m_SoloL)=true;
                                                }
                                                else
                                                        *(Filtro.m_SoloL)=false;
                                        }
                                        Boton.left-=offset.x;      // Repintar el boton
                                        InvalidateRect(&Boton, false);
                                        break;        // Solo se pudede pulsar una cosa
                                }
                        }
                }
        }
        return;
}


void CFiltradoView::OnLButtonUp(UINT nFlags, CPoint point)
{
        // La cota en la ganancia o atenuacion se recoge del dialogo de filtrado,
        // la cota es simetrica respecto a 0dB

        double cota, temp;
        CPoint offset;        // Almacena el scroll actual
```

```
CRect rc;             // Para repintar los tag que indican el valor del
                      // potenciómetro

// Finalizar la captura
if (m_Captura)  {
        offset=GetDeviceScrollPosition();
        ::ReleaseCapture();
        cota=((Filtro.Dlg_f.m_Gcir)/10.0);
        // Gestion del acceso a los arrays m_Cte's de CFiltro
        Filtro.m_Csfiltro.Lock();
        if (m_FilaSup)  {
                // Deslizadores de arriba, canal Derecho
                (Filtro.m_RefJog+m_Iden)->y=m_pointTopLeft.y;
                 // Salva la nueva posicion para poder escanearlo otra vez
                // 122-10 =112 pixels de desplazamiento, el medio en 66
                if (m_pointTopLeft.y < 66)              {
                        // Ganacia mayor que 1, amplifica...
                TRACE("Para Amplificar con valor %d\n",m_pointTopLeft.y);
         // Recta que mapea de los pixels [10,66] al intervalo [0,1] enteros
                        temp=((1.18-(m_pointTopLeft.y * .018)) * cota);
                        if (m_Iden == 0)        {
                                // Gestion del Master
                                Filtro.m_MasterR=pow(10, temp);
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                    Ds.Muestreo.m_Modo)
                                        Filtro.m_MasterL=Filtro.m_MasterR;
                        }
                        else    {
                                *(Filtro.m_Cte+m_Iden-1)= pow(10, temp);
                                if (Filtro.Dlg_f.m_BandasStereo &&
                                    Ds.Muestreo.m_Modo)
                                        *(Filtro.m_CteL+m_Iden-1)=*(Filtro.m_Cte+
                                          m_Iden-1);
                                    // Si se captura en Stereo y hay deslizadores
                                    // para 1 canal
                        }
        TRACE("m_Cte=%f\n",*(Filtro.m_Cte+m_Iden-1));
                }
                else if (m_pointTopLeft.y > 66) {
                        // Ganancia menor que 1, atenuar...
        TRACE("Para Atenuar con valor %d\n",m_pointTopLeft.y);
          // Esta recta mapea los pixels [66,122] al intervalo [0,-1] enteros
                        temp=((1.18-(.018 * m_pointTopLeft.y)) * cota);
                        if (m_Iden == 0)        {
                                // Gestion del Master
                                Filtro.m_MasterR=pow(10, temp);
                                if (Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.
                                    m_Modo)
```

```
                                Filtro.m_MasterL=Filtro.m_MasterR;
                }
                else    {
                        *(Filtro.m_Cte+m_Iden-1)= pow(10, temp);
                        if (Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.
                            m_Modo)
                // Si se captura en Stereo y hay deslizadores para 1 canal
                                *(Filtro.m_CteL+m_Iden-1)=*(Filtro.m_Cte+
                                   m_Iden-1) ;
                }
        TRACE("m_Cte=%f\n",*(Filtro.m_Cte+m_Iden-1));
        }
        else    {
        // Solo queda la igualdad
TRACE("Queda igual\n");
                if (m_Iden == 0)        {
                        // Gestion del Volumen principal
                        Filtro.m_MasterR=1.0;
                        if (Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.
                            m_Modo)
                                Filtro.m_MasterL=1.0;
                }
                else    {
                        *(Filtro.m_Cte+m_Iden-1)= 1.0;
                        if (Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.
                            m_Modo)
                                *(Filtro.m_CteL+m_Iden-1)=1.0;

        TRACE("m_Cte=%f\n",*(Filtro.m_Cte+m_Iden-1));
                }
        }
        // Para repintar los indicadores numericos con la ganancia
        rc.top=45;
        rc.left=32+(m_Iden*58)-offset.x;
        rc.bottom=rc.top+11;
        rc.right=rc.left+22;
        InvalidateRect(&rc, false);
}
else    {
        // Deslizadores de abajo, Canal Izquierdo
        (Filtro.m_RefJogL+m_IdenL)->y=m_pointTopLeft.y;
        // Salva la nueva posicion para poder escanearlo otra vez
        // 122-10 =112 pixels de desplazamiento, el medio en 66+175=241

        if (m_pointTopLeft.y < 241)             {
                // Ganacia mayor que 1, amplifica...
         TRACE("Para Amplificar con valor %d\n",m_pointTopLeft.y);
   // Recta que mapea de los pixels [10,66] al intervalo [0,1] enteros
```

```
                        temp=((1.18-((m_pointTopLeft.y-175) * .018)) * cota);
                        if (m_IdenL == 0)
                                // Gestion de los Masters
                                Filtro.m_MasterL=pow(10, temp);
                        else
                                *(Filtro.m_CteL+m_IdenL-1)= pow(10, temp);
                TRACE("m_CteL=%f\n",*(Filtro.m_CteL+m_IdenL-1));
                }
                else if (m_pointTopLeft.y > 241)        {
                        // Ganancia menor que 1, atenuar...
                TRACE("Para Atenuar con valor %d\n",m_pointTopLeft.y);
            // Esta recta mapea los pixels [66,122] al intervalo [0,-1] enteros
                        temp=((1.18-(.018 * (m_pointTopLeft.y-175))) * cota);
                        if (m_IdenL ==0)
                                // Gestion del Master
                                Filtro.m_MasterL=pow(10, temp);
                        else
                                *(Filtro.m_CteL+m_IdenL-1)= pow(10, temp);
                TRACE("m_CteL=%f\n",*(Filtro.m_CteL+m_IdenL-1));
                }
                else    {
                // Solo queda la igualdad
                TRACE("Queda igual, L\n");
                        *(Filtro.m_Cte+m_IdenL-1)= 1;
                TRACE("m_CteL=%f\n",*(Filtro.m_Cte+m_IdenL-1));
                }
                // Para repintar los indicadores numericos con la ganancia
                rc.top=220;
                rc.left=32+(m_IdenL*58)-offset.x;
                rc.bottom=rc.top+11;;
                rc.right=rc.left+22;
                InvalidateRect(&rc, false);
        }
        // Termina el acceso a los Arrays para CFiltro

        Filtro.m_Csfiltro.Unlock();
        m_Captura=false;
        ::SetCursor(::LoadCursor(NULL, IDC_ARROW));
}
// Aqui se puede hacer la lectura del valor final y por lo tanto hacer las
// transformaciones necesarias para obtener la cte que multiplique a esa
// banda de frecuencias

// Para levantar los botones de Reset, se comportan como pulsadores
if ((*(Filtro.m_MutesR)) == true)               {
        (*(Filtro.m_MutesR))=false;     // Cambia el estado del boton
        // Reinicialización de todos los arrays
        Filtro.Reiniciar(true);
```

```
                Invalidate(true);               // Refresca toda la ventana incluido
                                                // fondo y todo
                m_Iden=2000;
                m_IdenL=2000;   // Igual que en el constructor
        }
        // Para los de abajo...
        if (!Filtro.Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                if ((*(Filtro.m_MutesL)) == true)               {
                        (*(Filtro.m_MutesL))=false;   // Cambia el estado del boton
                        // Reinicialización de todos los arrays
                        Filtro.Reiniciar(false);
                        Invalidate(true);             // Refresca toda la ventana incluido
                                                      // fondo y todo
                        m_Iden=2000;
                        m_IdenL=2000;   // Igual que en el constructor
                }
        }
        return;
}


void CFiltradoView::OnMouseMove(UINT nFlags, CPoint point)
{
        LONG topeSup, topeInf;

        // Accion própiamente del arrastre
        if (m_Captura)  {
                CClientDC dc(this);
                OnPrepareDC(&dc);
                CRect OldPosition(m_pointTopLeft, m_sizeJog);
                dc.LPtoDP(OldPosition);
                InvalidateRect(OldPosition, false);

                // Solo me interesa el movimiento sobre el eje Y
                m_pointTopLeft.y=point.y-m_Offset.cy;

                // Para que no se salga de su sitio
                if (m_FilaSup)          {
                        // esta en la fila superior
                        topeSup=10;
                        topeInf=122;
                }
                else    {
                        // Esta en la Fila Inferior
                        topeSup=185;
                        topeInf=297;
                }
                // Ahora comprueba los limites, sabeindo en que fila está
```

```
                if (m_pointTopLeft.y <= topeSup)        {
                        //Por encima
                        m_pointTopLeft.y=topeSup;
                }
                else if (m_pointTopLeft.y >= topeInf)   {
                        // Por debajo
                        m_pointTopLeft.y=topeInf;
                }

                CRect NewPosition(m_pointTopLeft, m_sizeJog);
                dc.LPtoDP(NewPosition);
                InvalidateRect(NewPosition, false);
        }

        return;
}
```

3.3.6 Filtro

Filtro.h

```
// Filtro.h: interface for the CFiltro class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FILTRO_H__A9516515_2DD7_11D5_8915_C3BC2EEFCE78__INCLUDED_)
#define AFX_FILTRO_H__A9516515_2DD7_11D5_8915_C3BC2EEFCE78__INCLUDED_

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

#include "Dialog_Filtrado.h"

class CFiltro
{
public:
        CFiltro();
        virtual ~CFiltro();

        Configurar();
        FiltrarR(fftw_real *in, fftw_real *out);
        FiltrarL(fftw_real *in, fftw_real *out);
        void Reiniciar(bool der);
        Liberar();

public:
        int *m_Agrupacion;                      // N de cortes por banda
        int m_CortesMax;                        // Gestion del cálculo de las bandas
```

```
        bool *m_MutesR, *m_MutesL;              // Saben que botones están pulsados
        bool *m_SoloR, *m_SoloL;
        double *m_MascaraR, *m_MascaraL;        // Para emular Mute y Solo
        int m_ISoloR, m_ISoloL;

public:
        int m_Cortes;
        double *m_Cte, *m_CteL, m_MasterR, m_MasterL;   // Coeficientes de filtrado
        POINT *m_RefJog, *m_RefJogL;                    // Sabe donde estan los JOG's
        int  m_Fs, *m_Frecuencias;
        CDialog_Filtrado Dlg_f;
        CCriticalSection m_Csfiltro;

};

#endif // !defined(AFX_FILTRO_H__A9516515_2DD7_11D5_8915_C3BC2EEFCE78__INCLUDED_)
```

Filtro.cpp

```
// Filtro.cpp: implementation of the CFiltro class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Filtrado.h"
#include "Global.h"

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

bool g_AccesoGDI;

extern CDirectS Ds;
extern CFft Fft;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFiltro::CFiltro()
{
        // Inicializaciones adicionales, pero no es bueno que los constructores
        // tengan muchas si se pueden colocar en otro sitio porque aqui pueden
        // dar problemas difícilmente depurables
```

```
}

CFiltro::~CFiltro()
{
}


int CFiltro::Configurar()
{
        // Del archivo "FILTRADO.CFG" se obtienen los 2 arrays calculados en
        // Equalizador.exe, con las frecuencias y la agruapcion sobre los puntos
        // de la FFT.

        // Inicializacion de las coordenadas que servirán de referencia para
        // los Jog, como las distancias son relativas se conocen de antemano
        // y se puede inicializar aqui una vez
        // El m_Cortes+1 es para el control de los master para controlar el volumen

        LONG    x,y;
        int i;
        x=0;
        y=0;

        m_Cte=(double *) malloc(m_Cortes * sizeof(double));
        m_RefJog=(POINT *) malloc((m_Cortes+1) * sizeof(POINT));
        if (!Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo)
                m_RefJogL=(POINT *) malloc((m_Cortes+1) * sizeof(POINT));
        if (Ds.Muestreo.m_Modo)
                m_CteL=(double *) malloc(m_Cortes * sizeof(double));

        // Incremento en contador en 1 para albergar un control más para poder regular
        // el volumen general; Master Volume
        for (i=0; i<m_Cortes+1; i++)    {
                (m_RefJog+i)->x=x+10;
                (m_RefJog+i)->y=y+66; // Array que almacena las posisciones
                x+=58;
                // Inicializacion de los potenciometros, posicion inicial
        }
        m_MasterR=1.0;
        if (!Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo)        {
                x=0;
                y=175;
                for (i=0; i<m_Cortes+1; i++)    {
                        (m_RefJogL+i)->x=x+10;
                        (m_RefJogL+i)->y=y+66;  // Estructuras que almacenan las posisciones
                        x+=58;
                        // Inicializacion de los potenciometros, posicion inicial
```

```
                }
        }
        m_MasterL=1.0;

        // Inicializacion de los arrays con valores iniciales de reposo, es decir,
        // ganancia 1, si no se toca los deslizadores
        for (i=0; i< m_Cortes; i++)     {
                *(m_Cte+i)=1.0;
                if (Ds.Muestreo.m_Modo) {
                        *(m_CteL+i)=1.0;
                }
        }

        // Inicializacion del array de los Mutes, Solos y su gestion
        m_MutesR=new bool [(m_Cortes+1)*sizeof(bool)];
        m_SoloR=new bool [(m_Cortes+1)*sizeof(bool)];
        m_MascaraR=new double[(m_Cortes+1)*sizeof(double)];
        if (Dlg_f.m_BandasStereo || Ds.Muestreo.m_Modo)         {
                m_MutesL=new bool [(m_Cortes+1)*sizeof(bool)];
                m_SoloL=new bool [(m_Cortes+1)*sizeof(bool)];
                m_MascaraL=new double[(m_Cortes+1)*sizeof(double)];
        }
        for (i=0; i<m_Cortes+1; i++)    {
                *(m_MutesR+i)=false;
                *(m_SoloR+i)=false;
                *(m_MascaraR+i)=1.0;
                if (Dlg_f.m_BandasStereo || Ds.Muestreo.m_Modo) {
                        *(m_MutesL+i)=false;
                        *(m_SoloL+i)=false;
                        *(m_MascaraL+i)=1.0;
                }
        }
        m_ISoloR=0;             // No se ha pulsado ningún solo aún
        m_ISoloL=0;

        return 0;
}


void CFiltro::Reiniciar(bool der)
{
        // Se usa para solo reiniciar a los valores inicialies de todos los parámetros
        // para el Reset que lanzan los botones de los Masters

        int i;
        LONG    x,y;
        x=0;
        y=0;
```

```
if (der)        {
        m_Csfiltro.Lock();
        // Inicializacion de los potenciometros, posicion inicial
        for (i=0; i<m_Cortes+1; i++)    {
                (m_RefJog+i)->x=x+10;
                (m_RefJog+i)->y=y+66;   // Arrays que almacenan las posisciones
                x+=58;
        }
        m_MasterR=1.0;
        if (Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo)
                m_MasterL=1.0;

        // Inicialización de los arrays con los multiplicadores de las bandas
        for (i=0; i< m_Cortes; i++)     {
                *(m_Cte+i)=1.0;
                if (Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                        *(m_CteL+i)=1.0;
                }
        }
        for (i=0; i<m_Cortes+1; i++)    {
                *(m_MutesR+i)=false;
                *(m_SoloR+i)=false;
                *(m_MascaraR+i)=1.0;
                if (Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo) {
                        *(m_MutesL+i)=false;
                        *(m_SoloL+i)=false;
                        *(m_MascaraL+i)=1.0;
                }
        }
        m_ISoloR=0;             // No se ha pulsado ningún solo aún
        if (Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo)
                m_ISoloL=0;

        m_Csfiltro.Unlock();
}
else    {
        // Solo resetear el canal izquierdo
        m_Csfiltro.Lock();
        x=0;
        y=175;
        for (i=0; i<m_Cortes+1; i++)    {
                (m_RefJogL+i)->x=x+10;
                (m_RefJogL+i)->y=y+66;  // Arrays que almacenan las posisciones
                x+=58;
                // Inicializacion de los potenciometros, posicion inicial
        }
        m_MasterL=1.0;
```

```
                for (i=0; i< m_Cortes; i++)
                        *(m_CteL+i)=1.0;
                for (i=0; i<m_Cortes+1; i++)    {
                        *(m_MutesL+i)=false;
                        *(m_SoloL+i)=false;
                        *(m_MascaraL+i)=1.0;
                }
                m_ISoloL=0;
                m_Csfiltro.Unlock();
        }

        return;
}


CFiltro::FiltrarR(fftw_real *in, fftw_real *out)
{
        // Aqui se procesan los datos en los arrays para producir el filtrado de la
        // señal, se pasa por aqui justo después de obtener la FFT en frecuencia y
        // antes de volver a pasarla al dominio temporal

        int i,j;

        i=1;            // Recorre todos los componentes, evita la DC, empieza en 1
        j=0;            // Recorre las bandas

        // Copia del nivel de continua
        *out=*in;
        // Procesamiento del resto de componentes del Canal Derecho
        // Evitar conflictos eon el acceso a los arrays m_Cte's
        if (*m_SoloR == false)          {
                // Procesamiento del ecualizador
                m_Csfiltro.Lock();
                while (j < m_Cortes)    {
                        while (i <= *(m_Agrupacion+j))  {
                                *(out+i)=*(in+i) * (*(m_Cte+j)) * (*(m_MascaraR+j+1)) *
                                          m_MasterR;
                                *(out+(Fft.m_n-i))=*(in+(Fft.m_n-i)) * (*(m_Cte+j)) *
                                          (*(m_MascaraR+j+1)) * m_MasterR;
                                i++;
                        }
                        j++;
                }
                m_Csfiltro.Unlock();
        }
        else    {
                // Se omiten los valores del ecualizador y se procesa plano
```

```
                m_Csfiltro.Lock();
                while (j < m_Cortes)    {
                        while (i <= *(m_Agrupacion+j))  {
                                *(out+i)=*(in+i) * m_MasterR;
                                *(out+(Fft.m_n-i))=*(in+(Fft.m_n-i)) * m_MasterR;
                                i++;
                        }
                        j++;
                }
                m_Csfiltro.Unlock();
        }

        return 0;
}

CFiltro::FiltrarL(fftw_real *in, fftw_real *out)
{
        // Aqui se procesan los datos en los arrays para producir el filtrado de la
        // señal, se pasa por aqui justo después de obtener la FFT en frecuencia y
        // antes de volver a pasarla al dominio temporal

        int i,j;

        i=1;            // Recorre todos los componentes, evita la DC, empieza en 1
        j=0;            // Recorre las bandas

        // Copia del nivel de continua
        *out=*in;
        // Procesamiento del resto de componentes del Canal Izquierdo
        // Objeto CCriticalSection para evitar errores de acceso a datos
        if (*m_SoloL == false)          {
                m_Csfiltro.Lock();
                while (j < m_Cortes)    {
                        while (i <= *(m_Agrupacion+j)+1)        {
                                *(out+i)=*(in+i) * (*(m_CteL+j)) * (*(m_MascaraL+j+1)) *
                                         m_MasterL;
                                *(out+(Fft.m_n-i))=*(in+(Fft.m_n-i)) * (*(m_CteL+j)) *
                                         (*(m_MascaraL+j+1))
                                        * m_MasterL;
                                i++;
                        }
                        j++;
                }
                m_Csfiltro.Unlock();
        }
        else    {
                m_Csfiltro.Lock();
                while (j < m_Cortes)    {
```

```
                        while (i <= *(m_Agrupacion+j)+1)        {
                                *(out+i)=*(in+i) * m_MasterL;
                                *(out+(Fft.m_n-i))=*(in+(Fft.m_n-i)) * m_MasterL;
                                i++;
                        }
                        j++;
                }
                m_Csfiltro.Unlock();
        }

        return 0;
}

CFiltro::Liberar()
{
        // Liberacion de toda la memoria dinamicamente asignada, dejando asi
        //preparado el objeto para una nueva configuracion

        free(m_Agrupacion);
        free(m_Cte);
        free(m_Frecuencias);
        free(m_RefJog);
        delete m_MutesR;
        delete m_SoloR;
        delete m_MascaraR;

        if (!Dlg_f.m_BandasStereo && Ds.Muestreo.m_Modo)
                free(m_RefJogL);
        if (Ds.Muestreo.m_Modo) {
                free(m_CteL);
                delete m_MutesL;
                delete m_SoloL;
                delete m_MascaraL;
        }

        return 0;
}
```

3.3.7 MainFrm

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__104E5540_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
#define AFX_MAINFRM_H__104E5540_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_
```

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

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)
        // Mis funciones...
        void Serialize(CFile *File, bool Salvar);
        LeerArchivo();
// 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

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnParar();
        afx_msg void OnComenzarFiltrado();
        afx_msg void OnClose();
        afx_msg void OnUpdateComenzarFiltrado(CCmdUI* pCmdUI);
        afx_msg void OnUpdateParar(CCmdUI* pCmdUI);
        afx_msg void OnViewStatusBar();
        afx_msg void OnUpdateViewStatusBar(CCmdUI* pCmdUI);
        afx_msg void OnSalir();
```

```
        //}}AFX_MSG
        afx_msg void OnError();
        DECLARE_MESSAGE_MAP()
};

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

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

#endif // !defined(AFX_MAINFRM_H__104E5540_4ED4_11D5_871D_9E7B2B87C879__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Filtrado.h"

#include "MainFrm.h"
#include "Global.h"

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

// Objetos Globales
CDirectS Ds;
CFft    Fft;
extern CFiltro Filtro;


// Variables Globales
bool g_Filtrado;
bool g_Grabar;
bool g_Analisis;
int g_CreandoPlan;

extern bool g_AccesoGDI;

CString str1;
/////////////////////////////////////////////////////////////////////////////
// CMainFrame
```

```
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_COMMAND(ID_PARAR, OnParar)
        ON_COMMAND(ID_COMENZAR_FILTRADO, OnComenzarFiltrado)
        ON_WM_CLOSE()
        ON_UPDATE_COMMAND_UI(ID_COMENZAR_FILTRADO, OnUpdateComenzarFiltrado)
        ON_UPDATE_COMMAND_UI(ID_PARAR, OnUpdateParar)
        ON_COMMAND(ID_VIEW_STATUS_BAR, OnViewStatusBar)
        ON_UPDATE_COMMAND_UI(ID_VIEW_STATUS_BAR, OnUpdateViewStatusBar)
        ON_COMMAND(ID_SALIR, OnSalir)
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_ERROR, OnError)
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,
        ID_SEPARATOR,           // Linea de Estado personalizada
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // Inicializaciones adicionales
        g_Analisis=false;
        g_Grabar=false;
        g_Filtrado=false;
        g_CreandoPlan=1;
        g_AccesoGDI=false;

        LeerArchivo();          // Lo que ha escrito Ecualizador.exe
}

CMainFrame::~CMainFrame()
{
}

CMainFrame::LeerArchivo()
{
        // Lee del disco el archivo con la configuracion de filtrado

        CString str("FILTRADO.CFG");
```

```
        CFile File;

        if (!File.Open(str, CFile::modeRead))   {
                TRACE("No se ha podido leer ni crear OPCIONES.CFG\n");
                g_Error=90;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return -1;
        }
        else    {
                // Lectura del archivo
                Filtro.Dlg_f.Serialize(&File, false);
                File.Close();
        }
        // Copias de datos importantes a los objetos correspondientes
        Filtro.m_Cortes=Filtro.Dlg_f.m_Cortes;

        return 0;
}


int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

        if (!m_wndStatusBar.Create(this, WS_CHILD | WS_VISIBLE| CBRS_BOTTOM,
                ID_MI_BARRA_ESTADO)||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // Configura el 2ž panel de la barra de estado como reescalable
        m_wndStatusBar.SetPaneInfo(0, 0, SBPS_STRETCH | SBPS_NOBORDERS, 0);
        m_wndStatusBar.SetPaneInfo(1, 0, 0, 140);

        // TODO: Remove this if you don't want tool tips or a resizeable toolbar
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
```

```
        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        // Modificando la estructura cs se modifica el estilo de la ventana, lo que se
        // ve y los botones que nos mostrará Güindous

        // Tambien se controla el tamaño del marco de ventana, asi que hay que tener
        // en cuenta tambien los bordes y barras de menús, herramientas, estado y
        // botones. Cálculos que no hay que hacer en OnInitialUpdate para la vista

        int temp;

        cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER | WS_MINIMIZEBOX ;

        // El tamaño se ajusta a los deslizadores, la pantalla y lo centra
        // Ventana con tamaño estandar hasta que se sepa cuantos cortes hay
        cs.cy = Filtro.Dlg_f.m_Filas ? 175*2 : 175;
        cs.cy+=(83+25+2);                               // Ajustes con los bordes
        temp=::GetSystemMetrics(SM_CXSCREEN)/58;        // Maximo de cortes que entran
        if ((Filtro.m_Cortes+1) <= temp)        {
                cs.cx = 58*(Filtro.m_Cortes+1);
        }
        else    {
                // Hay Scroll horizontal, luego aparecen las barras de desplazamiento
                cs.cx=58*temp;
                cs.cy+=19;                              // Por la barra de Scroll
        }
        cs.cx+=(5+5+2);                                // Ajustes con los bordes
        cs.y = (::GetSystemMetrics(SM_CYSCREEN)-cs.cy)/2;
        cs.x = (::GetSystemMetrics(SM_CXSCREEN)-cs.cx)/2;

        return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
```

```
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::Serialize(CFile *File, bool Salvar)
{
        // Para Guardar o Cargar del disco el contenido de algunas variables miembro

        int i;

        // save=true Salva los datos al disco y save=false carga datos del disco
        if (Salvar)     {
                // Se escribe en el disco...
                CArchive Archivo(File, CArchive::store, 512);
                CObject::Serialize(Archivo);
                if (Archivo.IsStoring())        {
                        // Aqui solo se lee...
                }
        }
        else    {
                // Se lee del disco...
                CArchive Archivo(File, CArchive::load, 512);
                CObject::Serialize(Archivo);            // Llamada a la clase base
                if (!Archivo.IsStoring())       {
                        Archivo >> Ds.Muestreo.m_Fs >> Ds.Muestreo.m_Bits
                                >> Ds.Muestreo.m_Modo;
                        // Hay que rescatar la estreucturas elemento por elemento
                        Archivo >> Ds.guidDs.Data1 >> Ds.guidDs.Data2 >> Ds.guidDs.Data3;
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDs.Data4[i];
                        Archivo >> Ds.guidDsC.Data1 >> Ds.guidDsC.Data2
                                >> Ds.guidDsC.Data3;
                        for (i=0; i<8; i++)
                                Archivo >> Ds.guidDsC.Data4[i];
                }
        }
        return;
}
```

```
void CMainFrame::OnComenzarFiltrado()
{
        // Lanza todos los pasos necesarios para realizar el filtrado, se supone
        // como en Analisis que existe el fichero "OPCIONES.CFG", donde están los
        // datos necesarios para inicializar la tarjeta de sonido, puesto que
        // las opciones propiamente dichas de Filtrado se gestionan desde aquí para
        // no obligar al usuario a ir a la Lanzadera y cambiarlo, a no ser un cambio
        // en la configuracion de los parámetros de captura

        static bool primera=false;
        CFile File;
        CString str("OPCIONES.CFG");
        RECT rc;
        int result;

        if (!g_Filtrado)        {
                // Lectura del archivo
                if (!primera)   {
                        primera=true;
                        if (!File.Open(str, CFile::modeRead))   {
                                TRACE("No se ha podido leer OPCIONES.CFG\n");
                                g_Error=90;
                                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR,
                                              0, 0);
                                return;
                                }
                        else    {
                                // Lectura del archivo
                                Serialize(&File, false);
                                File.Close();
                        }
                        result=Ds.CrearObjetos();
                        if (result != 0)        {
                                g_Error=30;
                                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR,
                                              0, 0);
                                return;
                        }
                }

                g_TerminarHilo.ResetEvent();
                g_Filtrado=true;

                // Comprobacion de la instalacion de los DirectX y cualquier otro error
                if (g_Error != 0)       {
                        ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                        return;
```

```
        }
        // Llamada a la Inicializacion del Plan para FFT
        Fft.CrearPlan();

        // Inicializacion del Objeto DirectSound
        result=Ds.Configurar();
        if (result != 0)        {
                g_Error=31;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return;
        }
// Segun el parametro que se le pase a InicializarBuffers, el Objeto Ds, creara
// los eventos en el buffer de Captura (Analiss, Filtrado y Grabacion) o en el
// secundario (Lectura de archivo WAV)

        result=Ds.InicializarBuffers(true);
        if (result != 0)        {
                g_Error=32;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return;
        }
        str="FILTRADO.CFG";
        if (!File.Open(str, CFile::modeRead))   {
                TRACE("No se ha podido leer FILTRADO.CFG\n");
                g_Error=90;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return;
                }
        else    {
                // Lectura del archivo
                Filtro.Dlg_f.Serialize(&File, true);
                File.Close();
        }
        result=Filtro.Configurar();     // Se configura...
        if (result != 0)        {
                g_Error=61;
                ::PostMessage(AfxGetMainWnd()->GetSafeHwnd(), WM_ERROR, 0, 0);
                return;

        // Pintar la primera vez todos los controles
        GetClientRect(&rc);
        InvalidateRect(&rc, false);

        // Por último lanzar el audio para que comienze la captura y reproduccion,
        // cuando sea necesario
        AfxBeginThread(Hilo_LanzarAudio, Ds.m_Hwnd,
                        THREAD_PRIORITY_TIME_CRITICAL);
```

```
                g_AccesoGDI=true;       // Para acceder a la vista
        }
        return;

}


void CMainFrame::OnUpdateComenzarFiltrado(CCmdUI* pCmdUI)
{
        // Gestion del boton de Lanzamiento del filtrado

        pCmdUI->Enable(!g_Filtrado);
        return;
}


void CMainFrame::OnParar()
{
        // Activa el objeto evento para que en el bucle que recoge los eventos
        // de gestion del buffer compruebe si tiene que terminar.
        // No es posible enviarle un mensaje puesto que no tiene ventana es un
        // hilo trabajador

        g_TerminarHilo.SetEvent();              // Manda parar los dos hilos
        g_AccesoGDI=false;

        Sleep(500);             // Esperar a que termine todo y ya refrescar

        // Repinte la GDI toda la pantalla
        DrawMenuBar();
        RedrawWindow(NULL, NULL, RDW_FRAME | RDW_INVALIDATE | RDW_ERASE);
        UpdateWindow();         // Fuerza un repintado de la vista

        // Colocar las variables de estado en reposo
        g_Filtrado=false;               // Estado de la aplicación
        g_Error=0;                              // Resetea la bandera de los errores

        return;
}


void CMainFrame::OnUpdateParar(CCmdUI* pCmdUI)
{
        //      Sincroniza el menu para Parar con el estado de la aplicacion
        pCmdUI->Enable((g_Filtrado) && !g_CreandoPlan);

        // Aprovecho que la MFC ejecuta esto para actualizar el estado de los
        // botones en momentos ociosos de güindous para actualizar la Barra de
```

```
// Estado personalizada

CString str;
CMainFrame* pFrame=(CMainFrame *) AfxGetApp()->m_pMainWnd;
CStatusBar* pStatus=&pFrame->m_wndStatusBar;

if (!g_Analisis && !g_Filtrado && g_CreandoPlan)        {
        // En reposo sin plan creado todavia
        if (pStatus)    {
        str.Format("Preparado para comenzar...");
        pStatus->SetPaneText(0, str);
        }
}
else if (!g_Analisis && !g_Filtrado && !g_CreandoPlan) {
        // En reposo pero con plan creado
        if (pStatus)    {
                str.Format("Preparado...");
                pStatus->SetPaneText(0, str);
        }
}
else if (g_Analisis && !g_Filtrado && !g_CreandoPlan) {
        if (pStatus)    {
                str.Format("Analisis gráfico");
                pStatus->SetPaneText(0, str);
        }
}
else if (!g_Analisis && !g_Filtrado && !g_CreandoPlan) {
        if (pStatus)    {
                str.Format("Caracterizacion de la tarjeta de sonido");
                pStatus->SetPaneText(0, str);
        }
}
else if (!g_Analisis && g_Filtrado && !g_CreandoPlan) {
        // En modo filtrado
        if (pStatus)    {
                str.Format("Filtrando...");
                pStatus->SetPaneText(0, str);
        }
}
else if (g_CreandoPlan == 2)    {
        if (pStatus)    {
                str.Format("Creando plan para FFT, este cálculo puede llevar
                            unos instantes");
                pStatus->SetPaneText(0, str);
        }

}
else    {
```

```
                // Estado indeterminado
                if (pStatus)    {
                        str.Format("Esto no debería salir, así que algo va mal !!!!");
                        pStatus->SetPaneText(0, str);
                }

        }
        // Por último pone la info de captura en la barra de estado
        CString temp;
        if (pStatus)    {
                if (g_Filtrado) {
                        str=Ds.Muestreo.m_Fs;
                        temp.Format(", %dBits",Ds.Muestreo.m_Bits==0 ? 8 : 16);
                        str=str+temp;
                if (Ds.Muestreo.m_Modo == 0)
                                str=str+" Mono";
                        else
                                str=str+" Stereo";
                }
                else    {
                        str.Empty();
                }
                pStatus->SetPaneText(1, str);

        }

        return;
}


void CMainFrame::OnError()
{
        // Gestion de los mensajes de error
        // Si el flujo del programa a llegado hasta aquí algo ha fallado...
        // La variable global g_Error contendrá el entero que identificará que
        // error es el que ha sucedido;

        CDialog_Error Dlg;              // Crea un objeto para representar el dialogo

        switch (g_Error)        {
        case 15:
        case 14:
        case 13:
        case 12:
        case 11:
//              Dd.Restaurar();
        case 10:
                ShowCursor(true);
```

```
//              Dd.Destruir(true);              // No hay parte de DirectDraw
        case 41:
        case 40:
        case 39:
        case 38:
        case 37:
        case 36:
                Ds.Parar();
        case 35:
        case 34:
        case 33:
        case 32:
        case 31:
        case 30:
                Ds.Liberar(true);
                break;
        default:
                break;
        }

        Dlg.DoModal();                  // Lanza el dialogo a la pantalla

        while (g_CreandoPlan == 2)      {
                // Espera a que acabe de crear plan para FFT
                g_Analisis=true;
                g_Filtrado=true;
        }
        g_Analisis=false;
        g_Filtrado=false;

        PostMessage(WM_CLOSE, 0, 0);    // Cierra la aplicación

        return;
}


void CMainFrame::OnClose()
{
        // Gestion de la aplicacion antes de terminar
        // En caso de cerrar la aplicacion se comprueba que se deja todo para
        // evitar fugas de memoria o punteros asignados

        if (g_Filtrado) {
                // Esta filtrando al venir aqui=> parar todo y liberarlo todo
                OnParar();              // Se encauza primero hacia la parada
        }
        // Cuando se sale normalmente hay que liberar los interfaces a los objetos
        // Tanto DirectSound como DirectDraw
```

```
        Ds.Liberar(true);

        if (!g_CreandoPlan)             {
                Fft.DestruirPlan();
        }
        // Liberacion de los arrays que se generan según las opciones del ususario

        TRACE("PASO POR ONCLOSE\n");
        CFrameWnd::OnClose();
}


void CMainFrame::OnSalir()
{
        // Para salir de la aplicacion desde el menu, simplemente ir a la función
        // OnClose que ya gestiona la salida correctamente...

        OnClose();

}

void CMainFrame::OnViewStatusBar()
{
        // Muestra en pantalla la barra de estado personalizada

        m_wndStatusBar.ShowWindow((m_wndStatusBar.GetStyle() & WS_VISIBLE) ==0);
        RecalcLayout();
}

void CMainFrame::OnUpdateViewStatusBar(CCmdUI* pCmdUI)
{
        // Sincroniza el estado del menu con la barra de estado

        pCmdUI->SetCheck((m_wndStatusBar.GetStyle() & WS_VISIBLE) != 0);
        return;

}

/////////////////////////////////////////////////////////////////////////////
//      Función global para ejecutar hilos, por requerimientos de Güindous y de la
//      función empleada para lanzarlo
//
//
UINT Hilo_LanzarAudio(LPVOID pParam)
{
        // Funcion externa que envuelve a la funcion que realmente filtra
        // por requerimeintos de los hilos
```

```
        int result;

        result=Ds.LanzarCaptura();
        if (result != 0)
                PostMessage((HWND) pParam, WM_ERROR, 0, 0);
        // Si hay error tambien hay que parralo todo, asi que dejo que siga
        // la parada normal...
        // g_Error se asigna donde se produzca el error

        // Deja el Ojeto Ds preparado para lo siguiente
        Ds.Parar();
        Ds.Liberar(false);

        g_TerminarHilo.ResetEvent();            // Desseñala para la siguiente vez
        // Preparacion del objeto Filtro, para siguiente lanzamiento
        Filtro.Liberar();

        return 0;
}
```

Capítulo 4

Presupuesto

Herramientas Software

Microsoft Visual C++ 5.0	.....	20.000
Paint Shop Pro 5.0	.....	17.000
Office 97	.....	45.000
Microsof Install Shield v5.0 free edition	.....	0
Axialis Ax-Icons	.....	2.200
SDK Microsoft DirectX v6.0	.....	0
Licencia GNU de fftw	.....	472.500

Hardware

Tarjeta de sonido (Sb128 PCI o equivalente)	.....	4.500
Micrófono Patrón	.....	70.000
Otros micrófonos	.....	18.000
2 Cables de RCA→mini-jack	..... 2×1.500	3.000
1 Adaptador Jack→mini-jack	.....	700

Mano de obra

$9meses \times 22dias \times 8horas = 1584h$

1h Ingeniero 3.800 ptas $\Rightarrow 1584h \times 3.800$ ptas/h 6.019.200

Gastos adicionales por venta

1 Cd grabado con los ejecutables	.....	75
1 Caja de cartón	.....	120

Amortización

Mano de obra	.....	6.019.200
Software	.....	556.700
Hardware	.....	78.200
Producción	.....	195 /Ud
Total Gastos	.....	6.654.100

Una posibilidad para los cálculos de amortización es establecier el número de unidades del producto con las que se desea recuperar lo invertido y apartir de él entrar en ganancias, por lo tanto se establece N como cantidad fija y se haya el precio necesario para esa amortización, con lo que solo queda añadirle el I.V.A. y se obtiene el precio del producto.

Se ha optado por la elección de un precio para salir al mercado de 34.000 pts ó 204 Euros, luego se realiza un cálculo para averiguar a partir de qué número de ventas se amoriza todo lo invertido y se entra en ganancias.

$$6.654.100 + 195 \times N \geq 34.000 \times N$$

$$N \geq 197 \text{ Unidades}$$

Por lo tanto a partir de la venta 197 se ha recuperado la inversión inicial, y todas las ventas que se realicen a partir de ésta son ganancias netas. Así queda el producto con el precio final en:

Precio del software

Programa Ecualizador v1.0	.....	34.000
16 % I.V.A.	.....	5440
Total software Ecualizador v1.0	.....	39.440

Precio del equipo completo

Portátil Pentium III 933 MHz y 128 MBytes RAM	.....	258.900
Tarjeta de sonido USB o PCMCIA	.....	15.000
Micrófono patrón	.....	70.000
Cables auxiliares	.....	3.700
Software Ecualizador v1.0	.....	34.000
Subtotal	.....	414.900
Impuesto I.V.A. 16%	.....	66.384
Total	.....	481.284

Índice de Figuras

2.1 Esquema típico de conexión 6
2.2 Esquema típico de conexión 7
2.3 Diálogo de opciones gráficas 11
2.4 Diálogo de opciones de filtrado 13
2.5 Pantalla de grabación 15
2.6 Diálogo de opciones de captura 16
2.7 Diálogo de Selección de dispositivo 16
2.8 Información de reproducción 17
2.9 Información de captura 18

Bibliografía

[1] Ayuda On-Line que está en el CD de instalación de Microsoft Visual C++ 5.0

[2] Ayuda On-Line de la SDK DirectX 6.0

[3] Docuementación de www.fftw.org, sección de Documentación(Doc)

[4] A fondo DirectX, Bradley Bargen y Peter Donnely, 1998.

[5] Programación avanzada con Visual C++. David j. Kruglinsky, Scot Wingo y George Shepherd, 1998.

[6] Construcción de software Orientado a Objetos. Bertrand Meyer, 1998.

[7] Programación en C/C++, Chris H. Pappas y William H. Murray, 1996.

[8] Programación en C, Herbert Schildt.