# ImageMan 2.0 Help Reference

ImageMan 2.0 API

API Quick Reference

ImageMan How-To Section

Changes from ImageMan Version 1.06

Supported File Formats

How to Contact DTI

How to Order ImageMan

ImageMan Release Notes

# ImageMan Technical Support

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

Data Techniques, Inc.
340 Bowditch Street #6
Burnsville, NC

| | | |
|---|---|---|
| **Tech Support**: | 704-682-4111 | 9am to 5pm EST |
| **Fax**: | 704-682-0025 | |
| **BBS**: | 704-682-4356 | (2400-14.4K V.42, 24 Hours/Day) |
| **CompuServe:** | GO DATATECH | |
| **Cserve ID**: | 74431,1412 | |

# ImageMan Ordering Information

**<u>Pricing</u>**
ImageMan DLL          $495
ImageMan/VB           $295
ImageMan for Windows NT $895

ImageMan Src Code              $1495

We accept Visa, MasterCard, American Express, COD or prepayment. All orders include free ground shipping via UPS and Next Day Air shipping is available for an additional charge.

All DTI products are backed by our 90 day full money back guarantee.

## To order call:

**<span style="color:red">800-955-8015</span>**
**<span style="color:red">704-682-4111</span>**

**<u>Or mail orders to:</u>**

Data Techniques, Inc.
340 Bowditch Street, Ste 6
Burnsville, NC 28714

## int FAR PASCAL 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.

## int FAR PASCAL ImgClose( hImage )

This function closes the referenced image.
Example

**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 FAR PASCAL 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.

<u>Example</u>

| 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). |

**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.

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 FAR PASCAL 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.
Example

| 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.
Example

| 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 FAR PASCAL ImgErrBox( hParent )

This function displays a Windows messagebox containing a textual description of the last ImageMan error.
Example

| 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 FAR PASCAL ImgErrString( lpszBuf, nMaxLen )

This function copies the text describing the last error into a user supplied buffer.
Example

| 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 FAR PASCAL ImgFromClipboard()

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

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 FAR PASCAL ImgFromDIB(hDIB)

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

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

| 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 FAR PASCAL ImgGamma(hImage, nGamma)

This function performs gamma correction on the given hImage.
Example

| 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 FAR PASCAL ImgGetDDB(hImage, hDC)

This function returns a handle to a Windows device-dependent bitmap (DDB) which represents the given image.
Example

| 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 FAR PASCAL ImgGetDIB(hImage, bNewDib, lpSrcRect)
This function retrieves a handle to hImage in packed DIB format.
Example

| 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 FAR PASCAL ImgGetExt()

This function returns a string containing the image formats currently supported by ImageMan.
Example

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 FAR PASCAL ImgGetInfo(hImage, lpFlags)

This functions returns a pointer to an BITMAPINFO struct which defines the image.
Example

| 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 IMGMAN2.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 FAR PASCAL ImgGetPage(hImage, lpPage)

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

Example

| 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 FAR PASCAL ImgGetPalette(hImage)

This function returns a handle to a GDI logical palette which describes the color content of an image.
Example

**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 FAR PASCAL ImgGetROP( hImage)

This function returns the current ROP code for the given image.
Example

This function takes no parameters.

**Return Value**
The return value is the current ROP code

## int FAR PASCAL ImgGetStatus()

This function returns the current status of ImageMan.

Example

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 FAR PASCAL ImgGetWMF(hImage, lpRect)

This functions returns a handle to a Windows Metafile that represents hImage. This function is only valid for vector images.
Example

| Parameter | Type/Description |
|---|---|
| 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 FAR PASCAL ImgInit()

This function initializes the ImageMan library, and must be called before any other ImageMan calls are made.
Example

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 FAR PASCAL 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.
Example

**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 FAR PASCAL ImgMirror( hImage, bVert, bHoriz )

This function mirrors the referenced image along the specified axis.
Example

| 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.
Example

| 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. |

**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. 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.
Example

| Parameter | Type/Description |
|-----------|------------------|
| 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 FAR PASCAL ImgPageCount(hImage, lpCnt)

This function returns the number of pages that are in the image file into the variable pointed to by lpCnt.
Example

**Parameter**              **Type/Description**
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.
Example

| 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 FAR PASCAL ImgReduceColors( hImage, nColors, nFlags)
This function returns a new image with the specifed number of colors.
Example

| 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 FAR PASCAL ImgRotate(hImage, nDegree, rgbCol)

This function rotates the referenced image in the counterclockwise direction and returns a handle to the new image.
Example

| 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.

### Comments

Currently, images may only be rotated by multiples of 90 degrees. Future releases will support abitrary rotations.

### Note

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

## int FAR PASCAL ImgSetPage(hImage, nPage)

This function prepares the given image to read from image number nPage in the file.
Example

| 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 FAR PASCAL ImgSetROP(hImage, dwNewROP)
This function sets the Windows ROP code to be used when displaying and printing the given image.
Example

| 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 FAR PASCAL 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.
<u>Example</u>

| **Parameter** | **Type/Description** |
|---|---|
| hImage | **HANDLE** Specifies the image to register the status function for. |
| lpStatProc | **<u>STATUSPROC</u>** 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.

## void FAR PASCAL 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.
Example

This function takes no parameters.

## int FAR PASCAL ImgToClipboard(hImage)

This function places the given ImageMan image on the clipboard in CF_DIB or CF_METAFILEPICT format.
Example

| 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 FAR PASCAL 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.
Example

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

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

| 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 ImgXOptBlkAdd(hOptBlk, lpszKey, lpszValue)
This function adds a key/value pair to an <u>Option Block</u> (OptBlk).
Example

| 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 <u>option block</u> 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 <u>option block</u> string must be specified exactly as shown; i.e., there must be a space to either side of the '='.

## HANDLE ImgXOptBlkCreate(lpszInit)

This function creates an Option Block (OptBlk) and optionally initializes it.
Example

**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 ImgXOptBlkDel(hOptBlk)

This function deletes an Option Block (OptBlk) from memory.

Example

| 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 ImgXOptBlkGet(hOptBlk, lpszKey, lpszBuf)

This function is used to retrieve the value for a particular key from an <u>Option Block</u> (OptBlk).
Example

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

| 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 <u>option block</u> 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 ImgXWriteBlock(hJob, nLines, lpBits, lpInfo)
This function writes a block of bits to the export file specified by hJob.
Example

| 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 ImgXWriteBMPBlock(hJob, hDC, hBmp, nLines, lpInfo)

This function writes a bitmap to the export file specified by hJob.

Example

| Parameter | Type/Description |
|-----------|------------------|
| 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 ImgXEndWrite(hJob, lpXInfo)

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

| 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 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.

Example

| 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   ImgXErrString(lpBuf)

This function places a string containing the last ImageMan/X error into lpBuf.
Example

**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   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.
Example

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

**Return Value**
Returns IMGX_OK.

## HANDLE   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.
Example

This function takes no arguments.

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

## int ImgXAbort(hImage)

This function instructs ImageMan/X to delete the current export file when the ImgXEndWrite function is called to end writing the file.
Example

| **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 2.0 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 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.

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.

# Required Functions

All programs using ImageMan must call the following:
<u>ImgInit</u>                               Initializes the ImageMan system. Call on application startup.
<u>ImgShutdown</u>                    Cleans up all ImageMan memory. Call on application exit.


The minumum required to display a disk-based image (in order):
<u>ImgOpenSolo</u>   Opens the image file and obtains header info, including the color palette (if applicable).
<u>ImgDrawImage</u>  Draws the image on a given display context.
<u>ImgClose</u>       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 <u>ImgGetDIB</u>)
<u>ImgXWriteDIB</u>   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 libraries to work with ImageMan, contact DTI.

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.

**TIFF Revision 5.0**
JPEG/JFIF
PCX
**DCX**
GIF
Windows BMP and DIB
Ventura IMG (IMG)
Windows Metafile (WMF)
Word Perfect Graphic (WPG)
EPSF (Encapsulated PostScript)
TGA (Targa)

# 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). |
| 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:**
    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. |

# 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**

```
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

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

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

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**  Close  Copy  Print

```
//
// 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?
//
```

# Release Notes

If you have anything you'd like to see in this help file (is there something that caused you particular trouble?) please call us and tell us what it is. Help files are pretty easy to edit, after all.

## void FAR PASCAL SetDefaultStatusProc(lpfnStat, lCnt, dwUser)

This function sets up a default status procedure for reporting the status of image loading and saving.

<u>Example</u>

| 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 Version 1.06

### New Features
- Version 2.0 is significantly faster than version 1.06, especially for Group3/Group4 images.
- Support for JPEG (.JPG), Word Perfect Graphic (.WPG) files, both raster and vector (raster images are returned as a DIB, vector images as WMF), Gem IMG format.
- Export support for JPEG (with control over image quality), GIF, and TARGA images.
- Export support for TIFF Group4 images.
- Multi-page image support. Read and write multi-page TIFF and DCX files. (See Multi-Page Functions)
- Supports callback function for reporting load/save status. (See ImgSetDefaultStatusProc, ImgSetStatusProc)
- High-Speed color reduction/dithering (See ImgReduceColors)
- Fast image rotation (See ImgRotate)
- TWAIN Scanner Support
- Direct support for Windows Common Dialog file open function (See ImgGetExt)
- Clipboard Support (See ImgToClipboard / ImgFromClipboard)


### The Biggest Change
- The biggest change from Version 1.06 is how an image is defined. Previous ImageMan versions defined an image in relation to a file on disk. This approach precluded working with memory-based images, including the clipboard, and also made it somewhat more difficult to define image transformations (rotations, color reduction, etc.) in terms of an "ImageMan image handle."

- Version 2.0 defines an image as the DIB or WMF attached to a given image handle. This enables ImageMan to do all the things it did in Version 1.06, plus work with in-memory images passed in from applications and from the clipboard. It also means that image transformations on an ImageMan image handle can return another ImageMan image handle, making the API much more orthogonal.

### Other significant changes
- ImageMan/X has been incorporated into ImageMan, allowing image export as well as image import. The old ImageMan/X function calls should work unaltered, but you no longer need to link with the IMGX.LIB file, nor do you need to include IMGX.DLL with your application. (See Image Export Functions)

- You must include calls to ImgInit and ImgShutdown.

- The whole concept of tolerance (via ImgSetTolerance function) is no longer valid. Since the image definition has changed, tolerance no longer makes sense. The ImgSetTolerance function is no longer available.

- The ImgDrawImage function no longer creates a scaled-to-size DDB. To create a scaled DDB of an image, you should call the ImgCopy function to create a DIB of the desired size, then call ImgCreateDDB on the scaled image. ImgDrawImage will still scale an image to fit a desired rectangle, but it now relies on the StretchDIBits function, which is notoriously slow (it's much, much faster to call ImgCopy to scale an image). These changes give an application much more control over memory allocation.

# Rotating Images

ImageMan allows you to rotate any image in 90-degree increments.

## Miscellaneous Image Manipulations

```
.
.
// hImage has been previously opened
hImage2 = ImgRotate(hImage, 180, COLORREF(0));

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


.
.
```

## Using Status Procedures

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


**int FAR PASCAL 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**

The return value is currently insignificant. In future versions, a non-zero return value will cause the load/save operation to be aborted.

## ImgSetDefaultStatusProc / ImgSetStatusProc

```
.

// declare our status procedure
int _export FAR PASCAL 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 FAR PASCAL 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 FAR PASCAL 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.

# ImageMan 2.0 Demo Notes

**Please Remember: This demo is still in Beta. If something doesn't work, that doesn't mean that ImageMan doesn't work - it may just be a bug in the demo program itself. Also, the way the demo uses ImageMan is specific to the purposes of the demo (i.e., to show off ImageMan's features in a general way); ImageMan can be tailored in many ways, so the results you get for your specific application may be significantly faster.**

This demo was created using Microsoft's Visual C++ and the Microsoft Foundation Classes. If you know what you're doing with MFC and ImageMan, you could put this demo together from scratch in a day or two.

The general-release version of the demo has been linked directly with the ImageMan object files for security purposes. Any products you distribute will utilize the IMGMAN2.DLL dynamic link library.

Additional Notes:

- For the demo, all color images >= 256 colors are dithered to a common palette. This makes it slightly slower to display the image, but reduces the flickering due to palette management when switching between images.

- The default dither method is None (i.e., no dithering is performed). If you wish to alter the dither method, you must select it from the menu before dithering an image. Without dithering, color images reduced to 2-colors will not be very attractive.

- When displaying the thumbnail window, the source image can be scrolled by dragging the rectangle visible on the thumbnail window. Just click in the rectangle and drag.

- Monochrome thumbnail images are produced by preserving black pixels. For inverted images this will not produce good results.

- The help file associated with the demo (IM2.HLP) must be in the same directory as the demo or on the path for the help to be accessible.

- We have noticed problems with video drivers when performing certain operations (most notably when scrolling an image). If you see artifacts when using the demo it is most likely due to the video driver. These problems usually manifest themselves when working with DIBs, so a possible solution is to convert images to DDBs (which is easy with ImageMan). If you've got any questions about this issue, please call us and ask about it.
-