AUROVA-VIS-MAN-SOF-2003-16

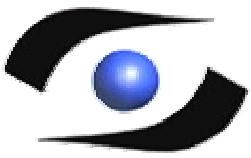
Visual 3

Interfaz para la creación de módulos externos (V.1)

Francisco Andrés Candelas Herías

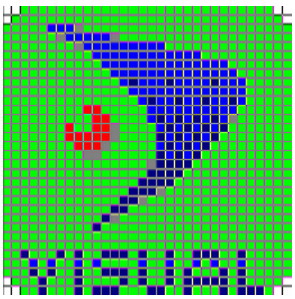
6 de junio de 2003

Universitat d'Alacant
Universidad de Alicante



Grupo de Automática, Robótica y Visión Artificial

Visual 3

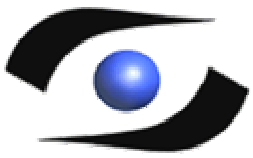


Interfaz para la creación de módulos externos (V.1)

Francisco A. Candelas Herías

27 de mayo de 2003

Universitat d'Alacant
Universidad de Alicante



Grupo de Automática, Robótica y Visión Artificial

Contenido

1. Introducción

Las operaciones de un esquema están representadas por los *OPIs* (Objetos de procesamiento de Imágenes usados en el entorno Visual). Un OPI mismo puede realizar una o más funciones dependiendo de su tipo. Así por ejemplo, existe un tipo de OPI de operaciones aritméticas que realiza las funciones de suma, resta producto y división de imágenes. Para cada tipo de OPI se define un único módulo de código como un archivo ejecutable externo a la aplicación.

Dentro de un esquema del entorno visual, un OPI tiene unas propiedades propias de su tipo de OPI, que se configuran con un diálogo. Para especificar esas propiedades y su aspecto en el diálogo de configuración se debe definir un archivo de interfaz para cada módulo de código.

Finalmente, a cada función de los distintos tipos de OPIs se les puede asociar un dibujo o icono que es mostrado en los OPIs de un esquema. Estos iconos se definen como imágenes BMP externas a la aplicación.

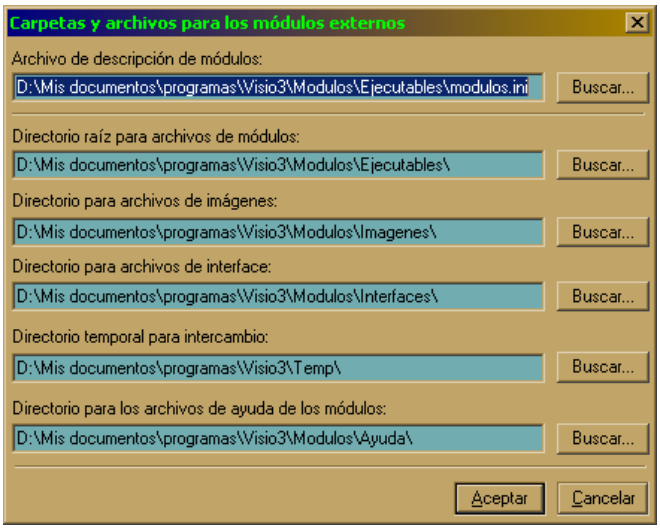
2. Directorios

Los distintos ficheros utilizados para la gestión de los módulos externos de los OPIS se almacenan clasificados en las siguientes carpetas:

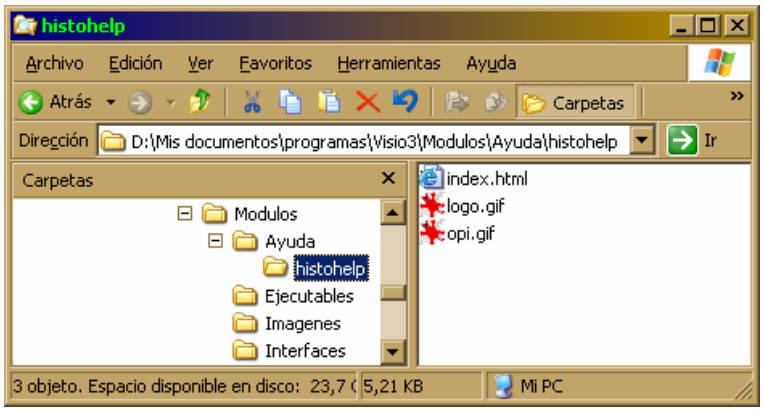
- **Directorio de módulos**: Incluye todos los ficheros ejecutables (EXE o DLL) de los módulos.
- **Directorio de ficheros de imágenes**. Contiene los ficheros de imágenes que representan cada una de las funciones que desarrollan cada uno los módulos. Las imágenes son del tipo DIB-Bitmap (.bmp), de 32x32 pixels y 16 colores sin compresión.
- **Directorio de ficheros de interfaz**. Contiene los ficheros de descripción de interfaz de usuario para cana módulo.
- **Directorio temporal de intercambio**. Se utiliza para depositar las imágenes que representan los datos de entrada y salida de los módulos. La imágenes de entrada a un módulo se crean antes de ejecutar ese modulo y se destruyen cuando acaba su ejecución. Así, este directorio debe estar vacío antes de ejecutar y después de cerrar el entorno Visual.
- **Directorio de archivos de ayuda**. Este directorio contiene a su vez las carpetas con los archivos de ayuda HTML para los módulos que disponen de ayuda. Estos archivos se visualizan cuando se selecciona la ayuda de un OPI dentro del entorno Visual.

Cada uno de los directorios puede estar en cualquier parte del sistema de archivos, y pueden especificarse en el entorno Visual por el usuario, en el diálogo que se muestra con la opción del menú *Archivo* → *Configuración de Directorios*. Si al ejecutarse el entorno visual no tiene adecuadamente configurados los directorios, también se mostrará dicho diálogo y será necesario configurarlos correctamente para arrancar la aplicación.

En el diálogo de configuración de las carpetas, que se muestra en la siguiente figura, también se define el *archivo de descripción de módulos*, cuya función se describe en el siguiente punto.



La configuración del diálogo anterior se corresponde con esta estructura de directorios:



3. Archivo de descripción de módulos

La lista de módulos que la aplicación debe cargar, la jerarquía de clases de estos, y sus datos principales se definen en un *archivo de descripción de módulos*, en formato texto plano, cuyo nombre y ruta pueden ser escogidos por el usuario del entorno Visual en menú *Archivo* → *Configuración de Directorios*. Su formato se describe a continuación.

El archivo se compone de dos bloques de definiciones, el que describe la lista de módulos de los OPIs y sus funciones, y el que define la jerarquía o árbol de clases que se mostrará en el entorno visual para permitir seleccionar las operaciones a utilizar en el diseño de esquemas. Los bloques se deben limitar con estas etiquetas:

```
[MODULOS]
  # Definición de módulos y funciones
[END]
[CLASES]
  # Definición de clases
[END]
```

Es indiferente el orden de definición de los dos bloques. Además, su texto puede escribirse en mayúsculas o minúsculas.

Para que el aspecto del fichero de configuración se más legible, pueden usarse líneas de comentario, las cuales comienzan por el carácter "#" y líneas en blanco, así como espacios o tabuladores a la izquierda de las diferentes entradas.

El entorno Visual comprueba que el fichero de configuración es correcto mientras lo interpreta, y en caso de detectar errores informa sobre el tipo de error y la línea en donde se encuentra.

3.1. Definición de módulos

Dentro del bloque de módulos se debe describir las características básicas de cada módulo seguidas de las características de sus funciones, del siguiente modo:

```
Nombre_del_OPI Ident_OPI Num_funciones Ayuda
Nomb_función_1 Id_f_1 Ent_Min_1 Ent_Max_1 Sal_Min_1 Sal_Max_1 Fich_dibujo_1
Descripción_de_la_función_1
Nomb_función_2 Id_f_2 Ent_Min_2 Ent_Max_2 Sal_Min_2 Sal_Max_2 Fich_dibujo_2
Descripción_de_la_función_2
    ...
```

A continuación se describe cada parámetro:

- **Nombre_del_OPI**: Cadena de texto con el nombre del OPI.
- **Ident_OPI**: Valor numérico con el identificador inequívoco del OPI.
- **Num_funciones**: Número de funciones que incluye el módulo, y que se definen a continuación.
- **Nombre_fichero_exe**: Cadena de texto con el nombre del archivo ejecutable del módulo. No contiene ruta, ya que se supone que esta en el directorio de módulos. Tampoco debe contener la extensión, que se supone ".exe" o ".dll" y la añade el Entorno Visual.
- **Nomb_función_x**: Cadena de texto con el nombre de la función.
- **Id_f_x**: Identificador de la función, único dentro del módulo. Es un valor entero 0, 1,…
- **Ent_Min_x**, **Ent_Max_x**: Números de entradas mínimo y máximo que puede tener el OPI. Si son iguales, indican que se requiere un número concreto de entadas.
- **Sal_Min_x**, **Sal_Max_x**: Números de salidas mínimo y máximo que puede tener el OPI. Si son iguales, indican que se genera un número concreto de salidas.
- **Fich_dibujo_1**: Nombre del fichero que incluye el dibujo asociado a la función. No incluye la ruta ya que se supone que está en el directorio de imágenes, ni la extensión, que se supone ".bmp".
- **Ayuda**: Nombre de la carpeta que contiene los archivos de la ayuda del OPI y que está situada dentro del directorio de archivos de ayuda (ver apartado 2). Este dato es optativo, y no se indica para los OPIs que no disponen de ayuda.

Las líneas de descripción contienen el texto que mostrará el Entorno Visual como breve descripción de lo que hace una determinada función.

Las cadenas de caracteres con nombres no pueden contener espacios (excepto para las líneas de descripción de funciones), aunque si el carácter "_" que será reemplazado por un espacio. Los valores numéricos deben ser enteros.

Los valores de identificador de OPI para los distintos módulos definidos deben ser distintos. También deben ser distintos los valores de identificador de función dentro del grupo de funciones de un módulo. Es importante no cambiar estos valores en un fichero de configuración que ya ha sido utilizado por el Entorno Visual para dibujar esquemas, puesto que el esquema podría malinterpretarse. Además, los identificadores de OPIs y funciones deben ser valores

comprendidos desde 0 (incluido este) hasta 30000 (no incluido), ya que los valores de 30000 o mayores son usados por el Entorno Visual para sus OPIs internos

La definición de OPIs básicos utiliza identificadores 1,2… tal y como se describe en el Anexo 7.1. Otros módulos adicionales pueden usar numeración más alta, como 100,101…

3.2. Definición de clases

Dentro del bloque de clases se define un árbol de jerarquía de tipos de OPIs. Este árbol se especifica mediante sentencias como estas:

```
Clase_Padre_1
    Clase_Hija_11
    Clase_Hija_12
    Clase_Hijo_13

Clase_Padre_2
    Clase_Hija_21
    Clase_Hija_22
    Clase_Hijo_23
...
```

Una *Clase_Padre* es una nueva clase de OPIs que engloba varios de estos. Una *Clase_Hija* es un nombre de OPI o una *Clase_Padre* definida anteriormente. De este modo, el árbol se define desde las ramas hacia la raíz. La raíz del árbol se indica con el carácter "*" como valor de nueva *Clase_Padre*.

3.3. Ejemplo

Un ejemplo de archivo de configuración puede ser este:

```
# MODULOS DE OPERACIONES PARA LOS OPIS

[MODULOS]

Constante 1 1 constan.exe
  generar 0 0 0 1 1 constan.ico
  Genera una imagen de las características indicadas

Aritméticas 2 4 basicas1.exe AyudaAritm
  Suma 0 2 10 1 1 suma.ico
  Suma varias imágenes
  Resta 1 2 10 1 1 resta.ico
  Resta a una imagen otras imágenes
  Producto 2 2 10 1 1 prod.ico
  Multiplica varias imágenes
  División 3 2 2 1 1 div.ico
  Divide dos imágenes

Lógicas 3 4 basicas2.exe AyudaLog
  And 0 1 10 1 1 and.ico
  Devuelve el producto lógico de varias imágenes
  Or 1 1 10 1 1 or.ico
  Devuelve la suma lógica de varias imágenes
  Xor 2 1 10 1 1 xor.ico
  Devuelve la suma lógica en módulo 2 de varias imágenes
  Not 3 1 1 1 1 not.ico
  Devuelve la imagen complementaria de una imágen

Morfología 4 4 morfo.exe AyudaMorf
```

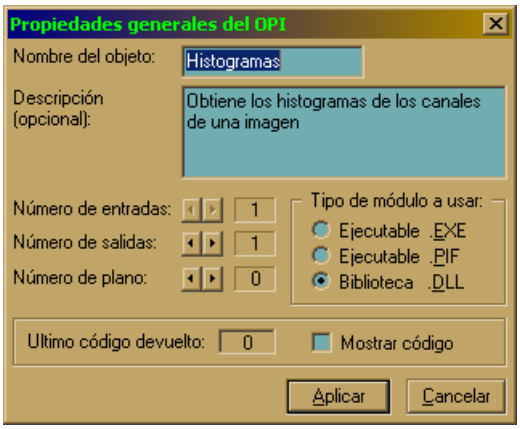
```
    Erosión 0 2 2 1 1 ero.ico
    Eorsiona una imagen con el elemento estructurante dado como otra imagen
    Dilatación 1 2 2 1 1 dila.ico
    Dilata una imagen con el elemento estructurante dado como otra imagen
    Apertura 2 2 2 1 1 aper.ico
    Apertura morfológica
    Cierre 3 2 2 1 1 cierre.ico
    Cierre morfológico

  RGB 5 2 rgb.exe
    Separar_RGB 0 1 1 3 3 seprgb.ico
    Devuelve imágenes de canales RGB separados
    Juntar_RGB 1 3 3 1 1 junrgb.ico
    Combina las imágenes de canales RGB en una imágen

  HSI 6 2 hsi.exe
    Separar_HSI 0 1 1 3 3 sephsi.ico
    Devuelve imágenes de canales HSI separados
    Juntar_HSI 1 3 3 1 1 junhsi.ico
    Combina las imágenes de canales HSI en una imágen
  [FIN]

  # ARBOL DE CLASES DE OPERACIONES PARA LOS OPIS

  [CLASES]

  Básicas 3
    Constante
    Aritméticas
    Lógicas
  Color 2
    RGB
    HSI
  * 3
    Básicas
    Morfología
    Color

  [FIN]
```

4. Archivos de módulos

Un módulo es un fichero ejecutable EXE o una biblioteca DLL (Dinamic Link Library) que representa las operaciones de un OPI.

En el caso del módulo EXE, este debe ser compilado como una aplicación de de consola Win 32. Un módulo tipo DLL debe ser compilado para ser compatible con los objetos MFC, esto es una DLL de extensión MFC. Los módulos tipo DLL, aunque pueden resultar un poco más complicados de programar en un principio, ofrecen varias ventajas: se cargan más rápidamente en memoria, sólo se cargan una vez aunque sean usados por muchas operaciones de los esquemas en ejecución, no muestran una ventana de consola que entorpece la visualización…

El formato concreto, EXE o DLL, se escoge con las propiedades genéricas del OPI dentro del entorno visual, en un diálogo como el siguiente:



Cada módulo puede realizar a su vez varías funciones del mismo tipo. Todos los archivos de módulos se encuentran todos en el directorio de módulos.

Cuando se ejecuta un módulo, éste recibe como entradas unos datos necesarios para realizar sus operaciones, así como el nombre completo (con la ruta) de los ficheros con las imágenes de entrada a operar, y el nombre de las imágenes en donde dejar los resultados. Como salida puede devolver un código de éxito o error.

Los módulos operan imágenes, aunque estás pueden ser desde un simple número o vectores hasta imágenes de varios planos o canales.

4.1. Paso de parámetros para un módulo EXE

Un módulo EXE recibe los parámetros de entrada como argumentos de línea de comando (cadenas de texto tras el nombre la aplicación) en el siguiente orden;

- **Nombre y camino del módulo**. Por compatibilidad con el interprete de comandos MS-DOS, el primer parámetro coincide con el camino y el nombre completo del módulo. Se puede ignorar este parámetro.
- **Función a realizar**. Número entero desde 0 que indica la función a realizar.
- **Número de entradas**: Número de imágenes de entrada que deben ser operadas.
- **Número de salidas**: Número de imágenes de salida que debe generar el módulo.
- **Directorio temporal**: Directorio temporal donde están las imágenes de entrada y salida.
- **Datos de configuración de entrada**. Lista de parámetros con los valores de los datos de configuración de entrada que requiere el módulo. Cada módulo conoce de antemano el número de estos parámetros y el tipo de cada uno, de acuerdo con lo especificado en el fichero de interfaz correspondiente (apartado 5). El orden en que se reciben los datos es el mismo que el definido en el archivo de interfaz.
- **Nombres de las imágenes de entrada**. Nombres sin camino de las imágenes de entrada que debe operar el módulo, dados como tantos parámetros como número de imágenes de entrada.
- **Nombres de las imágenes de salida**. Nombres sin camino de las imágenes de salida que debe generar el módulo, dados como tantos parámetros como número de imágenes de salida.

Para que las cadenas de texto de los parámetros se interpreten correctamente, en ellas se han sustituido los caracteres en blanco (espacio) por el carácter "*". El módulo debe realizar la conversión inversa para las distintas cadenas antes de utilizarlas.

En un programa en C, la forma de obtener el número de parámetros y las cadenas en donde se hallan estos se basa en los parámetros que recibe la función *main*:

```
int main(int argc, char* argv[])
```

Para los parámetros que representan valores numéricos se requiere una conversión a partir de la cadena recibida. Se puede usar una función como *sscanf*, *strtod*, *strtol*, etc.

Los parámetros recibidos no necesitan en principio comprobación o validación, aunque se recomienda verificar que los números de entradas, salidas y controles son correctos para evitar excepciones al acceder a posiciones no válidas de los vectores con los nombres de imágenes o valores de los parámetros.

4.2. Paso de parámetros para un módulo DLL

El uso de un módulo como biblioteca DLL difiere del uso de un módulo ejecutable Win32 en el paso de parámetros.

En este caso, un OPI utiliza un archivo tipo ".dll" compatible con los objetos MFC. En él se requiere definir una función exportable llamada Ejecutar que es usada por el entorno visual para ejecutar una función del OPI. La cabecera de esta función es la siguiente:

```
int Ejecutar(int iFunc, int iNumCtr, int iNumEnt, int iNumSal, CString&
strTemp, CString* pStrDatos, CString* pStrImgEntr, CString* pStrImgSal);
```

Para que esta función sea exportable, debe declararse dentro del apartado *EXPORTS* del archivo ".def" del proyecto de la DLL.

Para los nombres de imágenes se utilizan objetos MFC tipo *CString*. El significado de los parámetros es este:

- **IFunc**: Funcion a realizar (0, 1, 2...)
- **InumCtr**: Numero de controles o datos de configuración
- **INumEnt**: Numero de entradas
- **INumSal**: Numero de salidas
- **StrTemp**: Directorio donde están las imágenes de entrada o salida
- **PstrDatos**: Array con los datos de configuración, todos como cadenas de texto
- **PStrImgEntr**: Array con los nombres de las imágenes de entrada
- **PStrImgSal**: Array con los nombres de las imágenes de salida

La función debe devolver un código conforme lo explicado en el punto 4.4.

Los parámetros recibidos no necesitan en principio comprobación o validación, aunque se recomienda verificar que los números de entradas, salidas y controles son correctos para evitar excepciones al acceder a posiciones no válidas de los vectores con los nombres de imágenes o valores de los parámetros.

4.3. Imágenes de entrada y salida

Las imágenes de entrada a operar y las de resultado de salida se almacenan en el directorio temporal de intercambio. Son imágenes sin comprimir que admiten varios planos o canales de dos dimensiones. También permiten especificar el tamaño del valor de un pixel.

El formato de un archivo de imagen debe obedecer a esta estructura (un WORD son dos BYTEs):

```
WORD wXSize
WORD wYSize
WORD wChanels
WORD wPixelSize
BYTE data[wXSize* wYSize* wChanels* wPixelSize]
```

Los valores representan respectivamente el tamaño X, el tamaño Y, el número de canales, los bytes que ocupa un píxel de un canal, y los datos sin comprimir de la imagen. Los datos de la imagen se ordenan alternando los valores de los valores de los distintos canales para cada píxel para cada fila de la imagen, empezando desde la fila superior. Por ejemplo, para una imagen RGB, con esta estructura:

	Columna 1	Columna 2	…	Columna Y
Fila 1	(R11, G11, B11)	(R12, G12, B12)		(R1X, G1X, B1X)
Fila 2	R21, G21, B21)	(R22, G22, B22)		(R2X, G2X, B2X)
…				
Fila Y	(RY1, GY1, BY1)	(RY2, GY2, BY2)		(RYX, GYX, BYX)

Se sigue esta ordenación en los datos:

```
R11, G11, B11, R12, G12, B12, …, R1X, G1X, B1X,
R21, G21, B21, R22, G22, B22, …, R2X, G2X, B2X, …
RY1, GY1, BY1, RY2, GY2, BY2, …, RYX, GYX, BYX
```

Como se ha indicado, el valor de un pixel puede ocupar uno o más bytes, siendo lo habitual trabajar con pixels de tamaño 1 byte con valores en el rango [0,255]. Pero también es posible trabajar con imágenes con otros formatos de píxel, como por ejemplo los siguientes tipos de datos de C/C++ para plataformas MS. Windows:

Tipo	Tamaño	Descripción
WORD	2 bytes = 16 bits	Entero sin signo de 16 bits
DWORD	4 bytes = 32 bits	Entero sin signo de 32 bits
int	4 bytes = 32 bits	Entero con signo de 32 bits
float	4 bytes = 32 bits	Coma flotante IEEE con bit de signo, 8 bits de exponente y 23 de mantisa
double	8 bytes = 64 bits	Coma flotante IEEE con bit de signo, 11 bits de exponente y 52 de mantisa

Valores como los de coma flotante pueden resultar muy costosos en cuanto a cantidad de memoria para imágenes, pero son muy útiles para devolver y mostrar resultados numéricos o estadísticos (ver documento “Operaciones de Visualización de Resultados”).

Los valores de más de un byte se deben almacenar de forma directa, sin conversión a un tipo básico previamente. Por ejemplo, en MFC-C++, se puede definir, iniciar y guardar una imagen de valores tipo *double* en un archivo binario de este modo.

```
Struct cabecera{
  WORD wXSize
  WORD wYSize
  WORD wChanels
```

```
  WORD wPixelSize
};

Struct cabecera cab;
cab.wXSize = 10;
cab.wYSize = 10;
cab.wChanels = 1;
cab.PixelSize = (WORD)sizeof(double); // 8

DWORD dwSize = (DWORD)cab.wXSize;
dwSize *= (DWORD)cab.wYSize;
dwSize *= (DWORD)cab.wChanels;
dwSize *= (DWORD)cab.PixelSize;

BYTE* pData = BYTE[dwSize];
BYTE* pIndData;

double numero = 0.0;

for(WORD y=0; y< wYSize; y++)
  for(WORD x=0; x< wXSize; x++)
  {
      (double)(*pIndData) = numero;
      pIndData += cab.PixelSize;
  }

CFile file;
if (file.Open(strImgSalida, CFile::modeCreate | CFile::modeWrite))
{
   TRY
    {
        file.Write(&cab, sizeof(cab));
        file.Write(pData, dwSize);
    }
   CATCH( CFileException, e )
   {
       return FALSE;
    }
    END_CATCH

    file.Close();
}
return TRUE;
```

Los datos también se pueden iniciar y guardar de este modo:

```
double* pData = double[cab.wXSize][cab.wYSize];

for(WORD y=0; y< wYSize; y++)
  for(WORD x=0; x< wXSize; x++)
     pData[x][y] = 0.0;

BYTE* pDataBy = (BYTE*)pData;

...
file.Write(pDataBy, dwSize);
...
```

Al dibujar esquemas hay que tener en cuenta los formatos de imagen que devuelven o reciben los OPIs para que se puedan ejecutar correctamente. Cada módulo debe verificar que el tipo de imagen que recibe es uno de los que puede procesar, comprobando la información de su cabecera.

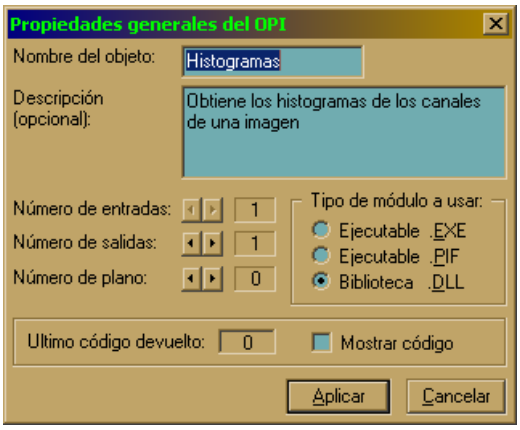
4.4. Códigos de error

Un módulo puede devolver códigos de error a través del valor de retorno de su función principal. El valor 0 se entenderá como que no hubo errores y las imágenes resultado se generaron sin problemas. Los valores positivos entre 0 y 100 indican errores en la ejecución. Los valores por encima de 100 están reservados para códigos internos. En un programa en C, el valor de retorno se puede generar así:

```
int main(int argc, char* argv[])
{
      if (Operación1()==FALSE)
          return 1;
      if (Operación2()==FALSE)
          return 2;
      return 0;
}
```

Para un módulo tipo DLL, el código de error es el valor entero devuelto por la función *Ejecutar*.

Los valores de error devueltos son mostrados por en entorno Visual en la ventana de mensajes. También se puede ver el último valor de error devuelto por un OPI en el diálogo de propiedades generales de un OPI. En ese diálogo también se puede activar la opción *Mostrar código* para forzar a que, en caso de que el OPI devuelva un código erróneo, se muestre una ventana de aviso con ese valor.



5. Archivos de interfaz

Básicamente, estos archivos definen el aspecto del diálogo que el Entorno Visual debe presentar al usuario cuando este solicita cambiar los datos de configuración de entrada de un módulo, como por ejemplo el nivel de ruido a añadir a una imagen, el umbral de binarización, los datos de tamaño y tipo de una nueva imagen, etc.

Cada módulo tiene su propio fichero de interfaz, común para todas sus funciones. En este fichero se define el número y tipo de datos de configuración requeridos, así como el tipo de control con que el usuario introduce los datos en la aplicación.

Un fichero de interfaz correspondiente a un módulo tiene su mismo nombre, pero con la extensión ".ini" y se encuentra en el directorio de ficheros de interfaz definido. Su formato es:

```
Nombre_par_1 Tipo x y ancho alto Valor_min Valor_max Valor_defecto
Nombre_par_2 Tipo x y ancho alto Valor_min Valor_max Valor_defecto
...
```

Debe existir una línea para cada control en donde se especifican su nombre, su tipo (ver tabla siguiente), la posición del punto superior izquierdo de su rectángulo dentro del diálogo y sus dimensiones, sus valores mínimo y máximo y valor por defecto. El fichero admite además líneas de comentario tras el símbolo "#", así como identación y líneas vacías.

Aunque siempre se interpretan los valores de posición *x*, *y* y *ancho*, el valor de alto depende del tipo de control. Por ejemplo, los controles de edición numérica y de archivo mantienen un alto fijo. Un control de *selección* usa el valor de alto para definir la separación ente opción. Solo los de candeda de texto y texto estático usa el valor de alto como tamaño de altura.

Los posibles tipos definidos actualmente, así como los parámetros y controles asociados son:

Tipo	Denominación	Tipo parámetro	Control Windows	Descripción
0	Texto estático	-	Static	Solo para aspecto gráfico del diálogo
1	Valor boolenao	>0 TRUE ≤0 FALSE	Check Box	Valor booleano
2	Cadena de texto	char[]	Edit	Para definir cadenas de texto, que pueden tener varias líneas
3	Número real	double	Edit	Para definir números reales
4	Número entero	long	Edit	Para definir números eneros
5	Selección	int (positivo)	Radio Button	Para definir opciones de selección
6	Archivo a abrir	char[]	Edit + File Dialog	Permite que el usuario especifique un archivo existente para abrir
7	Archivo a guardar	char[]	Edit + File Dialog	Permite que el usuario especifique un archivo para escribir
8	Control Slide	long	Slide	Selección de un valor entero con una barra horizontal con etiquetas
9	Entero con Spin	long	Edit + Spin	Selección de un valor entero con un control de incremento-decremento

Los valores máximo y mínimo se aplican a los controles numéricos. En el caso de cadenas de texto se utiliza también el valor máximo para indicar su longitud máxima. Para el control de selección, el valor máximo indica el número de opciones.

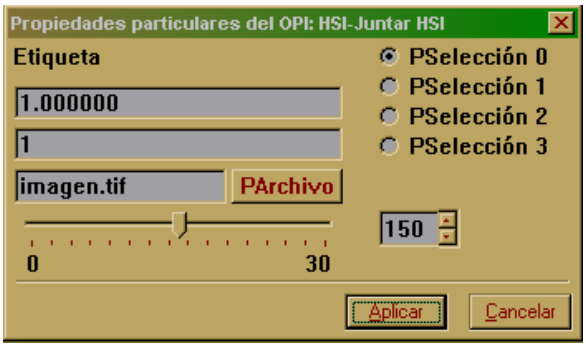
Excepto las cadenas de texto estático, el resto de valores definidos en un fichero de interfaz serán recibidos por el módulo correspondiente como cadenas de texto a través de la línea de comandos en el orden definido, y según lo expuesto anteriormente en el apartado 4.

El área disponible para dibujar controles es un rectángulo de 250 puntos de acho por 120 puntos de alto. Aunque en la carga no se verifican las coordenadas de los controles y sus tamaños, estos deben estar incluidos en este rectángulo para que sean mostrados.

Un ejemplo de archivo de interfaz puede ser este:

```
# Parámetros para el módulo de ejemplo
Etiqueta   0   0   0 200  22    0    0   0
PReal      3   0  25 200  22    0  100   1
PEntero    4   0  50 200  22 -100  100   1
PArchivo   6   0  75 200  22    0    0   imagen.tif
PSlider    8   0 100 200  18    0   30   15
PSelección 5 220   0 200  18    0    3   0
PSpin      9 220 100  50  18    0  300   150
```

El aspecto del diálogo correspondiente nada más aparecer por primera vez será el siguiente:

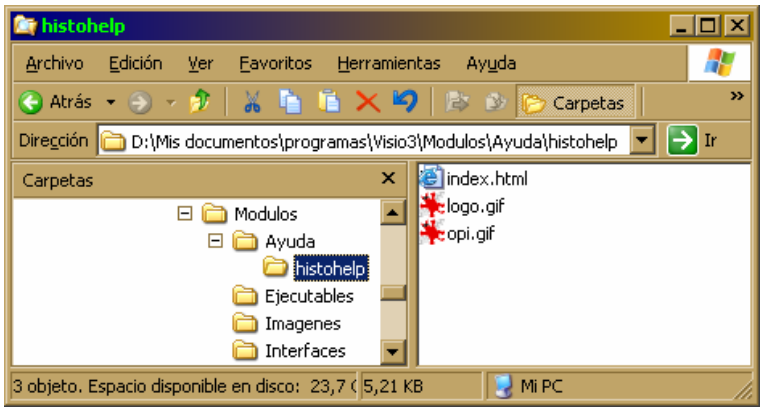


6. Sistema de ayuda

Cada OPI o módulo ejecutable puede tener asociado un conjunto de archivos de ayuda. Estos archivos, que deben tener una base HTML para poder ser visualizados con un cliente navegador de Internet, deben encontrarse dentro de una carpeta situada en el *directorio de archivos de ayuda* (ver apartado 2). El nombre de la carpeta de un módulo debe coincidir con el especificado en su definición dentro del *archivo de definición de módulos* (ver apartado 3.1). Cada módulo puede tener su propia carpeta para sus archivos de ayuda, y así conviene que sea. Aunque, en el caso de módulos semejantes o que deban compartir archivos comunes de ayuda, es posible utilizar la misma carpeta para ellos.

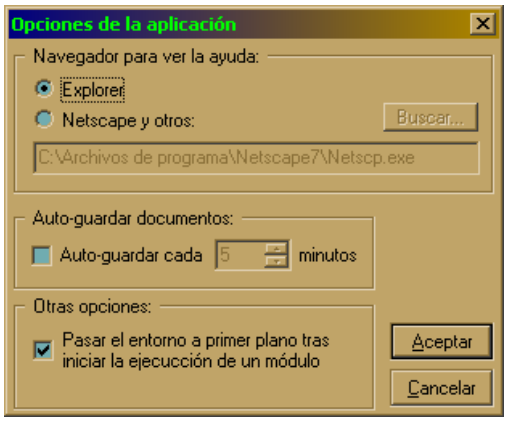
La ayuda de un módulo es opcional, y en el *archivo de definición de módulos* se puede obviar la indicación de la carpeta de ayuda, prescindiendo también de su carpeta y archivos de ayuda.

Dentro de la carpeta de ayuda de un módulo que dispone de ayuda, el único archivo obligatorio s *index.html*. Este es el archivo que se carga en el entorno Visual para mostrar la ayuda de un OPI. Pero para crear la ayuda de un OPI se pueden emplear imágenes u otros archivos de código que sean interpretados por el cliente navegador. Es decir, la ayuda de un módulo consiste en los archivos de un sitio web situados en su carpeta de ayuda.



6.1. Acceso a la ayuda desde el entorno Visual

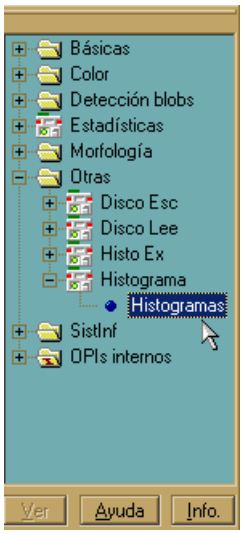
Para visualiza la ayuda desde el entorno Visual, primero se debe seleccionar el cliente navegador que utilizará la aplicación para mostrar la ayuda. Esto se hace en el diálogo al que se accede con la opción de menú *Archivo* → *Configurar aplicación*.



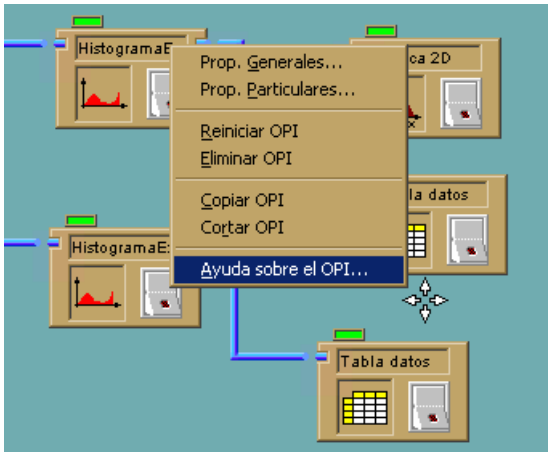
Se puede seleccionar entre usar el cliente MS. Explorer u otro programa navegador. En el segundo caso, se debe especificar la ruta de ese programa.

Para visualizar la ayuda de un OPI existen tres alternativas:

- En la ventana de selección de OPIs, cuando se selecciona un OPI que dispone de ayuda, se activa el botón *Ayuda*. Si se pulsa este botón, se abrirá el cliente navegador para mostrar la ayuda del OPI.



- Al pulsar con el botón derecho del ratón sobre un OPI pintado en un esquema, se muestra un menú emergente. Si el OPI tiene ayuda, aparecerá activada la opción *Ayuda sobre el OPO…*, que al ser pulsada abre el cliente navegador.



- Finalmente, se puede seleccionar la opción *Ayuda → Ayuda acerca de los OPIS...* del menú, o el botón correspondiente de la barra de herramientas, lo que permite seleccionar un OPI del esquema para ver su ayuda, en caso de que esté disponible la misma.

7. Anexos

7.1. Definición de OPIS básicos

```
Constante 1 1 constante
  generar 0 0 0 1 1 cons
  Genera una imagen de las características indicadas

Aritméticas 2 5 aritm
  Suma 0 2 10 1 1 ar_sum
  Suma dos imagenes
  Diferencia 1 2 2 1 1 ar_res
  Diferencia entre dos imagenes
  Producto 2 2 2 1 1 ar_pro
  Multiplica dos imágenes
  Division 3 2 2 1 1 ar_div
  Divide dos imágenes
  Media 4 2 2 1 1 m_media
  Media entre dos imágenes

Lógicas 3 8 logicas
  And 0 2 10 1 1 log_and
  Devuelve el producto lógico de varias imágenes
  Or 1 2 10 1 1 log_or
  Devuelve la suma lógica de varias imágenes
  Xor 2 2 10 1 1 log_xor
  Devuelve la suma lógica en módulo 2 de varias imágenes
  Not 3 1 1 1 1 log_not
  Devuelve la imagen complementaria de una imágen
  Supremo 4 2 10 1 1 log_sup
  Devuelve el maximo de dos funciones
  Infimo 5 2 10 1 1 log_inf
  Devuelve el minimo de dos funciones
  Intersec_Ext 6 2 10 1 1 log_inf
  Extensión de la intersección de imagenes binarias a escala grises
  Si_No_Negro 7 2 10 1 1 log_inf
  Si los pixels de la imagen 2 no son negros sustituyen a los de la 1

Elementales 4 4 morfo
  Erosión 0 1 1 1 1 m_ero
  Erosiona una imágen con el elemento estructurante dado como otra imágen
  Dilatación 1 1 1 1 1 m_dil
  Dilata una imágen con el elemento estructurante dado como otra imágen
  Apertura 2 1 1 1 1 m_ape
  Apertura morfológica
  Cierre 3 1 1 1 1 m_cie
  Cierre morfológico

RGB 5 2 rgb
  Separar_RGB 0 1 1 3 3 rgb_sep
  Devuelve imágenes de canales RGB separados
  Juntar_RGB 1 3 3 1 1 rgb_jun
  Combina las imágenes de canales RGB en una imágen
```

```
HSI-LAB 6 2 hsi
  Separar_HSI-Lab 0 1 1 3 3 hsi_sep
  Devuelve imágenes de canales HSI o Lab separados
  Juntar_HSI-Lab 1 3 3 1 1 hsi_jun
  Combina las imágenes de canales HSI o Lab en una imágen

Binarización 7 1 binariza
  Binarizar 0 1 1 1 1 binari
  Binariza una imagen poniendo a 0 o 255 sus pixeles

Histograma 9 1 histo histohelp
  Histogramas 0 1 1 1 9 histo
  Obtiene los histogramas de los canales de una imagen

Tophat 10 3 tophat
  tophat 0 1 1 1 1 m_top
  Tophat de la imagen de entrada
  Tophat_dual 1 1 1 1 1 m_topd
  Tophat dual de la imagen de entrada
  Tophat_extendido 2 1 1 1 1 m_tophe
  Tophat extendido de la imagen de entrada

Adelgazamiento 11 5 hitormiss
  Hit_or_Miss 0 1 1 1 1 m_hit
  Hit or Miss de la imagen de entrada
  Thickturn 1 1 1 1 1 m_tickt
  Thickturn de la imagen de entrada
  Thick 2 1 1 1 1 m_tick
  Thick de la imagen de entrada
  Geo-thickturn 3 2 2 1 1 m_gdsthurn
  Thickturn geodesico
  Geo-thick 4 2 2 1 1 m_gdsthick
  Thick geodesico

Geodesia_básica 12 6 geodesia
  Erosión_geo 0 2 2 1 1 m_gero
  Erosión geodésica de la imagen
  Dilatación_geo 1 2 2 1 1 m_gdil
  Dilatación geodésica de la imagen
  Reconstruccion 2 2 2 1 1 m_reco
  Reconstruccion de la imagen
  Apertura_Rec 3 1 1 1 1 m_rape
  Apertura por reconstrucción
  Cierre_Rec 4 1 1 1 1 m_rcie
  Cierre por reconstruccion
  Rec_erosión 5 2 2 1 1 m_reco2
  Reconstrucción por erosion

Watershed 13 1 watershed
  Watershed_gris 0 1 1 1 1 m_wat
  Watershed de una imagen de grises

Estadísticas 14 2 estadisticas
  Area 0 1 1 0 0 m_area
  Area de una imagen
  volumen 1 1 1 0 0 m_vol
  Volumen de una imagen

TogleMapping 15 1 togle
  Togle_mapping 0 2 2 1 1 m_togle
```

```
    Mejora de contraste morfologico

  Sup-Inf_vectorial 16 2 supinfvectorial
    Supremo_vect 0 2 2 1 1 m_supvec
    Supremo vectorial de dos imagenes
    Infimo_vect 1 2 2 1 1 m_infvec
    Infimo vectorial de dos imagenes

  Geodesia_color 17 6 geodesiacolor
    Erosión_geo_C 0 2 2 1 1 m_gero
    Erosión geodésica de la imagen
    Dilatación_geo_C 1 2 2 1 1 m_gdil
    Dilatación geodésica de la imagen
    Reconstru_C 2 2 2 1 1 m_reco
    Reconstrucción de la imagen
    Apertura_Rec_C 3 1 1 1 1 m_rape
    Apertura por reconstrucción
    Cierre_Rec_C 4 1 1 1 1 m_rcie
    Cierre por reconstruccion
    Rec_Erosión_C 5 2 2 1 1 m_reco2
    Reconstrucción de la imagen

  Gradiente 18 1 gradiente
    Gradiente 0 1 1 1 1 m_grad
    Gradiente de una imagen
```

7.2. Ejemplo de código para un módulo DLL

A continuación se muestran los archivos fuentes de un proyecto MS. Visual C++ 6 llamado *ModDLL* para crear un módulo DLL compatible MFC que implementa una sencilla operación de histograma.

Básicamente, el módulo recibe una única imagen de entrada de uno o más canales, y puede generar una o más salidas, siendo cada salida el histograma de un canal de la imagen de entrada. Si se especifican más salidas que canales tiene la imagen de entrada, el histograma del canal de índice más alto se repite en el resto de salidas.

Los histogramas generados consisten de imágenes de dos filas y 256 columnas, con valores de tamaño *byte*. La primera fila representa los índices de 0 a 256, y la segunda las cuentas para cada índice, escaladas a un entero en [0,100] respecto al valor máximo de las cuentas.

La salida generada por este módulo es compatible con las operaciones de representación de datos descritas en el documento *Operaciones de Visualización de Resultados*.

El asistente del MS. Visual C++ crea los archivos básicos, entre los que se encuentra el archivo *ModDLL.cpp*, con la función cargadora de la DLL:

```
// ModDLL.cpp : Defines the initialization routines for the DLL.

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

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

static AFX_EXTENSION_MODULE ModDLLDLL = { NULL, NULL };
```

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

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

          // Extension DLL one-time initialization
          if (!AfxInitExtensionModule(ModDLLDLL, hInstance))
                return 0;

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

Además se ha creado en el proyecto este archivo de cabecera para las declaraciones de funciones:

```
// funciones.h: declaracion de las funciones de esta DLL
// ---------------------------------------------------------------------

#if !defined(_MODDLL_EXP_FUNCIONES_)
#define _MODDLL_EXP_FUNCIONES_

// Para poder exportar objetos MFC sin problemas
#undef AFX_DATA
#define AFX_DATA AFX_EXT_DATA

// ---------------------------------------------------------------------
// Datos para imagen por defecto

#define IMG_X   4
#define IMG_Y   4
#define IMG_C   3
#define IMG_TAM IMG_X*IMG_Y*IMG_C

// ---------------------------------------------------------------------
// Cabecera de una imagen

typedef struct STCabecera
{
    WORD wX;
    WORD wY;
    WORD wC;
    WORD wB;
} StCabecera;

// ---------------------------------------------------------------------
// Funciones exportadas
// Están reflejadas en el archivo .DEF del proyecto en el mismo orden
```

```
// ---------------------------------------------------------------------

// Ejecuta las funciones de este módulo
// iFunc              -> Funcion a realizar (0, 1, 2...)
// iNumCtr            -> Numero de controles o datos de configuracion
// iNumEnt            -> Numero de entradas
// iNumSal            -> Numero de salidas
// strTemp            -> Directorio donde estan las imagenes de entrada
//                       o salida
// pStrDatos          -> Array con los datos de configuracion, todos
//                       como cadenas de texto
// pStrImgEntr        -> Array con los nombres de las imagenes de entrada
// pStrImgSal         -> Array con los nombres de las imagenes de salida

int Ejecutar(int iFunc, int iNumCtr, int iNumEnt, int iNumSal,
  CString& strTemp, CString* pStrDatos, CString* pStrImgEntr,
  CString* pStrImgSal);
```

También se ha creado el archivo de código donde se define la función Ejecutar:

```
// funciones.cpp: Código de las funciones de esta DLL
// ---------------------------------------------------------------------

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

// ---------------------------------------------------------------------
// Funciones exportadas
// Están reflejadas en el archivo .DEF del proyecto en el mismo orden
// ---------------------------------------------------------------------
// Funciones de Histograma
int Ejecutar(int iFunc, int iNumCtr, int iNumEnt, int iNumSal,
 CString& strDirTemp, CString* pStrDatos, CString* pStrImgEntr,
 CString* pStrImgSal)
{
    // Para depuracion
    #ifdef _DEBUG
          TRACE0("MODDLL: Modulo en ejecucion.\n");
          TRACE("Funcion: %d\nDatos: %d\nEntradas: %d\nSalidas: %d\n",
                iFunc, iNumCtr, iNumEnt, iNumSal);
          TRACE("Dir. Temporal: %s\n", strDirTemp);
          TRACE0("Datos: ");
          for (i=0; i<iNumCtr; i++)
                TRACE("%d:%s\n", i, pStrDatos[i]);
          TRACE0("Entradas: ");
          for (i=0; i<iNumEnt; i++)
                TRACE("%d:%s\n", i, pStrImgEntr[i]);
          TRACE0("\nSalidas: ");
          for (i=0; i<iNumSal; i++)
                TRACE("%d:%s\n", i, pStrImgSal[i]);
          TRACE0("\n");
    #endif

    // Requiere una imagen de entrada y almenos una salida
    if ((iNumEnt!=1) || (iNumSal<1))
          return 1;

    // No necesita datos de entrada
    if (iNumCtr>0)
          return 1;

    // Abrir la imagen de entrada
```

```
CString strNombre = strDirTemp + pStrImgEntr[0];
CFile file;
if(!file.Open(strNombre, CFile::modeRead | CFile::shareDenyWrite))
	return 2;

// Leer cabecera de la imagen de entrada
stCabecera cab;
if (file.Read(&cab, sizeof(cab)) < sizeof(cab))
{
	file.Close();
	return 3;
}

// Debe tener almenos tamaño 1 de ancho y de alto, y 1 canal
if ((cab.wX < 1) || (cab.wY < 1) || (cab.wC < 1) || (cab.wB != 1))
{
	file.Close();
	return 4;
}

// Calcula tamaño de la imagen y reserva memoria
UINT uTam = (UINT)(cab.wX * cab.wY * cab.wC);
BYTE* pDatos = new BYTE[uTam];
if (pDatos==NULL)
	return 5;

// Leer datos de la imagen de entrada
if (file.Read(pDatos, uTam) < uTam)
{
	file.Close();
	return 6;
}

// Cerrar archivo de entrada
file.Close();

// Reservar memoria para los histogramas de cada canal en %
int iCanales = cab.wC;
BYTE** pHistoPCien = new BYTE * [iCanales];
if (pHistoPCien==NULL)
{
	delete pDatos;
	return 5;
}

int i,j;
BOOL bError=FALSE;
for (i=0; i<iCanales; i++)
{
	pHistoPCien[i] = new BYTE[256];
	if (pHistoPCien[i]==NULL)
		bError=TRUE;
}

if (bError)
{
	for (i=0; i<iCanales; i++)
	{
		if (pHistoPCien[i]!=NULL)
			delete pHistoPCien[i];
	}
```

```
		delete pHistoPCien;
		delete pDatos;
		return 5;
	}

	// Reservar memoria para los histogramas de cada canal
	double** pHisto = new double * [iCanales];
	if (pHisto==NULL)
	{
		for (i=0; i<iCanales; i++)
			delete pHistoPCien[i];
		delete pHistoPCien;
		delete pDatos;
		return 5;
	}

	bError=FALSE;
	for (i=0; i<iCanales; i++)
	{
		pHisto[i] = new double[256];
		if (pHisto[i]!=NULL)
		{
			for (j=0; j<256; j++)
				pHisto[i][j]=0.0;
		}
		else
			bError=TRUE;
	}

	if (bError)
	{
		for (i=0; i<iCanales; i++)
		{
			if (pHisto[i]!=NULL)
				delete pHisto[i];
		}

		delete pHisto;
		for (i=0; i<iCanales; i++)
			delete pHistoPCien[i];
		delete pHistoPCien;
		delete pDatos;
		return 5;
	}

	// Genera histogramas
	int k;
	int iMaxX=cab.wX;
	int iMaxY=cab.wY;
	BYTE* pValor = pDatos;
	for (j=0; j<iMaxY; j++)
	{
		for (i=0; i<iMaxX; i++)
		{
			for (k=0; k<iCanales; k++)
			{
				pHisto[k][*pValor]+=1.0;
				pValor++;
			}
		}
```

```
}

// Escala histogramas para obtener valores en % con relacion
// al valor maximo
double dMax;
for (i=0; i<iCanales;i++)
{
	// Obtiene valor maximo para un histograma
	dMax=0.0;
	for (j=0; j<256;j++)
	{
		if (dMax<pHisto[i][j])
			dMax=pHisto[i][j];
	}

	// Escala valores del histograma
	for (j=0; j<256;j++)
		pHistoPCien[i][j]=(BYTE)((pHisto[i][j]/dMax)*100);
}

// Ya no hace falta la imagen
delete pDatos;

// Vector de indices
BYTE pIndices[256];
for (i=0; i<256; i++)
	pIndices[i]=i;

// Cabecera para las imagenes de histograma guardadas
cab.wX = 256;
cab.wY = 2;
cab.wC = 1;
cab.wB = 1;

// Guarda histogramas
bError=FALSE;
int iErr = 0;
j=0;
for (i=0; i<iNumSal; i++)
{
	strNombre = strDirTemp + pStrImgSal[i];
	if (file.Open(strNombre, CFile::modeCreate | CFile::modeWrite))
	{
		TRY
		{
			// Escribir cabecera
			file.Write(&cab, sizeof(cab));

			// Escribir fila de indices
			file.Write(pIndices, 256);

			// Escribir fila de datos
			file.Write(pHistoPCien[j], 256);

			// Incrementa numero de canales si queda,
			// y si no, repetriá el ultimo canal
			if (j<iCanales)
				j++;
		}
		CATCH( CFileException, e )
		{
```

```
                        bError=TRUE;
                }
                END_CATCH

                file.Close();

                if (bError)
                {
                        iErr = 8;
                        break;
                }
        }
        else
        {
                iErr = 7;
                break;
        }
    }

    // Libera datos de histogramas
    for (i=0; i<iCanales; i++)
                delete pHistoPCien[i];
        delete pHistoPCien;

    for (i=0; i<iCanales; i++)
        delete pHisto[i];
    delete pHisto;

    return iErr;
}
```

Finalmente conviene especificar las funciones exportables en el archivo de definiciones de la DLL (*ModDLL.def*), en este caso solamente la función *Ejecutar*, a la que también se le asigna el índice de función 1 dentro de la DLL. El nombre de la función ejecutar dentro de la DLL es una cadena generada por el entorno de MS Visual C++, y que se pede comprobar editando en modo ASCII-HEX en el entorno el archivo ".lib" de la DLL compilada.

```
; ModDLL.def : Declares the module parameters for the DLL.

LIBRARY      "HistoEx"
DESCRIPTION  'Histo Windows Dynamic Link Library'

EXPORTS
    ; Explicit exports can go here
    ?Ejecutar@@YAHHHHHAAVCString@@PAV1@11@Z   @1
```