imageMan

ImageMan 5.0 Release Notes

The ImageMan include file is now called IMGMAN.H.
The include file which contains low-level scanning function declares is called IMTWAIN.H.

The name of the ImageMan DLL has been changed. The new names are as follows:

16-bit DLL:	IMGMAN11.DLL
32-bit DLL:	IMGMAN31.DLL

The DIL & DEL files have also been changed using a new naming scheme:

16-bit files: IM11xxx.DIL or IM11xxx.DEL
32-bit files: IM31xxx.DIL or IM31xxx.DEL

The xxx represents the supported image type, for instance IM31TIF.DIL is the 32-bit TIFF import library.

This version includes functions for reading and writing images to/from memory:

- ImgOpenMem - opens a memory-based image for reading.
- ImgXBeginWriteMem / ImgXEndWriteMem - begins and ends writing an image to a memory buffer. To write the actual image bits, call the ImgXWriteBlock function in-between these calls.

Distributing Your Applications

In addition to the above listed DLL & DIL/DEL files, applications which include the Photo CD DIL will need to include the appropriate Photo CD DLL along with the DIL.

16 Bit Apps:	pcdlib.dll
32 Bit Apps:	pcdlib32.dll

Unlocking GIF and TIFF w/LZW Support

Because of a patent on LZW compression by Unisys Corp, we ship the GIF and TIFF w/LZW reader/writers in an encrypted zipfile (imlzw50.zip).
Before we can unlock these files, you must obtain a license from Unisys and fax us the first page of the license agreement. This is not a royaltly to DTI and is applicable to all vendors toolkits.

For more Information contact:

Mark T Starr
Unisys Corporation
PO Box 500
Blue Bell, PA 19424-0001

Voice:	215-986-4411
Fax:	215-986-5721

ImageMan Technical Support

You may obtain technical support for the ImageMan libraries by email, website, Phone, FAX, CompuServe and our own StarMan Bulletin Board system.

By joining the ImageMan email mailing list, you can keep informed of updates, tech tips and other important information from Data Techniques, Inc.   To join the list send email to: ***imageman-request@data-tech.com*** with the word ***subscribe*** in the message body.

Data Techniques, Inc.
300 Pensacola Rd
Burnsville, NC 28714

Email:	support@data-tech.com
WebSite:	www.data-tech.com
Tech Support:	704-682-4111 9am to 5pm EST
Fax:	704-682-0025
BBS:	704-682-4356 (2400-28.8K V.42, 24 Hours/Day)
CompuServe:	GO DATATECH
CompuServe ID:	74431,1412

int IMAPI ImgBrightness(hImage, nBrightness)
This function brightens or darkens the referenced image.
Example

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be altered.
nBrightness	**INT** Specifies the brightness factor to be applied to the image.

Return Value
The return is IMG_OK on success, an error value otherwise.

Comments
The value of nBrightness can range from -255 to 255. Postive numbers will darken the image while negative values will lighten it.

After calling this function you must call the ImgGetPalette function to get a handle to the new palette to be used when displaying the adjusted image.

Example: Using the ImgBrightness Function

```
/* This example opens and draws a PCX image then Brightens it & redraws
it. */

HANDLE hImage;
HPAL hPal;

hImage = ImgOpenSolo( "test.pcx", NULL );

...
hPal = ImgGetPalette( hImage );

if( hPal ) {
  SelectPalette( hDC, hPal, 0 );
  RealizePalette( hDC );
}

ImgDrawImage( hImage, hDC, lpDst, NULL );

// Now brighten Image & Redraw it

ImgBrightness( hImage, 100 );

// Delete the old palette
DeleteObject( hPal );

// Get a new palette since the image has different colors
hPal = ImgGetPalette( hImage );

if( hPal ) {
  SelectPalette( hDC, hPal, 0 );
  RealizePalette( hDC );
}

ImgDrawImage( hImage, hDC, lpDst, NULL );
```

int IMAPI ImgClose(hImage)
This function closes the referenced image.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be closed.

Return Value
The return value indicates whether the image was closed successfully. It is nonzero if the image was closed successfully. Otherwise, it is zero.

Comments
This call closes the image file and frees any resources which were allocated for processing the image. An application should use this call to close an image when it is no longer needed. Failure to call this function may cause memory or other resources to remain allocated needlessly.

HANDLE IMAPI ImgCopy(hImage, nWid, nHi, lpSrc, lFlags)
This function copies all or part of the given image into another image, size nWid x nHi, returning an ImageMan handle which represents the new image.
Example   Changes From Version 2.00

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to copy from.
nWid, nHi	**int** Specify the width and height of the resulting image.
lpSrc	**LPRECT** Specifies the portion of the source image to copy into the new image.
lFlags	**LONG** Flags specifying scaling method for monochrome images. This can be either COPY_DEL (delete scan lines & pixels), COPY_AND (preserve black pixels), or COPY_OR (preserve white pixels), or COPY_ANTIALIAS (Scale-to-Gray).

Return Value
Return value is a valid ImageMan image handle on success, 0 on error. In the event of an error the ImgGetStatus function will return the error code.

Comments
ImgCopy is one of the most important functions available in ImageMan. With it you can scale an existing image, or portion thereof, up or down however you like.

If the source image is not currently loaded, ImgCopy will perform the image copy in bands, allowing an application to create a low-resolution copy of a much larger image without having the larger image in memory at any point. In this case, the source image will still be unloaded when the ImgCopy function is completed.

The lFlags parameter allows you to control the scaling method used when shrinking monochrome images. The COPY_DEL flag, which is the default, simply eliminates scan lines and pixels without regard to color. This is the fastest scaling method. COPY_AND and COPY_OR allow you to scale an image while preserving either black or white pixels, respectively. These methods are slightly slower, yet can produce significantly superior results.

The COPY_ANTIALIAS option will use a sophisticated scaling algorythm to scale the image into a 16 color grayscale image. This method returns the best looking scaled images but is the slowest method. After copying an image using this method you'll need to get & realize its palette since it has changed.

Unfortunately, there is no easy way to determine which of these methods should be used for a given scaling operation. As a general rule, shinking an image less than half could be done using the COPY_DEL flag; anything greater than half-size should use the COPY_AND or COPY_OR flags.

int IMAPI ImgCreateDDB(hImage, hDC, nWid, nHi)

This function creates a Windows Device Dependent Bitmap (DDB) for the given image, enabling the image to be drawn more quickly than a DIB or Metafile.

Parameter	Type/Description
hImage	**HANDLE** Identifes the image to create a DDB for.
hDC	**HDC** Specifies a device context used when creating the DDB. The resulting DDB will reflect this DC in terms of color makeup.
nWid, nHi	**int** Specify the width and height for the resulting DDB. These are only used when converting a vector image to a DDB; otherwise they are ignored.

Return Value

The return value is IMG_OK if the DDB was created successfully, an error value otherwise.

Comments

For really fast drawing of a given image, this function should be called before ImgDrawImage is called. If a DDB exists for a given image, ImgDrawImage will use the DDB when drawing.

int ImgDrawImage(hImage, hDC,   lpDestRect,   lpSrcRect)
This function displays   the image referenced by hImage on the screen at the specified location and size.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the Image to be displayed.
hDC	**HDC** Identifies the device context for the display.
lpDestRect	**LPRECT** Points to a RECT data structure containing the logical screen coordinates to draw the image into.
lpSrcRect	**LPRECT** Points to a RECT data structure specifying the portion of the source image which will be displayed.

Return Value
The return value indicates whether the image was displayed successfully. It is nonzero if the image was displayed successfully. Otherwise, it is zero.

Comments
The ImgDrawImage does some simple decision making to determine how to draw the given image. If a device dependent bitmap (DDB) exists for this image, the resulting bitmap will be drawn to the screen with a StretchBlt call; otherwise, raster-based images will be drawn using StretchDIBits and vector-based images will be drawn by playing the metafile to the output device context. Note that if there is no currently loaded image handle (DIB, DDB, or WMF), the ImgDrawImage function will load one.

If lpSrcRect is NULL then the entire image will be displayed in lpDestRect.

int IMAPI ImgErrBox(hParent)
This function displays a Windows messagebox containing a textual description of the last ImageMan error.

Parameter	Type/Description
hParent	**HANDLE** Specifies the handle of Window to be used as the parent of the messagebox.

Return Value
The return value is IMG_OK.

int IMAPI ImgErrString(lpszBuf, nMaxLen)
This function copies the text describing the last error into a user supplied buffer.

Parameter	**Type/Description**	
lpszBuf	**LPSTR**	Points to a user defined buffer to receive the error string.
nMaxLen	**WORD**	Indicates the size in bytes of the buffer pointed to by lpszBuf.

Return Value
The return value indicates whether the error string was retrieved successfully. It is nonzero if the error string was retrieved successfully. Otherwise, it is zero.

Comments
The error string contains the filename and the description of the error separated by a semicolon. For some errors the string will not contain a filename so the first character will be a semicolon.

HANDLE IMAPI ImgFromClipboard()

This function returns an ImageMan image handle representing the clipboard contents.

This function takes no parameters.

Return Value

The return value is a valid ImageMan image handle on success, a NULL handle on failure.

Comments

ImageMan can currently accept the CF_DIB and CF_METAFILEPICT formats over the clipboard. An application should verify that one of these formats is available before calling this function; otherwise, a NULL handle will be returned.

An image created from the clipboard can be manipulated in the same manner as a "normal" ImageMan image, with the exception that it cannot be unloaded via the ImgUnload function. Also, the image will return IMG_MEM_BASED in the flags field of an ImgGetInfo call.

HANDLE IMAPI ImgFromDIB(hDIB)
This function creates an ImageMan image from a Windows device independent bitmap (DIB) and returns an ImageMan handle to the caller.

Parameter	Type/Description
hDIB	**HANDLE** A handle to a DIB in packed DIB format. ImageMan takes responsibility for the DIB when the function is completed.

Return Value
The return value is a valid ImageMan image handle on success, NULL on failure.

Comments
Once the call has completed, the application can treat this image as it would any other ImageMan image, with the caveat that the image cannot be unloaded via ImgUnload. Also, a call to ImgGetInfo will return the IMG_MEM_BASED flag in the lFlags parameter to indicate that this image has no external file to recreate itself from.

HANDLE IMAPI ImgFromWMF(hWMF, lpRect)
This function returns an ImageMan image handle given a handle to a Windows Metafile and it's bounding box.

Parameter	**Type/Description**
hWMF	**HANDLE** Handle to a Windows Metafile. ImageMan taks responsibility for the Metafile upon calling this function.
lpRect	**LPRECT** Points to a RECT structure which contains the metafile's bounding box.

Return Value
The return value is a valid ImageMan image handle on success, NULL on failure.

Comments
As with the ImgFromDIB function call, once the handle   has been returned, the application can treat it as it would any other ImageMan image, except that it cannot be unloaded via the ImgUnload function. Calling ImgGetInfo for the returned image will reveal that the IMG_MEM_BASED flag is set, indicating that the image has no external representation to recreate itself from.

int IMAPI ImgGamma(hImage, nGamma)
This function performs gamma correction on the given hImage.

Parameter	Type/Description
hImage	**HANDLE** Image to gamma-correct.
nGamma	**INT** Gamma-correction value (10-50)

Return Value
The return value is IMG_OK on success, an ImageMan error code on failure.

Comments
Acceptable nGamma values range from 10 (1.0) to 50 (5.0).

HANDLE IMAPI ImgGetDDB(hImage, hDC)
This function returns a handle to a Windows device-dependent bitmap (DDB) which represents the given image.

Parameter	**Type/Description**
hImage	**HANDLE** identifies the image.
hDC	**HDC** The image is created to be compatible with this device context. This paramter cannot be NULL.

Return Value
The return value is a handle to a Windows bitmap on success, NULL on failure.

Comments
The returned image will take on the color characteristics and bit-depth of the passed device context; therefore, it's important to make sure that a palette has been selected into the device context before calling this function

HANDLE IMAPI ImgGetDIB(hImage, bNewDib, lpSrcRect)
This function retrieves a handle to hImage in packed DIB format.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the Image whose bits should be retrieved.
bNewDIB	**BOOL** If false, the function returns a handle to ImageMan's internal DIB; if true, ImageMan allocates a new DIB and returns that (in this case, the caller is responsible for the returned handle).
lpSrcRect	**LPRECT** Identifies the portion of the image to be extracted in DIB format. Any rectangular part of the image may be specified.

Return Value
The return value is a global handle to a packed DIB if the function is successful, NULL if not.

Comments
This function should be used to retrieve the bits of a raster image in DIB format. The DIB format maintains all the color information of the original image.

Note
ImgGetDIB works only with raster images; using this function on vector images will result in an IMG_BAD_TYPE error being returned to your application.

LPSTR IMAPI ImgGetExt()
This function returns a string containing the image formats currently supported by ImageMan.

This function takes no parameters.

Return Value
The returned string is in the Windows common dialog format, and is therefore suitable for passing directly into the GetOpenFileName function. The caller is responsible for freeing the string when done with it (call GlobalFreePtr(lpString) to release it).

LPBITMAPINFO IMAPI ImgGetInfo(hImage, lpFlags)
This functions returns a pointer to an BITMAPINFO struct which defines the image.

Parameter	Type/Description
hImage	**HANDLE** Identifies the image.
lpFlags	**LPINT** Points to an integer which receives the flags for the given image (see below for a list of possible flags).

Return Value
Returns a pointer to ImageMan's internal BITMAPINFO struct for the given image. Since this struct belongs to ImageMan, the caller should not alter the values.

Comments
The currently defined flags (as found in IMGMAN.H) are as follows:

IMG_RENDER_SELF When set, ImgDrawImage will call the DIL to render the image to the screen. This allows a library to have more control over the drawing of an image. In general, it probably won't help you much to look at this flag.

IMG_PRINT_SELF When set, ImgPrintImage will call the DIL to handle printing the image. This is particularly useful for the EPSF DIL, which needs to output PostScript code directly to the printer.

IMG_PRNT_VECTOR Indicates that the image will be printed as a metafile.

IMG_DISP_VECTOR Indicates that the image will display as a metafile.

IMG_MEM_BASED This flag is set if the image came from a memory-based source (clipboard, hDIB, hWMF).

int IMAPI ImgGetPage(hImage, lpPage)
This function places the currently active page (0-based) for the given image in the variable pointed to by lpPage.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to get the information for.
lpPage	**LPINT** Location to place the returned page count into.

Return Value
The return value is IMG_OK on success, an error value otherwise

HANDLE IMAPI ImgGetPalette(hImage)
This function returns a handle to a GDI logical palette which describes the color content of an image.

Parameter	**Type/Description**	
hImage	**HANDLE**	Identifies the image you want a palette for.

Return Value
The return value is a handle to a GDI logical palette. It is NULL if an error occurred.

Comments
The handle returned from this function is created via the GDI CreatePalette function, and must be selected into a device context and realized before it has an effect on image display.

Note
Your application is responsible for deleting this object when it is no longer of any use.

DWORD IMAPI ImgGetROP(hImage)
This function returns the current ROP code for the given image.

This function takes no parameters.

Return Value
The return value is the current ROP code

int IMAPI ImgGetStatus()
This function returns the current status of ImageMan.

This function takes no parameters.

Return Value
The return value is the error code set by the last ImageMan function call. If this function returns a value other than IMG_OK then the ImgErrBox() or ImgErrStr() functions can be called to get a more detailed description of the error.

Comments
This function returns the status of only the last ImageMan function call. For a complete list of valid status values, refer to Appendix A.

HANDLE IMAPI ImgGetWMF(hImage, lpRect)
This functions returns a handle to a Windows Metafile that represents hImage. This function is only valid for vector images.

<u>**Parameter**</u>	<u>**Type/Description**</u>
hImage	**HANDLE** Identifies the image.
lpRect	**LPRECT** Points to a rectangle which will receive the metafile's bounding box. This is necessary to display the metafile with the proper aspect ratio.

Return Value
The return value is a handle to a Windows Metafile on success, a NULL handle on failure.

Comments
This function should only be called for images which have the IMG_DISP_VECTOR flag set. Note that this function will load the metafile if it isn't already loaded (or has been explicitily unloaded)

int IMAPI ImgInit()
This function initializes the ImageMan library, and must be called before any other ImageMan calls are made.

This function takes no parameters.

Return Value
The return value is IMG_OK if ImageMan was initialized without errors. Otherwise, it is an ImageMan error code

int IMAPI ImgLoad(hImage, lpRect)
This function forces all or part of a given image to be "loaded", i.e., read in from disk into a DIB or Metafile.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to load.
lpRect	**LPRECT** Specifies the portion of the image to load, in image coordinates.

Return Value
Returns IMG_OK on success, an error value otherwise.

Comments
An image is not loaded when ImgOpen is called; instead, ImageMan gives each application explicit control over when tor if he image is loaded. While this function loads the image explicitily, there are several which will, if called when the image is not loaded, implicitly load the entire image. These are as follows:

ImgDrawImage
ImgGetDIB
ImgGetWMF
ImgGetDDB
ImgToClipboard
ImgPrintImage

If ImgLoad is used to load a portion of an image, subsequent calls to ImgGetInfo will reflect the image's loaded dimensions, not it's original size. To restore the image at it's original dimensions, use the ImgUnload function.

Note
The ImgCopy function operates as a special case in order to allow an application to reduce a large image to a smaller one without having to have the large image entirely in memory. If the image to copy isn't loaded at the time of the ImgCopy call, it will be loaded in small bands and copied piecemeal to the destimation image. When the ImgCopy function is completed, the source image will not have a loaded image to draw from. Of course, if the source image is already loaded, ImgCopy will simply copy it to the destination and the original image will remain loaded and intact.

int IMAPI ImgMirror(hImage, bVert, bHoriz)
This function mirrors the referenced image along the specified axis.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be closed.
bVert	**BOOL** If True then the image is mirrored vertically.
bHoriz	**BOOL** If True then the image is mirrored horizontaly.

Return Value
The return value indicates whether the image was transformed successfully. It is nonzero if the operation was successful. Otherwise, it is zero.

Note
This function changes the image that you pass to it. If you want to preserve the original image, use the ImgCopy function to copy the image and mirror the copy.

HANDLE ImgOpenEmbedded(hFile, lOffset, lLen, lpExt)

This functions opens and initializes an image file that is embedded in another file.

Parameter	**Type/Description**
fHand	**INT** Specifies the DOS file handle of the (previously opened) file in which the image is embedded.
lOffset	**LONG** Specifies the offset (in bytes) of the embedded image from the beginning of the file containing it.
lLen	**LONG** Specifies the length (in bytes) of the embedded image within the containing file.
lpExt	**LPSTR** Specifies that ImageMan should process the file as an image with the extension specified. If NULL, ImageMan will auto-detect the image format.

Return Value

The return value specifies the handle used to refer to the image. It is NULL if the file could not be opened.

Comments

This function allows you to open an image file that is contained within another file. ImageMan will determine the image type from the lpExt parameter, or will auto-detect the image format if the lpExt is incorrect or NULL. Before ImageMan can access this file, your application must open the file - ImageMan will not open or close the file at any point

HANDLE ImgOpenSolo(lpFilename, lpExt)

This function opens the specified image and returns a handle to be used with the other ImageMan functions.

<u>**Parameter**</u>	<u>**Type/Description**</u>
lpFileName	**LPSTR** Points to a buffer containing the path & filename of the image to open.
lpExt	**LPSTR** Specifies that ImageMan should process the file as an image with the extension specified.

Return Value

The return value specifies the handle used to refer to the image. It is NULL if the file could not be opened.

Comments

ImageMan will determine the image type from the extension passed in the filename. If the image file has a nonstandard extension (i.e. a PCX file with extension other than .PCX) the lpExt parameter can specify which format the image should be processed as. For instance, to open a PCX file called junkpcx.jnk, you would use the following ImgOpen call:

```
ImgOpenSolo("junkpcx.jnk","PCX")
```

Note

You normally should not need to specify the lpExt parameter, as for the most part the images you encounter will have valid extensions; in these cases, lpExt should be set to NULL.

int IMAPI ImgPageCount(hImage, lpCnt)
This function returns the number of pages that are in the image file into the variable pointed to by lpCnt.

<u>**Parameter**</u>	<u>**Type/Description**</u>
hImage	**HANDLE** Identifies the image to get the page count for.
lpCnt	**LPINT** Points to an integer which will receive the page count.

Return Value
The return value is IMG_OK on success, an error value otherwise.

int ImgPrintImage(hImage, hPrnDC, lpDestRect, lpSrcRect)
This function prints the specified image or portion thereof on the printer at the specified location and size.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the Image to be displayed.
hPrnDC	**HDC** Identifies the device context for the printer.
lpDestRect	**LPRECT** Points to a RECT data structure containing the area on the printer in which the image will be printed.
lpSrcRect	**LPRECT** Points to a RECT data structure con taining the coordinates of the portion of the source image which should be printed.

Return Value
The return value indicates whether the image was printed successfully. It is nonzero if the image was printed successfully. Otherwise, it is zero.

Comments
If lpSrcRect is NULL then the entire image will be displayed in the area specified by the RECT pointed to by lpDestRect.

Note
Raster images cannot be printer on vector devices such as plotters. If you attempt to print a raster image on a vector device ImageMan will return the IMG_BAD_PRN error code.

HANDLE IMAPI ImgIncreaseColors(hImage, nBitDepth)
This function returns a new image with the specifed bit depth.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be reduced.
nBitDepth	**INT** Specifies the bit depth of the new image.

Return Value
The return value indicates whether the image was processed successfully. A return value of NULL indicates the color reduction failed, while a non-zero value is the ImageMan handle to the new image.

Comments
Allowable values for the nBitDepth parameter are 4, 8, and 24 bits.

Note
This function does not alter the original image; rather, it returns a new image based on the passed parameters.

HANDLE IMAPI ImgReduceColors(hImage, nColors, nFlags)
This function returns a new image with the specifed number of colors.

Parameter	Type/Description
hImage	**HANDLE** Identifies the image to be reduced.
nColors	**INT** Specifies the number of colors in the new image (currently limited to 256, 16, or 2).
nFlags	**INT** Specifies options to be used when reducing the colors.

Return Value
The return value indicates whether the image was processed successfully. A return value of NULL indicates the color reduction failed, while a non-zero value is the ImageMan handle to the new image.

Comments
Allowable values for the nColors parameter are 2, 16, and 256.Option Flags include:

IMG_BURKES	Specifies the Burkes Dithering algorithm.
IMG_BAYER	Specifies the Bayer Dither be used.
IMG_FLOYD	Specifies the Floyd Steinberg dither.
IMG_GRAYSCALE	Specifies the image should be converted to grayscale.
IMG_FIXEDPALETTE	Dithers the image to a fixed palette that is internal to ImageMan. This is extremely useful for displaying several color images simultaneously.

Note
This function does not alter the original image; rather, it returns a new image based on the passed parameters.

HANDLE IMAPI ImgRotate(hImage, nDegree, rgbCol)
This function rotates the referenced image in the counterclockwise direction and returns a handle to the new image.
Changes from V 2.00

Parameter	Type/Description
hImage	**HANDLE** Identifies the image to be rotated.
nDegree	**INT** Specifies the number of degrees (measured counterclockwise) to rotate the image.
rgbCol	**COLOREF** Specifies the background color when rotating to non 90 degree multiples.

Return Value
The return value indicates whether the image was rotated successfully. A return value of NULL indicates the rotation failed while a non-zero value is the ImageMan handle to the newly rotated image.

Note
This function does not alter the original image; rather, it returns a new image based on the passed parameters.

ImgRotate now supports rotating to any angle.

int IMAPI ImgSetPage(hImage, nPage)
This function prepares the given image to read from image number nPage in the file.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies image to set page for.
nPage	**int** 0-based page number to seek to.

Return Value
The return value is IMG_OK on success. If an invalid page number is passed to the function, it will return IMG_INV_PAGE.

int IMAPI ImgSetROP(hImage, dwNewROP)
This function sets the Windows ROP code to be used when displaying and printing the given image.

Parameter	**Type/Description**
hImage	**HANDLE** Handle to the image to be affected.
dwNewROP	**DWORD** Contains the ROP code to be used when displaying or printing raster images.

Return Value
The return value should be IMG_OK.

Comments
The ROP code is only used when displaying or printing raster images. An application can determine whether an image is in vector format by looking at the IMG_DISP_VECTOR and IMG_PRNT_VECTOR flags returned by ImgGetInfo.

int IMAPI ImgSetStatusProc(hImage, lpStatProc, lCnt, dwUser)
This function allows an application to register a status function to be called during image loading with the completed percentage of the loading process.
Example Changes From Version 2.00

Parameter	Type/Description
hImage	**HANDLE** Specifies the image to register the status function for.
lpStatProc	**STATUSPROC** Pointer to the function to call each interval.
lCnt	**LONG** Specifies the interval between calls to lpStatProc. This is based on the size of the created image.
dwUser	**DWORD** User-defined info to be passed to status procedure.

Return Value
The return value is IMG_OK.

Comments
Use this function to implement a bar-chart during loading of the image file.

You can now return a 0 from the StatusProc to cancel the current operation. This means that **you must return a nonzero value from the StatusProc to continue the current operation.**

void IMAPI ImgShutdown()
This function shuts down the ImageMan library, and must be called before the application terminates to ensure that all internal objects are properly released.

This function takes no parameters.

int IMAPI ImgToClipboard(hImage)
This function places the given ImageMan image on the clipboard in CF_DIB or CF_METAFILEPICT format.

Parameter	**Type/Description**
hImage	HANDLE Identifies the image to place on the clipboard.

Return Value
The return value is IMG_OK if the image is successfully placed on the clipboard, IMG_ERR otherwise.

Comments
The image is placed on the clipboard in either CF_DIB or CF_METAFILEPICT format, depending on the value of the IMG_DISP_VECTOR flag (as returned from ImgGetInfo). If the flag is set, the image is placed in CF_METAFILEPICT format.

int IMAPI ImgUnload(hImage, bDIB, bDDB, bWMF)
This function causes one or all of the given image's internal image representations (i.e., DIB, Device Dependent Bitmap, or Metafile) to be unloaded from memory.

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be unloaded.
bDIB	**BOOL** True if the DIB portion of the image should be unloaded.
bDDB	**BOOL** True if the Device Dependent Bitmap portion of the image should be unloaded.
bWMF	**BOOL** True if the Windows Metafile portion of the image should be unloaded.

Return Value
The return value is IMG_OK on success, an error value otherwise.

Comments
This function is useful for conserving memory. Note that even if the internal image is unloaded, the image information returned from ImgGetInfo remains consistent. Note also that no single image could have both a Metafile and a DIB loaded simultaneously—vector images will load a Metafile, raster images will load a DIB; both image types can create a DDB through the use of the ImgCreateDDB function.If the ImgLoad function is used to load a portion of the image, subsequent calls to ImgGetInfo will reflect the image's "new" dimensions, i.e., the dimensions specified to load the image portion. The original image, and it's original dimensions, cannot be accessed unless ImgUnload is called, followed by an ImgLoad.

Note
This function does nothing to memory-based images, and will simply return IMG_OK.

int IMAPI ImgXWriteDIB(lpFile, lpExt, hDIB, hOptBlk, hWnd, lOpts)
This function is used to write a DIB in memory to a supported image format.

Parameter	**Type/Description**
lpFile	**LPSTR** Points to asciiz string containing the name of the file to save the image in or NULL.
lpExt	**LPSTR** Points to a asciiz string containing the extension of the image format to use when saving this image or NULL.
hDIB	**HANDLE** This is a global memory handle to a Windows DIB in CF_DIB format.
hOptBlk	**HANDLE** Handle to an option block for this image or NULL
hWnd	**HANDLE** This is a handle to the applications main wnidow.
lOpts	**LONG** Contains flags which set various options when saving the image.

Return Value
The return value is IMGX_OK if the image was saved, otherwise it is IMGX_ERR.

Comments
If you need to write a DIB in sections (bands) then you will need to call the low-level functions ImgXBeginWrite, ImgXWriteBlock, and ImgXEndWrite. The filename pointed to by lpFile must contain a supported file extension   unless the lpExt parameter points to a string containing the format extension to use. The lpFile parameter can be NULL if you specify the IMGXOPT_FILE_PROMPT option in the lOpts parameter.If you are saving an image and wish to use a non-standard file extension then the lpExt parameter must point to a string containing the 1-3 character extension of the image type you wish to save the image as. This parameter can be set to NULL when saving images using a standard file extension. For instance, if you wish to save an image as a TIFF file in a file called SAMPLE.001 you would have to pass a pointer to "SAMPLE.001" as the lpFile parameter and a pointer to a string, "TIF", as the lpExt parameter.

The lpOpts parameter allows you to specify some additional parameters to the function. Multiple options can be specified by logically or-ing them i.e. IMGXOPT_COMPRESS | IMGXOPT_OVERWRITE.The following options can be specified in the lOpts parameter:

IMGXOPT_COMPRESS This option causes the image to be written in compressed form if the image format supports compression otherwise it is ignored.

IMGXOPT_OVERWRITE This option causes any existing file with the same name to be overwritten.

IMGXOPT_OVERWRITE_PROMPT This option causes the function to prompt the user if an existing file has the same name. The user can then select to overwrite the existing file or select a new filename.

IMGX_FILE_PROMPT This options causes the common file save dialog to be displayed to prompt the user for the output filename. The user can also select the format of the file to export. This filename will override any filename specified in the lpszFilename parameter. If you specify this option you can pass NULL as the lpFile parameter.

int IMAPI ImgXGetExt(lpszBuf)
This function is used to retrieve the extensions currently supported by ImageMan/X.

Parameter	**Type/Description**
lpszBuf	**LPSTR** Points to the buffer to hold the extension list.

Return Value
The return value depends on the lpszBuf parameter: if lpszBuf is NULL, the return value is the size of the buffer needed to hold the extension string; if lpszBuf is not NULL, the return value is undefined.

Comments
This function copies the list of extensions currently supported by ImageMan/X into the buffer pointed to by lpszBuf. lpszBuf should be large enough to contain the string; to determine the string length, call this function with lpszBuf set to NULL. The returned extensions are of the form:

```
ext;description~ext;description~....~ext;description
```

where each extension is separated by a '~' (tilde), "ext" is the (up to) 3 letter file extension and "description" is a short textual description of the extension. For example:

```
PCX;Publisher’s Paintbrush (*.pcx)~BMP;Windows Bitmap (*.bmp)
```

The short textual description is designed to be used in a dialog box as a description of each extensions’s format.

int IMAPI ImgXOptBlkAdd(hOptBlk, lpszKey, lpszValue)
This function adds a key/value pair to an Option Block (OptBlk).

Parameter	**Type/Description**
hOptBlk	**HANDLE** Handle to the opt block to add or alter (obtained from ImgXOptBlkCreate)
lpszKey	**LPSTR** Points to the key value to add or alter.
lpszValue	**LPSTR** Points to the new value for the key.

Return Value
The return value is a handle to the OptBlk.

Comments
This function adds a new key value to an option block or alters an existing key value. The opt block consists of a set of key/value pairs of the form "key = value", for instance:

```
Compress = DEFAULT
CPU = INTEL
```

To remove a key from the opt block, call ImgXOptBlkAdd, setting the key to the key you wish to remove and the value to NULL:

```
ImgXOptBlkAdd(hOptBlk, "Compress", NULL);
```

All optblock strings are case insensitive.

Note
The option block string must be specified exactly as shown; i.e., there must be a space to either side of the '='.

HANDLE IMAPI ImgXOptBlkCreate(lpszInit)
This function creates an Option Block (OptBlk) and optionally initializes it.

Parameter	Type/Description
lpszBuf	**LPSTR** Points to a buffer containing initial values for the opt block.

Return Value
The return value specifies a handle used to reference this OptBlk in the future.

Comments
This function creates and, optionally, initializes an opt block. An opt block is a set of parameters which define the current exporting procedure. When initializing an opt block (lpszInit is not NULL), each opt block parameter is separated by a carriage return (char value 13). For example:

```
lpszBuf = "Compress = ON\rEmbed = 5235,5"
```

NOTE
The entries to an OptBlk must have spaces around the equals ('=') symbol for the block to properly recognize the values.

int IMAPI ImgXOptBlkDel(hOptBlk)
This function deletes an Option Block (OptBlk) from memory.

Parameter	**Type/Description**
hOptBlk	**HANDLE** Handle to the opt block to delete.

Return Value
The return value is NULL if the function is successful.

Comments
This function deletes an opt block. It should be called when you’re finished with a given opt block to clean up memory.

int IMAPI ImgXOptBlkGet(hOptBlk, lpszKey, lpszBuf)
This function is used to retrieve the value for a particular key from an Option Block (OptBlk).

Parameter	Type/Description
hOptBlk	**HANDLE** Handle to opt block obtained from ImgXCreateOptBlk.
lpszKey	**LPSTR** Key value you wish to retrieve.
lpszBuf	**LPSTR** Buffer to contain returned key value.

Return Value
The return value from this function is zero (0) on success, 1 on failure.

Comments
This function places the value of the requested key into a user-supplied buffer. Note that opt block values cannot exceed 80 characters in length.
This function can be used to obtain ALL of the key/value pairs in an opt block by setting the lpszKey parameter to NULL. Upon return, lpszBuf will contain an LPSTR which points to the actual opt block string containing all key/value pairs. Each line is separated by a carriage return ('\r').

HANDLE IMAPI ImgXBeginWrite(lpFile, lpDIB, hOptBlk, lpInfo)
This function is used to begin exporting an image.

Parameter	**Type/Description**
lpFile	**LPSTR** Points to asciiz string containing name of file to save the image in.
lpDIB	**LPBITMAPINFO** Points to a BITMAPINFO struct which defines the image.
hOptBlk	**HANDLE** Handle to an option block for this image. If this is NULL, ImageMan/X will assume logical defaults.
lpInfo	**LPXINFO** Points to an XINFO struct defining the export procedure for this image.

Return Value
The return value is the handle to use when referencing this export job in future ImageMan/X functions; it is NULL if an error occurred.

Comments
This is the function which initiates the export of an image. Before calling this function, you should create an OptBlk with the desired options (or leave this NULL), Fill in the XINFO struct pointed to by lpInfo with the desired values, and etermine the name of the file you will be exporting. This may be done through the OptBlk or by filling in the lpFileName parameter; if the lpFileName parameter exists, it will supersede a filename in an OptBlk.

int IMAPI ImgXWriteBlock(hJob, nLines, lpBits, lpInfo)
This function writes a block of bits to the export file specified by hJob.

Parameter	**Type/Description**
hJob	**HANDLE** Handle identifying job (from ImgXBeginWrite).
nLines	**int** Number of raster lines in this output block.
lpBits	**LPSTR** Pointer to block of bits to be output.
lpInfo	**LPXINFO** Points to the XINFO struct which describes the current state of this export job.

Return Value
The return value is IMGX_OK if the bits are exported successfully; otherwise, the return value is an ImageMan/X error code.

Comments
The given bits are written subsequent to any bits currently output -- there is no facility for writing blocks of the image out of order. Note that since the output file may be a backwards format (i.e., last line in image appears first in file, as in Windows BMP & DIB formats), you have to be careful when writing out your image in sections -- make sure you begin with the correct row (first or last, depending on the image file format). You can do this by checking the lFlags parameter of the XINFO struct for the XF_BACKWARDS flag –if this flag is set, the strips should be written from bottom to top.

int IMAPI ImgXWriteBMPBlock(hJob, hDC, hBmp, nLines, lpInfo)
This function writes a bitmap to the export file specified by hJob.

<u>Parameter</u>	<u>Type/Description</u>
hJob	**HANDLE** Handle identifying job (from ImgXBeginWrite).
hDC	**HANDLE** Handle to a device context which has the same color makeup as the image.
hBmp	**HANDLE** Handle to the bitmap to output. This bitmap should not be selected into a device context at the time it is passed to this function.
nLines	**int** Number of raster lines in this output block.
lpBits	**LPSTR** Pointer to block of bits to be output.
lpInfo	**LPXINFO** Points to the XINFO struct which describes the current state of this export job.

Return Value
The return value is IMGX_OK if the bitmap is exported successfully; otherwise, the return value is an ImageMan/X error code.

Comments
The given bitmap is written subsequent to any bits currently output -- there is no facility for writing blocks of the image out of order. Note that since the output file may be a backwards format (i.e., last line in image appears first in file, as in Windows BMP & DIB formats), you have to be careful when writing out your image in sections -- make sure you begin with the correct row (first or last, depending on the image file format). You can do this by checking the lFlags parameter of the XINFO struct for the XF_BACKWARDS flag -- if this flag is set, the strips should be written from bottom to top.
This function can be used to write several bitmaps in sequence, thus allowing a large bitmap to be written in several smaller sections.

Note
This function can handle a bitmap as large as you can create, so there is no need to export a bitmap in strips unless your application doesn't have the entire image as a single bitmap.

int IMAPI ImgXEndWrite(hJob, lpXInfo)

This function ends an export job, closes all files associated with the job, and frees all memory allocated for the job.

Parameter	Type/Description
hJob	**HANDLE** Handle indentifying this job (from ImgXBeginWrite).
lpXInfo	**LPXINFO** Points to an XINFO struct which contains results of the operation.

Return Value

The return value is IMGX_OK on success; otherwise it is an ImageMan/X error code.

HANDLE IMAPI ImgXFileDialog(hOptBlk, hParentWnd)
This function displays a file dialog which allows the user to enter a filename for output, along with a file format to use, including any specific options for that format.

Parameter	**Type/Description**
hOptBlk	**HANDLE** Specifies the Option Block (OptBlk) which will be used with the Dialog.
hParentWnd	**HANDLE** Handle to the parent window for the dialog box.

Return Value
The return value is a handle to an option block to be used in an ImgXBeginWrite function call.

Comments
Upon return from this function, you can pass the returned hOptBlk directly into an ImgXBeginWrite call. Through the dialog function, an application can specify all aspects of an export job.

Note
If you pass a NULL hOptBlk, ImageMan/X will generate an OptBlk which reflects the options through the dialogs.

int   IMAPI ImgXErrString(lpBuf)
This function places a string containing the last ImageMan/X error into lpBuf.

Parameter	**Type/Description**
lpBuf	**LPSTR** Points to buffer which will contain the error string text.

Return Value
Returns IMGX_OK.

Comments
The returned string is guaranteed to fit in a buffer 300 characters long.

HANDLE IMAPI ImgXErrBox(hWnd)

This function produces a message box detailing the last ImageMan/X error that occurred, including the name of the file that produced the error and the module that encountered the error.

Parameter	**Type/Description**
hWnd	**HWND** Handle to the parent window for the message box.

Return Value

Returns IMGX_OK.

HANDLE IMAPI ImgXGetStatus(void)
This function returns the current ImageMan/X status. The status will correspond with one of the ImageMan/X status codes found in the IMGX.H include file.

This function takes no arguments.

Comments
The status of ImageMan/X is updated after each ImageMan/X function call.

int IMAPI ImgXAbort(hImage)
This function instructs ImageMan/X to delete the current export file when the ImgXEndWrite function is called to end writing the file.

Parameters	**Type/Desctiption**	
hImage	**HANDLE**	Handle to an image returned from ImgXBeginWrite.

Comments
Calling this function has no effect until the ImgXEndWrite function is called. If an error is encountered, no other processing should be done

ImageMan 5.0 API

Required Functions
These functions must be included for ImageMan to function properly.

Image Initialization Functions
These functions create an "ImageMan image handle" which is subsequently used for all imaging functions (display, rotation, color reduction, etc..). The image handle can be based on a disk file (TIF or PCX image) or can be memory based (the Windows clipboard).

Image Export Functions
This function group allows you to create an external representation from an ImageMan image handle. The representation can be an external disk file (TIF or PCX file, for instance) or a memory-based object (DIB, WMF or DDB).

Manipulation Functions
This group includes such things as image rotation, color reduction, brightness and gamma adjustment, and image scaling. Some of these functions return a handle to a new image, while some of them affect the image handle passed to them.

Information Functions
These functions provide information about the ImageMan system or a given image handle. This includes a list of currently supported file types for import or export, image width, height and color makeup, and error codes and messages.

Multi-Page Functions
These functions provide the mechanism for determining the number of pages and moving between pages in a multi-page image.

Image Display/Printing Functions
Functions that are used for displaying/printing an image. Also, functions which affect the way in which an image is displayed/printed.

Scanning Functions
Functions that are used to control TWAIN compatible scanners, digital cameras and frame grabbers.

Required Functions

All programs using ImageMan must call the following:

ImgInit	Initializes the ImageMan system. Call on application startup.
ImgShutdown	Cleans up all ImageMan memory. Call on application exit.

The minumum required to display a disk-based image (in order):

ImgOpenSolo	Opens the image file and obtains header info, including the color palette (if applicable).
ImgDrawImage	Draws the image on a given display context.
ImgClose	Closes the image and releases all memory associated with it.

The minimum required to save an image on disk (in order):

Get a handle to a DIB representing the image (could come from ImgGetDIB)

ImgXWriteDIB	Outputs the image, allowing the user to select the desired file format.

Supported Image Formats

ImageMan directly supports the image file formats listed below. For information on writing your own image import/export filters to work with ImageMan, please contact DTI for our ImageMan Technical Reference.

Remember that ImageMan is the only imaging solution you can buy that allows you to support virtually any image format!

Formats in red text support multi-page image reading and writing.

AutoCad DXF (Read-Only)
TIFF*
JPEG/JFIF
PCX
DCX
GIF*
FaxMan Fax Format (FMF)
Windows BMP and DIB
RLE
CompuServe PNG
Kodak Photo CD (Read-Only)
Ventura IMG (Read-Only)
Windows Metafile (WMF) (Read-Only)
Word Perfect Graphic (WPG) (Read-Only)
EPSF (Encapsulated PostScript)
TGA (Targa) (Read-Only)

*Requires a license from Unisys Corp.

For Information on licensing the LZW compression code Contact:

Mark T Starr
Unisys Corporation
PO Box 500
Blue Bell, PA 19424-0001

Voice: 215-986-4411
Fax: 215-986-5721

You must obtain a license from Unisys before we can unlock the TIFF (w/LZW) and GIF readers and writers.

Image Initialization Functions

All of these functions will initialize an image and return an ImageMan image handle to your application. With the ImageMan image handle the image can be rotated, color-reduced, stored to disk, etc...

ImgOpenSolo	Opens an image in a standalone image file (i.e., a TIFF or PCX file).
ImgOpenMem	Open an image which is contained in memory.
ImgOpenEmbedded	Opens an image that is embedded within another file. Useful for working with a database of TIFF images, for instance.
ImgFromClipboard	Creates an ImageMan image from the contents of the Clipboard.
ImgFromDIB	Creates an ImageMan image from a handle to a packed DIB.
ImgFromWMF	Creates an ImageMan image from a Windows Metafile.

Image Export Functions

All of these functions will enable your application to convert a given ImageMan image handle into a readily-transferable image format, either in memory, on the clipboard, or on disk.

Memory based export functions:
ImgXBeginWriteMem / ImgXWriteBlock / ImgXEndWriteMem Writes an image to a memory block.

ImgGetDIB	Returns a packed DIB handle.
ImgGetDDB	Returns a handle to a Windows DDB
ImgGetWMF	Returns a handle to a Windows Metafile (vector images only)
ImgToClipboard	Places an image on the Windows clipboard.

Disk-based export functions.
These functions allow you to export a given image to a disk file in any one of several supported raster images (vector images cannot currently be written to disk). These functions will, in general, require a handle to a packed DIB; this is easily created from an ImageMan image handle via the ImgGetDIB function. These functions also require an Option Block for greatest control over the export process.

ImgXWriteDIB
This is the easiest way to export an image. Simply pass in the DIB you want exported, and this function does the rest. Your application doesn't even need to know which image formats are available for export.

ImgXBeginWrite / ImgXWriteBlock / ImgXEndWrite / ImgXWriteBMPBlock
These functions can be used to export an image in blocks (for instance, an extremely large image that doesn't fit into available memory) or whenever your application needs greater control over the export process. Use ImgXWriteBlock when exporting a Device Independent Bitmap (DIB), ImgXWriteBMPBlock when exporting a Device Dependent Bitmap (DDB).

Option Block Functions
- ImgXOptBlkAdd
- ImgXOptBlkCreate
- ImgXOptBlkDel
- ImgXOptBlkGet

Miscellaneous Image Export support functions
- ImgXAbort
- ImgXErrBox
- ImgXErrString
- ImgXFileDialog
- ImgXGetExt
- ImgXGetStatus

Option Blocks

An Option Block (OptBlk) is a block of memory which contains various image exporting options. Inside the memory block is a collection of strings which looks much like a Windows .INI file. You use the ImgXOptBlk... functions to create, edit, and destroy Option Blocks.

Manipulation Functions

These functions allow you to copy, scale, rotate, adjust gamma and brightness, and dither or color reduce images. They're divided into two groups: those that alter the original image, and those that produce new images which reflect the desired effects.

Functions which alter the original image

ImgBrightness	Adjusts the brightness level of an image.
ImgGamma	Adjusts the gamma point of an image.
ImgInvert	Inverts an image.
ImgMirror	Mirrors an image along the vertical and/or horizontal axis.

Functions which return a new image

ImgCopy	Returns a copy of the original image, scaled to any desired size.
ImgRotate	Returns a rotated copy of the original image.
ImgReduceColors	Returns a color-reduced and/or dithered copy of the original image.
ImgIncreaseColors	Returns a new image with the specified bit depth.

Information Functions

These functions give you information about the ImageMan system or about individual images. An application can also setup a callback function to report loading or saving status.

ImgGetInfo	Retrieves image height, width, color makeup, & various flags.
ImgGetExt	Returns string containing supported import extensions. String can be passed directly to Windows common dialog function.
ImgXGetExt	Returns string containing supported export extensions.
ImgGetPalette	Returns a handle to a Windows logical palette for a given image.
ImgSetStatusProc	Sets up a status function for a given image.
ImgSetDefaultStatusProc	Sets up a default status function.
ImgGetStatus	Returns the status of the most recent ImageMan operation.
ImgErrBox	Produces a message box for the most recent ImageMan error.
ImgErrString	Returns a string containing the text of the most recent ImageMan error.

Multi-Page Functions

These functions allow you to work with multi-page images.

ImgPageCount	Returns the number of pages in a given image.
ImgSetPage	Sets the current page for a given image.
ImgGetPage	Returns the currently active page for an image.

ImageMan How-To

Loading and Displaying an Image

Example

These are the general steps you should follow to open and display an image from a file.

- Obtain the image name you wish to open. For this you could use the ImgGetExt function in conjunction with the Windows Common Dialog functions.
- Call the ImgOpenSolo function to obtain an ImageMan image handle to the image.
- Call the ImgGetPalette function to obtain a logical palette for the image
- Select the palette into the device context using SelectPalette / RealizePalette
- Call ImgDrawImage to draw the image

Example: Loading and Displaying an Image

Close Copy Print

```
/*
This is a function to load and display an image at (x,y) on the given
device context.

Note that this is NOT the best way to work with images. This example merely
illustrates the steps required to open and dispay an image. In general, you would only
open an image once and then use the image handle until you were done with it; then
you would call ImgClose.

You should also note that there is no error handling in this example.
*/

void LoadandDisplay(LPSTR lpszFileName, HDC hDC, int x, int y, int nWidth, int
nHeight)
{
  HANDLE hImage;
  HPALETTE hPal;
  RECT rDest;

  hImage = ImgOpenSolo(lpszFileName, NULL);
  hPal = ImgGetPalette(hImage);
  SelectPalette(hDC, hPal, 0);
  RealizePalette(hDC);

  rDest.left = x;
  rDest.top = y;
  rDest.right = rDest.left + nWidth;
  rDest.bottom = rDest.top + nHeight;

  ImgDrawImage(hImage, hDC, &rDest, NULL);

  ImgClose(hImage);
}
```

ImageMan and the Common Dialogs

Example

To use ImageMan with the Windows GetOpenFileName function, just use the return value from the ImgGetExt function as the lpstrFilter entry in the OPENFILENAME structure passed to the GetOpenFileName function.

ImgGetExt and the Windows Common Dialogs

Close Copy Print

```
HANDLE GetImage(HANDLE hWnd)
{
	OPENFILENAME ofn;
	HANDLE hImage;
	char szFile[256], szFileTitle[256];

	szFile[0] = '\0';
	hImage = NULL;

	ofn.lStructSize = sizeof(OPENFILENAME);
	ofn.hwndOwner = hWnd;
	ofn.lpstrFilter = ImgGetExt();
	ofn.lpstrCustomFilter = (LPSTR) NULL;
	ofn.nMaxCustFilter = 0;
	ofn.nFilterIndex = 1;
	ofn.lpstrFile = szFile;
	ofn.nMaxFile = sizeof(szFile);
	ofn.lpstrFileTitle = szFileTitle;
	ofn.nMaxFileTitle = sizeof(szFileTitle);
	ofn.lpstrInitialDir = lpDefPath;
	ofn.lpstrTitle = "Open Image";
	ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
	ofn.nFileOffset = 0;
	ofn.nFileExtension = 0;
	ofn.lpstrDefExt = NULL;

	if( GetOpenFileName( &ofn ) ) {
		hImage = ImgOpenSolo( ofn.lpstrFile, NULL );
		if( !hImage )		ImgErrBox( hWnd );
	}

	// get rid of ext string
	GlobalFreePtr(ofn.lpstrFilter);
	return hImage;
}
```

Notice that your application is responsible for disposing of the string returned from ImgGetExt.

Scaling an Image

Example

The ImgCopy command makes it easy to scale an image to any size. The new image has the same color makeup as the original.

ImgCopy has been designed to help you conserve memory. Using it you can make a smaller copy of a large image without loading the large image into main memory (see the example for details).

ImgCopy Example

Close Copy Print

```
/*
hImage is an ImageMan handle to a previously opened image.
lpbi is a pointer returned from ImgGetInfo(hImage)
*/

//
// Create a 200x200 thumbnail from hImage
//
hNewImage = ImgCopy(hImage, 200, 200, NULL, COPY_DEL);

//
// Copy the top-left 100x150 pixels from hImage
// into a new 200 x 300 image
//
RECT rSrc;

rSrc.left = rSrc.top = 0;
rSrc.right = 100;
rSrc.bottom = 150;
hNewImage2 = ImgCopy(hImage, 100, 150, &rSrc, COPY_DEL);

//
// Scale the image up x2
//
bNewImage3 = ImgCopy(hImage,
 (int)(lpbi->bmiHeader.biWidth*2),
 (int)(lpbi->bmiHeader.biHeight*2), NULL, COPY_DEL);
```

Color Reducing an Image

Example

ImageMan gives you many options when it comes to color reduction and dithering. All of them make use of the ImgReduceColors function. The ImgReduceColors function allows your application to:

- Reduce a 24-bit image to 256 colors (with or without dithering)
- Reduce a 256-color image to 16 colors (with or without dithering)
- Reduce an image to 256-color grayscale
- Dither a color image to a fixed palette (This allows you to display multiple color images simultaneously)

Color Reduction Example

Close Copy Print

```
.
.
// hImg is a 24-bit image. Note that it is not altered in the
// following operations.
hImg = ImgOpenSolo("colorful.jpg", NULL);

// first we'll dither it to black & white using the Burkes dither
hImg2 = ImgReduceColors(hImg, 2, IMG_BURKES);

// next, we'll reduce it from 24-bit color to 256-colors without
// doing any dithering. To do this, ImageMan must calculate a
// palette based on the colors in the image. This is a
// time-consuming operation, but gives the best results.
hImg3 = ImgReduceColors(hImg, 256, NULL);

// The following command will reduce and dither the image to a
// pre-defined palette. This is much faster than the above code,
// but yields results that are not quite as good. This is also
// useful for displaying more than one color image at a time.
hImg4 = ImgReduceColors(hImg, 256, IMG_FIXEDPALETTE | IMG_FLOYD);

// You can also dither a 256-color image to a pre-defined
// palette. This makes it easy to display multiple color
// images simultaneously without palette problems.

// Here we reduce it to 16-bit color and dither it using
// the Floyd-Steinberg dither
hImg5 = ImgReduceColor(hImg, 16, IMG_FLOYD);

// finally, we'll produce a grayscale version of the image in
// 256 colors
hImg6 = ImgReduceColor(hImg, 256, IMG_GRAYSCALE);
.
.
.
```

Displaying Multiple Color Images

Example

By using the ImgReduceColors function in conjunction with the IMG_FIXEDPALETTE flag, it is an easy matter to display several palette-color images simultaneously. This is most commonly a problem on 256 or 16 color displays; if you're working with a 24-bit display, this isn't a problem.

The basic idea is to dither all images against a common palette. ImageMan has a standard rainbow palette that is used for dithering and color reduction when the IMG_FIXEDPALETTE flag is set. By color-reducing/dithering each image in this way before display, all color images can be displayed with very little loss of image quality.

Displaying multiple color images

```
//
// We're preparing these images for display on
// a 256-color video system, so we reduce to 256 colors.
//
hImage1 = ImgOpenSolo("bird.gif",NULL);   // a 256-color image
hImage2 = ImgOpenSolo("squirrel.gif",NULL);     // a 24-bit image

//
// hImage3 is a dithered version of hImage1
//
hImage3 = ImgReduceColors(hImage1, 256, IMG_FIXEDPALETTE | IMG_FLOYD);

//
// hImage4 is a color-reduced and dithered version of hImage2
//
hImage4 = ImgReduceColors(hImage2, 256, IMG_FIXEDPALETTE | IMG_FLOYD);

//
// We can now draw both images with good results, as long as we
// first select the proper palette into the device context beforehand.
// The proper palette is available from hImage3 or hImage4.
//
// assume that r1 and r2 have been pre-calculated
//
hPalette = ImgGetPalette(hImage3);
SelectPalette(hDC, hPalette, 0);
RealizePalette(hDC);
ImgDrawImage(hImage3, hDC, &r1, NULL);
ImgDrawImage(hImage4, hDC, &r2, NULL);

//
// Everything displayed just beautifully, huh?
//
```

ImageMan 5.04
Online Help
9/20/96

void IMAPI SetDefaultStatusProc(lpfnStat, lCnt, dwUser)
This function sets up a default status procedure for reporting the status of image loading and saving.
Example Changes From Version 2.00

Parameter	Type/Description
lpfnStat	**STATUSPROC** The procedure to call during the load/save operation.
lCnt	**LONG** Interval between calls to status procedure (in bytes).
dwUser	**DWORD** Any user information to be passed on to the status procedure.

Comments
With this procedure you can easily display a thermometer-bar or some other progress indicator for image loading/saving. The dwUser parameter is any information you might want your status procedure to be able to access.

Changes from Prior Versions

New Features

- Support for reading AutoCad DXF vector image files.
- Support for reading/writing CompuServe PNG image files.
- Support for reading Kodak Photo CD image files.
- ImgRotate now supports rotating images in single degree increments
- Scale-to-grey (Antialiasing) for black & white documents (via ImgCopy function)
- Color Reduction/Dithering code is now 10-30% faster
- Support for Cancelling loading & saving images via the Status function
- JPEG writer now supports writing any color depth images (Automatically converts to 24 bit data)
- ImageMan now supports reading and writing images to/from memory using the ImgOpenMem and ImgXBeginWriteMem / ImgXEndWriteMem functions.
- JPEG decompression is now approximately twice as fast.
- Group3 decompression will now ignore errors and attempt to resynch on EOLs. This makes ImageMan much more robust for faxing applications, which can't always guarantee that the data they receive is OK.
- Support for DTI's own fax file format (.FMP and .FMF extensions).
- The ImgCopy function has been improved when doing banded copy of unloaded images. It now seeks to the desired row, which can significantly decrease the time it takes to scale the image.
- TWAIN Scanner support is now integrated into the ImageMan DLLs.
- Misc. bug fixes

Rotating Images

Example

ImageMan allows you to rotate any image in single degree increments.

Miscellaneous Image Manipulations

```
.
.
// hImage has been previously opened
hImage2 = ImgRotate(hImage, 45, COLORREF(0));

ImgBrightness(hImage2, 100);         // brighten the image
ImgMirror(hImage2, TRUE, FALSE);   //Mirror vertically

.
.
```

Using Status Procedures

Example

With the ImgSetDefaultStatusProc and ImgSetStatusProc functions, ImageMan makes it easy for an application to display load/save status information for any image.

The easiest method is to call ImgSetDefaultStatusProc when you initialize ImageMan. This way you can set it up and just forget it.

The ImgSetStatusProc allows you to set a status proc for individual images. This should rarely be needed.

Both of these function utilize a callback function supplied by the calling application. This callback function receives three parameters when it is called. The first is the handle to the ImageMan image being loaded (for image saves, this parameter is 0). The second is an integer representing the percentage of the operation completed (0 - 100). The third is the DWORD passed to ImageMan when the status procedure was setup; this DWORD is used to allow the application to pass any extra information to the status procedure.

Status Callback Procedure

The callback procedure utilized by the ImgSetDefaultStatusProc / ImgSetStatusProc functions is defined below.
Example Changes From Version 2.00

int IMAPI status(hImage, nPercent, dwUserInfo)

Parameter	Type/Description
hImage	**HANDLE** Handle to the ImageMan image being loaded. For image writing, this is 0.
nPercent	**int** Percentage of the load/save operation completed.
dwUserInfo	**DWORD** User information passed to ImgSetDefaultStatusProc / ImgSetStatusProc. This allows the caller to send ancillary information to the status procedure.

Return Value
A return value of 0 will cause the current operation to be aborted.

ImgSetDefaultStatusProc / ImgSetStatusProc

Close Copy Print

```
.

// declare our status procedure
int _export IMAPI status(HANDLE hImg, int nPercent, DWORD dwStatus);
.
.
.
// Initialize ImageMan system and setup default status procedure.
// When we setup the status proc, pass in the handle to the
// status window we're using...
ImgInit();
ImgSetDefaultStatusProc(status, 25000L, (DWORD)hStatusWnd);
.
.
hImage = ImgOpenSolo("test.tif", NULL);   // status procedure isn't
called here
ImgLoad(hImage, NULL);  // status procedure is called now
ImgUnload(hImage, TRUE, TRUE, TRUE);// unload everything
// after following call, status proc will be called much more frequently
during load
ImgSetStatusProc(status, 1000L, (DWORD)hStatusWnd);
ImgLoad(hImage, NULL);  // status displayed again

//
// Here's the actual-factual status procedure. Remember that dwStatus
is, in this
// case, the handle to the status window.
//
int IMAPI status(HANDLE hImg, int nPercent, DWORD dwStatus)
{
 //
 // the status window displays a bar-graph in response to WM_USER
messages.
 //
 if (dwStatus) SendMessage((HWND)LOWORD(dwStatus), WM_USER, nPercent,
0L);
}

.
```

Display/Printing Functions

These functions deal with displaying and printing images.

ImgCreateDDB	Creates a device-dependent bitmap for use when drawing the image.
ImgDrawImage	Draws a given image on a given device context.
ImgSetROP	Sets the opcode to be used when drawing an image.
ImgGetROP	Retrieves the current ROP code used for drawing an image.
ImgLoad	Forces an image to be decoded and loaded into memory.
ImgPrintImage	Draws a given image on a given printer device context.
ImgUnload	Unloads one or more in-memory representations of a given image.

int IMAPI ImgInvert(hImage)
This function inverts a given image.
Example

Parameter	**Type/Description**
hImage	**HANDLE** Identifies the image to be inverted.

Return Value
The return value is IMG_OK on success, an error value on error.

Comments
Although inversion is most commonly performed on monochrome images, it can be applied to color images as well. When used with palette-based images (256 or 16 color images) you must call the ImgGetPalette function after calling the ImgInvert function, as ImgInvert inverts the palette and not the actual image pixels.

HANDLE IMAPI ImgOpenMem(hpMem, dwSize, lpExt)
This function opens the specified in-memory image pointed to by hpMem and returns a handle to be used with the other ImageMan functions.

Parameter	**Type/Description**
hpMem	**HPSTR** Points to a memory buffer containing the actual image data to be read.
dwSize	**DWORD** Size of the image in the memory buffer, in bytes.
lpExt	**LPSTR** Specifies that ImageMan should process the file as an image with the extension specified.

Return Value
The return value specifies the handle used to refer to the image. It is NULL if the image could not be opened.

Comments
The dwSize parameter is the size of the image, not the size of the memory buffer. You must pass the proper image size to this function to be able to read the image.

Following this function call, the normal ImageMan functions (such as ImgLoad and ImgDrawImage) will decode the image from the specified memory buffer when required.

Note
Although the lpExt parameter is not required due to ImageMan's image auto-detection, it is a good idea to fill in the lpExt parameter if you know what type of image you're passing to ImageMan. This way ImageMan can detect the format more quickly.

HANDLE IMAPI ImgXBeginWriteMem(lpszExt, lpDIB, hOptBlk, lpInfo, hMemBlk, dwUsed)
This function is used to begin exporting an image into a memory block. It is logically identical to ImgXBeginWrite, but operates on memory instead of disk.

Parameter	**Type/Description**
lpszExt	**LPSTR** Points to asciiz string containing extension of the file type to use when storing this image.
lpDIB	**LPBITMAPINFO** Points to a BITMAPINFO struct which defines the image.
hOptBlk	**HANDLE** Handle to an option block for this image. If this is NULL, ImageMan/X will assume logical defaults.
lpInfo	**LPXINFO** Points to an XINFO struct defining the export procedure for this image.
hMemBlk	**HANDLE** Handle of the memory block in which to place the exported image data. If NULL, a handle will be allocated by ImageMan.
dwUsed	DWORD Number of bytes currently used in hMemBlk. If hMemBlk is set to NULL, this should be set to 0.

Return Value
The return value is the handle to use when referencing this export job in future ImageMan/X functions; it is NULL if an error occurred.

Comments
This function is the memory-writing analog to the ImgXBeginWrite function, so you should review the ImgXBeginWrite function before for basic information about this function.

To export an image into an existing memory block, simply pass the handle to the memory block and the number of bytes currently used into this function. If you don't have an existing memory block, simply pass NULL for hMemBlk and set dwUsed to 0 and ImageMan will allocate a memory block to be used.

When done writing using this function, your application should call the ImgXEndWriteMem function to ensure that the image is properly exported.

Note
The memory handle passed into ImgXBeginWriteMem may change due to GlobalReAlloc function calls made to resize it!

int IMAPI ImgXEndWriteMem(hJob, lpInfo, HANDLE FAR *phMem, DWORD FAR *pdwLen)

This function ends an export to memory job. It is used in conjunction with ImgXBeginWriteMem.

Parameter	Type/Description
hJob	**HANDLE** Handle indentifying this job (from ImgXBeginWrite).
lpXInfo	**LPXINFO** Points to an XINFO struct which contains results of the operation.
phMem	**HANDLE FAR *** Points to a HANDLE variable to hold the resulting memory handle of the image.
pdwLen	**DWORD FAR *** Points to a DWORD variable to hold the number of bytes written to the memory block.

Return Value
The return value is IMGX_OK on success; otherwise it is an ImageMan/X error code.

Comments
Call this function after calling ImgXBeginWriteMem and ImgXWriteBlock to create an in-memory export file.

Upon exit from this function, the HANDLE pointed to by phMem is filled in with the memory handle of the exported image and the DWORD pointed to by pdwLen is filled in with the number of bytes written to the block.

Note
The memory handle passed into ImgXBeginWriteMem may change due to GlobalReAlloc function calls made to resize it!

We've added the COPY_ANTIALIAS flag for creating an antialiased (or scaled-to-tray) copy of a monochrome image.

ImgCopy has also been enhanced to improve the scaling speed when scaling a large image to a smaller image. As long as the larger image is not in memory, we should be able to make a displayable copy at a reduced size. We've displayed images up to 90MB in size using this method.

Packed DIB Format simply means that the image header and image data are contiguous in a single memory block. For example, for a 256-color DIB, a packed DIB would look like this in memory:

BitmapinfoHeader
256 Palette entries
Image Data (in DIB Format -- see SDK for details)

Reading Images From Memory

Example

Using the ImgOpenMem function, your application can easily load an image that's stored in a memory block instead of a disk file. This can be useful for a number of reasons, but the most frequently cited reason we've seen is that the image is stored in a database as a variable-length field and can't be accessed using the ImgOpenEmbedded function.

As the example code shows, once you've opened an image using ImgOpenMem, it can be treated identically to an image opened using ImgOpenSolo or ImgOpenEmbedded. Not a complicated concept.

Using ImgOpenMem

```
.

// assume that hMem is a memory block containing the image we want
// to open. dwSize is the size of the image within the block.
// We've removed error checking to clarify the code.

HANDLE              hImg;
LPBITMAPINFO        lpInfo;
LONG          lFlags;
RECT          rDraw;

// Leave the extension NULL - ImageMan will autodetect the image format!
hImg = ImgOpenMem(hMem, dwSize, NULL)
lpInfo = ImgGetInfo(hImg, &lFlags);

// draw the image into a 100x100 rectangle
SetRect(&rDraw, 0, 0, 99, 99);
ImgDrawImage(hImg, hDC, &rDraw, NULL);
ImgClose(hImg);

// That's all there is to ImgOpenMem!
.
```

TWAIN Scanning Functions

These functions deal with scanning images from TWAIN compatible scanners, digital cameras and frame grabbers.

ScanSelectSource	Allows the user to select a default TWAIN device.
ScanAcquirePage	Allows the user to scan an image.
ScanIsTWAINAvailable	Determines whether any TWAIN compatible devices are installed.

Low Level Scanning Functions

ScanLowInitTwainJob-	Initializes a scanning job.
ScanLowRegisterApp	Sets application specific information
ScanLowSetScanArea	Specifies the area to scan.
ScanLowSetBrightnessContrast	Sets the Bightness/Contrast parameters.
ScanLowSetResolution	Sets the resolution in DPI.
ScanLowAcquirePages	Acquires 1 or more pages from the TWAIN device.
ScanLowCloseJob	Frees resources allocated to the job.

HANDLE IMAPI ScanAcquirePage(hWnd, wPixType)
This function is used to acquire an image from a TWAIN compatible device.
Example

Parameter	**Type/Description**
hWnd	**HWND** Handle to the window to be used as the parent for the scanner dialog.
wPixType	**UNSIGNED** Specifies the type of data to be returned.

Return Value

Upon succesfull completion of scanning the function returns a HANDLE to a DIB. You can use the ImgFromDIB function to obtain an ImageMan handle to the returned image.

Comments

The defines for the **wPixType** parameter are:

Define	**Value**	**Meaning**
TWAIN_BW	0x01	Black & White
TWAIN_GRAY	0x02	Greyscale
TWAIN_RGB	0x04	24 Bit Color Data
TWAIN_PALETTE	0x08	16/256 Color Data
TWAIN_ANYTYPE	0x00	Return any color type

If you specify a color type via the **wPixType** parameter that is not supported then the scan will fail.

int IMAPI ScanIsTWAINAvailable(void)
This function is used to determine whether TWAIN is installed.
Example
Return Value
This function returns 1 if TWAIN is installed otherwise 0 if TWAIN is not installed.

Applications which allow scanning should   call this function to determine whether any TWAIN devices are installed and disable any scanning related menu options if not.

int IMAPI ScanSelectSource(hWnd)
This function is used to display the TWAIN source selection dialog. The dialog allows the user to select the default TWAIN input device.
Example

Parameter	**Type/Description**
hWnd	**HWND** Handle to the window to be used as the parent for the dialog.

Return Value
This function returns 1 if the dialog was displayed succesfully otherwise it returns 0.

HTWAINJOB IMAPI ScanLowInitTwainJob(hWnd)
This low level function returns a handle to a TWAIN job.
Example

Parameter	Type/Description
hWnd	**HWND** Handle to a Window to be used as the parent for the scanner’s dialog.

Return Value
The return value is a handle to a TWAIN Job handle to be used in subsequent low level scanner calls. If the function fails it will return NULL.

Comments
This function only needs to be called when using the low level scanning functions.

int IMAPI ScanLowRegisterApp(hJob, nMajor, nMinor, nLanguage, nCountry, lpszVersion, lpszMfg, lpszFamily, lpszProduct)

This low level function specifies information about the scanning application.

Example

Parameter	Type/Description
hWnd	**HWND** Handle to a Window to be used as the parent for the scanner's dialog.
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.
Major	**int** Specifies Major portion of the app's version.
Minor	**int** Specifies Minor portion of the app's version.
nLanguage	**int** Specifies language of the app.
nCountry	**int** Specifies the Country of the app.
lpVers	**LPSTR** A string specifying the app's Version #.
lpMfg	**LPSTR** The app's manufacturer.
lpFamily	**LPSTR** The app's family description.
lpProduct	**LPSTR** The app's product name

Return Value

The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Comments

This function only needs to be called when using the low level scanning functions.

int IMAPI ScanLowSetScanArea(hJob, left, top, right, bottom)
This low level function specifies the area of the image to be captured.
Example

Parameter	Type/Description
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.
left	**float** Specifies the left edge of the image area to acquire.
top	**float** Specifies the top edge of the image area to acquire.
right	**float** Specifies the right edge of the image area to acquire.
bottom	**float** Specifies the bottom edge of the image area to acquire.

Return Value
The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Comments
This function only needs to be called when using the low level scanning functions.

int IMAPI ScanLowSetBrightnessContrast(hJob, nBrightness, nContrast)
This low level function specifies the Brightness and Contrast settings to be used when scanning.
Example

Parameter	Type/Description
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.
nBrightness	**WORD** Specifies the brightness setting to be used.
nContrast	**WORD** Specifies the Contrast setting to be used.

Return Value
The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Comments
This function only needs to be called when using the low level scanning functions and must be called prior to calling ScanLowAcquirePages.
The allowable values for the nBrightness and nContrast parameters are -1000 to 1000. A value of zero sets the device to its default setting. A value of 1000 specifies the highest setting while a value of -1000 specifies the lowest setting.

int IMAPI ScanLowSetResolution(hJob, nResolution)

This low level function specifies the resolution in DPI to be used when scanning.

Example

Parameter	**Type/Description**
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.
nResolution	**WORD** Specifies the Resolution in DPI.

Return Value

The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Comments

This function only needs to be called when using the low level scanning functions and must be called prior to calling ScanLowAcquirePages. If the resolution is not specified by calling this function then the device default or user selection will be used.

int IMAPI ScanLowAcquirePage(hJob, nPages, wPixTypes, nFlags, lphDib)
This low level function initiates scanning.
Example

Parameter	Type/Description
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.
nPages	**int** Specifies the number of pages to acquire.
wPixTypes	**WORD** Specifies the color depth of the image data to be returned.
nFlags	**int** Specifies scanning options.
lphDIB	**HANDLE FAR *** Specifies a pointer to handle to DIB. Used when acquiring 1 page.

Return Value
The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Comments
When an image is acquired the function will send a WM_IMSCAN message to the hWnd associated with the job. The wParam parameter of the message contains a Handle to a DIB for the acquired image. If the Scan fails then no WM_IMSCAN message will be generated.

The following options are defined for the xPixTypes parameter:

Define	Value	Description
TWAIN_ANYTYPE	0x00	Any Color format
TWAIN_BW	0x01	Black & White (1 Bit)
TWAIN_GRAY	0x02	Greyscale (4 or 8 Bit)
TWAIN_PALETTE	0x08	Color (4 or 8 Bit)
TWAIN_RGB	0x04	Color (24 Bit)

The following options are defined for the nFlags parameter:

Define	Value	Description
TWAIN_SHOWUI	0x02	Display Scanner U/I Dialog
TWAIN_USEADF	0x01	Use ADF (If available)

int IMAPI ScanLowAcquirePage(hJob)
This low level functionis called to free the resources allocated by a call to the ScanLowInitTwainJob function.
Example

Parameter	**Type/Description**
hJob	**HTWAINJOB** Handle to a TWAIN job returned from ScanLowInitTwainJob.

Return Value
The return value is SCAN_OK if the function completed successfully. If the function fails it will return SCAN_FAILED.

Low Level Scanning

Close Copy Print

```
#include "imtwain.h"

HTWAINJOB hJob;

hJob = ScanLowInitTwainJob( hWnd );

if( hJob ) {

      // Scan 2 B/W pages using the ADF, without the U/I

      ScanLowAcquirePages( hJob, 2, TWAIN_BW, TWAIN_USEADF, NULL );

      ScanLowCloseJob( hJob );
}

...


// Code in Window Handler for hWnd

HANDLE hImage;

CASE WM_IMSCAN:
      //
      // wp is the wParam of the Window procedure
      //
      if( wp ) {
            hImage = ImgFromDIB( (HANDLE)wp );
            // hImage is now a legitimate ImageMan Image Handle
      }
```

Scanning Sample

```
#include "imgman.h"

HDIB hDib;
HANDLE hImage;

if( ScanIsTwainAvailable() ) {            // Make sure TWAIN is installed

      hDIB = ScanAcquirePage( hWnd, TWAIN_ANYTYPE );

      if( hDIB ) {
            hImage = ImgFromDIB( hDIB );
            // Using hImage you can call any other ImageMan functions
      }
}
```

Ordering DTI Products

You may order our products directly from Data Techniques or through one of our many distributors in the US and abroad.

To order direct

Phone:	800-955-8015 704-682-4111
Fax:	704-682-0025
Email:	custsvc@data-tech.com
Web:	www.data-tech.com

Product	Retail
ImageMan/VBX (Includes 16 bit Image and Twain VBX Controls)	$295
ImageMan ActiveX Suite (Includes 16 & 32 bit Active X (OLE) Controls and 16 bit VBX Controls)	$495
ImageMan DLL Suite (Includes 16 & 32 bit DLLs)	$795
ImageMan Source Code (Includes complete source code for VBX, Active X and DLLs)	$1895
FaxMan (Programmable Fax engine for Windows developers)	$495

We ship via Federal Express and can also ship via email or ftp.
For overnight shipping in the US add $14, for 2 day shipping add $7.50. North Carolina residents add 6% sales tax.

For shipping outside the US please call or email.

Ordering Via a Distributor

For a current list of distributors please visit our website at: *www.data-tech.com* or call us.