"EZTW32.DLL"

TWAIN_Testing123(pz cPtr, n cLong, h cLong, d cDouble, u cLong) cLong

Displays a dialog box with the parameter values in it. Use this to test that you can call EZTWAIN and pass parameters correctly.
It returns the value of the HANDLE h parameter.

--------- Top-Level Calls

TWAIN_Acquire(hwndApp cLong) cLong

The minimal use of EZTWAIN.DLL is to call this function with a 0 argument.

Acquires a single image, from the currently selected Data Source, using Memory Transfer Mode. Only one image can be acquired per call.
The return value is a handle to global memory containing a DIB - a Device-Independent Bitmap. Numerous EZTWAIN DIB_xxx functions can be used to examine, modify, and save these DIB images. Remember to use TWAIN_FreeNative to free each DIB when you are done with it!

By default this function shuts down TWAIN before returning, but you can use TWAIN_SetMultiTransfer to change this behavior.

By default, the default data source (DS) is opened, displays its dialog, and determines all the parameters of the acquisition and transfer.
If you want to (try to) hide the DS dialog, see TWAIN_SetHideUI.
To set acquisition parameters, you need to do something like this:

```
    TWAIN_OpenDefaultSource() -or- TWAIN_OpenSource(sourceName)
    TWAIN_Set*        - one or more capability-setting functions
    hdib = TWAIN_Acquire(hwnd)
    if (hdib) { ... process image, TWAIN_FreeNative(hdib)  }
```

TWAIN_FreeNative(hDib cLong)

Release the memory allocated to a DIB, as returned by TWAIN_AcquireNative. (If you are coding in C or C++, this is just a call to GlobalFree.)
If you use TWAIN_AcquireNative and don t free the returned image handle, it stays around taking up Windows (virtual) memory until your application terminates.

TWAIN_SelectImageSource(hWnd cLong) cLong

This is the routine to call when the user chooses the "Select Source..." menu command from your application s File menu.
In the standard implementation of "Select Source...", your application doesn t need to do anything except make this one call.
Note: If only one TWAIN device is installed on a system, it is selected automatically, so there is no need for the user to do Select Source.
You should not require your users to do Select Source before Acquire.

This function posts the Source Manager s Select Source dialog box.
It returns after the user either OK s or CANCEL s that dialog.
A return of 1 indicates OK, 0 indicates one of the following:

a) The user cancelled the dialog

b) The Source Manager found no data sources installed
c) There was a failure before the Select Source dialog could be posted

If you want to be meticulous, disable your "Acquire" and "Select Source" menu items or buttons if TWAIN_IsAvailable() returns 0 - see below.

TWAIN_AcquireNative(hwndApp cLong, wPixTypes cLong) cLong

The minimal use of EZTWAIN.DLL is to just call this routine, with 0 for both parameters.

Acquires a single image, from the currently selected Data Source, using Native-mode transfer. It waits until the source closes (if or forces the source closed if not. The return value is a handle to the acquired image.Only one image can be acquired per call.

The return value is a handle to global memory containing a DIB - a Device-Independent Bitmap. Numerous EZTWAIN functions can be used to examine, modify, and save these DIBs. Remember to use TWAIN_FreeNative to free each DIB when you are done with it, these things eat up a lot of memory.

For the 2nd parameter, you can OR or add together the following masks to indicate what kind(s) of image you prefer to receive.
Caution: Some TWAIN devices will ignore your preference here.
-- If you care, check the parameters of the DIB.

```
Global Const TWAIN_BW = &H1        1-bit per pixel, B&W
Global Const TWAIN_GRAY = &H2      1,4, or 8-bit grayscale
Global Const TWAIN_RGB = &H4       24-bit RGB color
Global Const TWAIN_PALETTE = &H8   1,4, or 8-bit palette
Global Const TWAIN_ANYTYPE = &H0   any of the above (use by itself)
```

TWAIN_AcquireToClipboard(hwndApp cLong, wPixTypes cLong) cLong

Like AcquireNative, but puts the resulting image, if any, into the system clipboard as a CF_DIB item. If this call fails, the clipboard is either empty or retains its previous content.
Returns TRUE (1) for success, FALSE (0) for failure.

Useful for environments like Visual Basic where it is hard to make direct use of a DIB handle. In fact, TWAIN_AcquireToClipboard uses TWAIN_AcquireNative for all the hard work.

TWAIN_AcquireToFilename(hwndApp cLong, sFile cPtr) cLong

Acquire an image and write it to a file using the current Save Format, (See TWAIN_SetSaveFormat) which by default is BMP.
If pszFile is NULL or an empty string, the user is prompted for the file name *and format* in a standard Save File dialog,

Return values:
 0 success.
-1 the Acquire failed.
-2 file open error (invalid path or name, or access denied)
-3 invalid DIB, or image incompatible with file format, or...
-4 writing failed, possibly output device is full.
-10 user cancelled File Save dialog

TWAIN_AcquireFile(hwndApp cLong, nFF cLong, sFile cPtr) cLong

Acquire an image directly to a file, using the given format and filepath.

- Unlike AcquireToFilename, this function uses TWAIN File Transfer Mode.
- The image is written to disk by the Data Source, not by EZTWAIN.
- Warning: This mode is -not- supported by all TWAIN devices.

Use TWAIN_SupportsFileXfer to see if the open DS supports File Transfer.

You can use TWAIN_Get(ICAP_IMAGEFILEFORMAT) to get an enumeration of the available file formats, and CONTAINER_ContainsValue to check for a particular format you are interested in.

nFF can be any file format supported by the DS, see the TWFF_* constants in twain.h for the list of allowed formats. A compliant DS should at least support TWFF_BMP, but as usual there are no guarantees.

If pszFile is NULL or an empty string, the user is prompted for the file name in a standard Save File dialog.

Return values (N.B. A Boolean, not an error code like AcquireToFilename!)
 TRUE(1) for success
 FALSE(0) for failure - use GetResultCode/GetConditionCode for details.
 If the user cancels the Save File dialog, the result code is TWRC_CANCEL

--------- Basic TWAIN Inquiries

TWAIN_IsAvailable() cLong

Call this function any time to find out if TWAIN is installed on the system. It takes a little time on the first call, after that it s fast, just testing a flag. It returns 1 if the TWAIN Source Manager is installed & can be loaded, 0 otherwise.

TWAIN_EasyVersion() cLong

Returns the version number of EZTWAIN, multiplied by 100.
E.g. version 2.01 will return 201 from this call.

TWAIN_GetBuildName(Name cPtr)

Returns the build name of EZTWAIN - usually either Debug or Release .
Remember to pre-allocate Name with at least 256 spaces: Name = Space$(256)
and to trim afterward: Name = Left(Name, InStr(Name, vbNullChar) - 1)

TWAIN_State() cLong

Returns the TWAIN Protocol State per the spec.

```
Global Const TWAIN_PRESESSION = 1           source manager not loaded
Global Const TWAIN_SM_LOADED = 2            source manager loaded
Global Const TWAIN_SM_OPEN = 3              source manager open
Global Const TWAIN_SOURCE_OPEN = 4          source open but not enabled
Global Const TWAIN_SOURCE_ENABLED = 5       source enabled to acquire
Global Const TWAIN_TRANSFER_READY = 6       image ready to transfer
Global Const TWAIN_TRANSFERRING = 7         image in transit
```

TWAIN_GetSourceName(Name cPtr)

Copies current/last opened data source name into Name.
Remember to pre-allocate Name with at least 256 spaces: Name = Space$(256)
and to trim afterward: Name = Left(Name, InStr(Name, vbNullChar) - 1)

--------- DIB handling utilities ---------

DIB_SetResolution(hDib cLong, xdpi cDouble, ydpi cDouble)

Sets the horizontal and vertical resolution of the DIB.

DIB_XResolution(hDib cLong) cDouble

Horizontal (x) resolution of DIB, in DPI (dots per inch)

DIB_YResolution(hDib cLong) cDouble

Vertical (y) resolution of DIB, in DPI (dots per inch)

DIB_Depth(hDib cLong) cLong

Depth of DIB, in bits i.e. bits per pixel.

DIB_Width(hDib cLong) cLong

Width of DIB, in pixels (columns)

DIB_Height(hDib cLong) cLong

Height of DIB, in lines (rows)

DIB_RowBytes(hDib cLong) cLong

Number of bytes needed to store one row of the DIB.

DIB_ColorCount(hDib cLong) cLong

Number of colors in color table of DIB

*** TODO ***
BITMAPINFOHEADER*

DIB_Lock(HANDLE hdib)

Lock the given DIB handle and return a pointer to the header structure.
Technically, increments the lock count of hdib and returns its address.

```
DIB_Unlock(hDib cLong)
```

Unlock the given DIB handle (decrement the lock count.)
When the number of Unlocks = the number of Locks, any pointers
into the DIB should be presumed invalid.

```
DIB_ReadRow(hDib cLong, nRow cLong, prow As Any)

DIB_ReadRowRGB(hDib cLong, nRow cLong, prow As Any)
```

Read row n of the given DIB into buffer at prow.
Caller is responsible for ensuring buffer is large enough.
Row 0 is the *top* row of the image, as it would be displayed.
The first variant reads the data as-is from the DIB, including
BGR pixels from 24-bit DIBs, or packed 1-bit, 4-bit or 8-bit.
The RGB variant unpacks every DIB pixel into 3-byte RGB pixels.

```
DIB_WriteRow(hDib cLong, r cLong, pdata As Any)
```

Write data from buffer into row r of the given DIB.
Caller is responsible for ensuring buffer and row exist, etc.

```
DIB_WriteRowChannel(hDib cLong, r cLong, pdata As Any, nChannel cLong)
```

Write data from buffer into one color channel of row r of the given DIB.
Channels are: 0=Red, 1=Green, 2=Blue. If the DIB depth < 24, acts like
WriteRow.

```
DIB_Allocate(nDepth cLong, nWidth cLong, nHeight cLong) cLong
```

Create a DIB with the given dimensions. Resolution is set to 0. A default
color table is provided if depth <= 8.
The image data is uninitialized i.e. garbage.

```
DIB_Free(hDib cLong)
```

Release the storage of the DIB.

```
DIB_SetGrayColorTable(hDib cLong)
```

Fill the DIB s color table with a gray ramp - so color 0 is black, and
the last color (largest pixel value) is white. No effect if depth > 8.

```
DIB_SetColorTableRGB(hDib cLong, i cLong, r cLong, G cLong, b cLong)
```

Set the ith entry in the DIB s color table to the specified color.
R G and B range from 0 to 255.

```
DIB_CreatePalette(hDib cLong) cLong
```

```
  Create and return a logical palette to be used for drawing the DIB.
  For 1, 4, and 8-bit DIBs the palette contains the DIB color table.
  For 24-bit DIBs, a default halftone palette is returned.
```

```
DIB_DrawToDC(hDib cLong, hDC cLong, dx cLong, dy cLong, w cLong, h cLong, sx cLong, sy cLong)
```

```
  Draws DIB on a device context.
  You should call CreateDibPalette, select that palette
  into the DC, and do a RealizePalette(hDC) first.
```

```
DIB_WriteToBmp(hDib cLong, pszFile cPtr) cLong

DIB_WriteToJpeg(hDib cLong, pszFile cPtr) cLong
```

```
DIB_LoadFromFilename(pszFile cPtr) cLong
```

--- OBSOLETE: The following functions are for backward compatibility only:

```
TWAIN_DibDepth(hDib cLong) cLong

TWAIN_DibWidth(hDib cLong) cLong

TWAIN_DibHeight(hDib cLong) cLong

TWAIN_DibNumColors(hDib cLong) cLong

TWAIN_DibRowBytes(hDib cLong) cLong
```

```
TWAIN_ReadRow(hDib cLong, nRow cLong, prow As Any)
```

```
TWAIN_CreateDibPalette(hDib cLong) cLong

TWAIN_DrawDibToDC(hDC cLong, nDestX cLong, nDestX cLong, nWidth cLong, nHeight cLong, hDib cLong, nSrcX cLong, nSrcX
cLong)
```

--------- File Read/Write

---- File Formats

```
Global Const TWFF_BMP = 2
Global Const TWFF_JFIF = 4
```

```
TWAIN_IsJpegAvailable() cLong
```

```
  Return TRUE (1) if JPEG/JFIF image files can be read and written.
  Returns 0 if JPEG support has not been installed.
```

TWAIN_SetSaveFormat(nFF cLong) cLong

Select the file format for subsequent calls to WriteNativeToFilename
Displays a warning message if the format is not available.
Returns TRUE (1) if ok, FALSE (0) if format is invalid or not available.

nFF	type	ext	notes
2	BMP	.bmp	uncompressed by default
4	JFIF	.jpg	JPEG File Interchange Format 1.02

TWAIN_GetSaveFormat() cLong

Return the currently selected save format.

TWAIN_SetJpegQuality(nQ cLong)

TWAIN_GetJpegQuality() cLong

Set the quality of subsequently saved JPEG/JFIF image files.
nQ = 100 is maximum quality & minimum compression.
nQ = 75 is good quality, the default.
nQ = 1 is minimum quality & maximum compression.

TWAIN_WriteNativeToFilename(hDib cLong, sFile cPtr) cLong

Writes a DIB handle to a file using the current save format.

hdib = DIB handle, as returned by TWAIN_AcquireNative
pszFile = far pointer to NUL-terminated filename

If pszFile is NULL or points to a null string, the user is prompted for the filename *and format* with a standard Windows file-save dialog.

Return values:

0	success
-1	user cancelled File Save dialog
-2	file open error (invalid path or name, or access denied)
-3	image is invalid, or cannot be written in this format.
-4	writing data failed, possibly output device is full

TWAIN_LoadNativeFromFilename(pszFile cPtr) cLong

Load a .BMP file (or JPEG) and return a DIB handle.
Accepts a filename (including path & extension).
If pszFile is NULL or points to a null string, the user is prompted to choose a file with a standard File Open dialog.
Returns a DIB handle if successful, otherwise NULL.

--------- Global Options

```
TWAIN_SetMultiTransfer(fYes cLong)

TWAIN_GetMultiTransfer() cLong
```

This function controls the multiple transfers flag.
If this flag is non-zero: After an Acquire, the Data Source is not closed, but is left open to allow additional images to be acquired in the same session. The programmer may need to close the Data Source after all images have been transferred, for example by calling

    TWAIN_CloseSource or
    TWAIN_UnloadSourceManager

In multi-transfer mode, you can check for the availability of more images by calling TWAIN_State. A state of 6 (TWAIN_TRANSFER_READY) means that the Data Source knows it has at least one more image.
A state of 5 (TWAIN_SOURCE_ENABLED) means that the user has not closed the Data Source, so it might be able to acquire another image.
If you try to acquire and the Data Source can t do it, the acquire function will return an error indication.

By default, this feature is set to FALSE (0).

```
TWAIN_SetHideUI(nHide cLong)

TWAIN_GetHideUI() cLong
```

These functions control the hide source user interface flag.
This flag is cleared initially, but if you set it non-zero, then when a source is enabled it will be asked to hide its user interface.
Note that this is only a request - some sources will ignore it!
This affects AcquireNative, AcquireToClipboard, and EnableSource.
If the user interface is hidden, you will probably want to set at least some of the basic acquisition parameters yourself - see SetCurrentUnits, SetCurrentPixelType, SetBitDepth and SetCurrentResolution below.
See also: HasControllableUI

--------- Application Registration

```
TWAIN_RegisterApp(nMajorNum cLong, nMinorNum cLong, nLanguage cLong, nCountry cLong, sVersion cPtr, sMfg cPtr, sFamily cPtr, sProduct cPtr)
```

TWAIN_RegisterApp can be called *AS THE FIRST CALL*, to register the application. If this function is not called, the application is given a generic registration by EZTWAIN.
Registration only provides this information to the Source Manager and any sources you may open - it is used for debugging, and (frankly) by some sources to give special treatment to certain applications.

--------- Error Analysis and Reporting -----------------------------------

```
TWAIN_GetResultCode() cLong
```

Return the result code (TWRC_xxx) from the last triplet sent to TWAIN

TWAIN_GetConditionCode() cLong

```
Return the condition code from the last triplet sent to TWAIN.
(To be precise, from the last call to TWAIN_DS below)
If a source is NOT open, return the condition code of the source manager.
```

TWAIN_ErrorBox(sMsg cPtr)

```
Post an error message dialog with an exclamation mark, OK button,
and the title  TWAIN Error .
pszMsg points to a null-terminated message string.
```

TWAIN_ReportLastError(sMsg cPtr)

```
Like TWAIN_ErrorBox, but if some details are available from
TWAIN about the last failure, they are included in the message box.
```

--------- TWAIN State Control ------------------------------------

TWAIN_LoadSourceManager() cLong

```
Finds and loads the Data Source Manager, TWAIN.DLL.
If Source Manager is already loaded, does nothing and returns TRUE.
This can fail if TWAIN.DLL is not installed (in the right place), or
if the library cannot load for some reason (insufficient memory?) or
if TWAIN.DLL has been corrupted.
```

TWAIN_OpenSourceManager(hWnd cLong) cLong

```
Opens the Data Source Manager, if not already open.
If the Source Manager is already open, does nothing and returns TRUE.
This call will fail if the Source Manager is not loaded.
```

TWAIN_OpenDefaultSource() cLong

```
This opens the TWAIN default source i.e. the last source selected
in the TWAIN Select Source dialog, in any application.
If a source is already open, does nothing and returns TRUE.
Fails returning FALSE (0) if: the default source fails to open,
if a source is currently open and enabled, or if the TWAIN
Source Manager is not open and cannot be opened.
```

TWAIN_GetSourceList() cLong

```
Fetches the list of sources into memory, so they can be returned
one by one by TWAIN_GetNextSourceName, below.
TRUE (1) if successful, FALSE (0) otherwise.
```

TWAIN_GetNextSourceName(Name cPtr) cLong

```
Copies the next source name in the list into Name
Remember to pre-allocate Name with at least 256 spaces: Name = Space$(256)
and to trim afterward: Name = Left(Name, InStr(Name, vbNullChar) - 1)
Returns TRUE (1) if successful, FALSE (0) if there is no next name to return.
```

TWAIN_GetDefaultSourceName(Name cPtr) cLong

```
Copies the name of the TWAIN default source into Name.
Remember to pre-allocate Name with at least 256 spaces: Name = Space$(256)
and to trim afterward: Name = Left(Name, InStr(Name, vbNullChar) - 1)
Normally returns TRUE (1) but will return FALSE (0) if:
  - the TWAIN Source Manager cannot be loaded & initialized or
  - there is no current default source (e.g. no sources are installed)
```

TWAIN_OpenSource(Name cPtr) cLong

```
Opens the source with the given name.
If that source is already open, does nothing and returns TRUE
If another source is open, closes it and attempts to open the specified source
Will load and open the Source Manager if needed.
```

TWAIN_EnableSource(hWnd cLong) cLong

```
Enables the open Data Source. This posts the source s user interface
and allows image acquisition to begin.  If the source is already enabled,
this call does nothing and returns TRUE.
```

TWAIN_DisableSource() cLong

```
Disables the open Data Source, if any.
This closes the source s user interface.
If there is not an enabled source, does nothing and returns TRUE.
```

TWAIN_CloseSource() cLong

```
Closes the open Data Source, if any.
If the source is enabled, disables it first.
If there is not an open source, does nothing and returns TRUE.
```

TWAIN_CloseSourceManager(hWnd cLong) cLong

```
Closes the Data Source Manager, if it is open.
If a source is open, disables and closes it as needed.
If the Source Manager is not open, does nothing and returns TRUE.
```

TWAIN_UnloadSourceManager() cLong

Unloads the Data Source Manager i.e. TWAIN.DLL - releasing any associated memory or resources.
This call will fail if the Source Manager is open, otherwise it always succeeds and returns TRUE.

TWAIN_MessageHook(lpMsg As Any) cLong

This function detects Windows messages that should be routed to an enabled Data Source, and picks them off. In a full TWAIN app, TWAIN_MessageHook is called inside the main GetMessage loop.
The skeleton code looks like this:

```
MSG msg
while (GetMessage((LPMSG)&msg, NULL, 0, 0)) {
    if (!TWAIN_MessageHook ((LPMSG)&msg)) {
        TranslateMessage ((LPMSG)&msg)
        DispatchMessage ((LPMSG)&msg)
    }
}   while
```

TWAIN_WaitForNativeXfer(hWnd cLong) cLong

TWAIN_ModalEventLoop()

Process messages until termination, source disable, or image transfer.

TWAIN_EndXfer() cLong

TWAIN_AbortAllPendingXfers() cLong

--------- High-level Capability Negotiation Functions --------

These functions should only be called in State 4 (TWAIN_SOURCE_OPEN)

TWAIN_NegotiateXferCount(nXfers cLong) cLong

Negotiate with open Source the number of images application will accept.
nXfers = -1 means any number

TWAIN_NegotiatePixelTypes(wPixTypes cLong) cLong

Negotiate with the source to restrict pixel types that can be acquired.
This tries to restrict the source to a *set* of pixel types,
See TWAIN_AcquireNative above for some mask constants.
A parameter of 0 (TWAIN_ANYTYPE) causes no negotiation & no restriction.
You should not assume that the source will honor your restrictions, even if this call succeeds!

----- Unit of Measure

TWAIN unit codes (from twain.h)

```
Global Const TWUN_INCHES = 0
Global Const TWUN_CENTIMETERS = 1
Global Const TWUN_PICAS = 2
Global Const TWUN_POINTS = 3
Global Const TWUN_TWIPS = 4
Global Const TWUN_PIXELS = 5
```

TWAIN_GetCurrentUnits() cLong

Return the current unit of measure: inches, cm, pixels, etc.
Many TWAIN parameters such as resolution are set and returned
in the current unit of measure.
There is no error return - in case of error it returns 0 (TWUN_INCHES)

TWAIN_SetCurrentUnits(nUnits cLong) cLong

Set the current unit of measure for the source.
Common unit codes are given above.

1. Most sources do not support all units, some support *only* inches!
2. If you want to get or set resolution in DPI, make sure the current units are inches, or you might get Dots-Per-Centimeter!
3. Similarly for ImageLayout, see below.

TWAIN_GetBitDepth() cLong

Get the current bitdepth, which can depend on the current PixelType.
Bit depth is per color channel e.g. 24-bit RGB has bit depth 8.
If anything goes wrong, this function returns 0.

TWAIN_SetBitDepth(nBits cLong) cLong

(Try to) set the current bitdepth (for the current pixel type).
Note: You should set a PixelType, then set the bitdepth for that type.

------- TWAIN Pixel Types (from twain.h)

```
Global Const TWPT_BW = 0  Black and White
Global Const TWPT_GRAY = 1
Global Const TWPT_RGB = 2
Global Const TWPT_PALETTE = 3
```

TWAIN_GetPixelType() cLong

Ask the source for the current pixel type.
If anything goes wrong (it shouldn t), this function returns 0 (TWPT_BW).

TWAIN_SetCurrentPixelType(nPixType cLong) cLong

Try to set the current pixel type for acquisition.

The source may select this pixel type, but don t assume it will.

TWAIN_GetCurrentResolution() cDouble

Ask the source for the current (horizontal) resolution.
Resolution is in dots per current unit! (See TWAIN_GetCurrentUnits above)
If anything goes wrong (it shouldn t) this function returns 0.0

TWAIN_GetYResolution() cDouble

Returns the current vertical resolution, in dots per *current unit*.
In the event of failure, returns 0.0.

TWAIN_SetCurrentResolution(dRes cDouble) cLong

Try to set the current resolution (in both x & y).
Resolution is in dots per current unit! (See TWAIN_GetCurrentUnits above)
Note: The source may select this resolution, but don t assume it will.

TWAIN_SetXResolution(dxRes cDouble) cLong

TWAIN_SetYResolution(dyRes cDouble) cLong

TWAIN_SetContrast(dCon cDouble) cLong

Try to set the current contrast for acquisition.
The TWAIN standard *says* that the range for this cap is -1000 ... +1000

TWAIN_SetBrightness(dBri cDouble) cLong

Try to set the current brightness for acquisition.
The TWAIN standard *says* that the range for this cap is -1000 ... +1000

TWAIN_SetThreshold(dThresh cDouble) cLong

Try to set the threshold for black and white scanning.
Should only affect 1-bit scans i.e. PixelType == TWPT_BW.
The TWAIN default threshold value is 128.
After staring at the TWAIN 1.6 spec for a while, I imagine that it implies
that for 8-bit samples, values >= nThresh are thresholded to 1, others to 0.

TWAIN_GetCurrentThreshold() cDouble

TWAIN_SetXferMech(mech cLong) cLong

TWAIN_XferMech() cLong

Try to set or get the transfer mode - one of the following:
Note: EZTWAIN does not yet support file or memory transfer directly.

```
Global Const XFERMECH_NATIVE = 0
Global Const XFERMECH_FILE = 1
Global Const XFERMECH_MEMORY = 2
```

TWAIN_SupportsFileXfer() cLong

Returns TRUE(1) if the open DS claims to support file transfer mode (XFERMECH_FILE)
This mode is optional. If TRUE, you can use AcquireFile.

TWAIN_SetPaperSize(nPaper cLong) cLong

See TWSS_* in TWAIN.H

```
Global Const PAPER_A4LETTER = 1
Global Const PAPER_B5LETTER = 2
Global Const PAPER_USLETTER = 3
Global Const PAPER_USLEGAL = 4
Global Const PAPER_A5 = 5
Global Const PAPER_B4 = 6
Global Const PAPER_B6 = 7
Global Const PAPER_USLEDGER = 9
Global Const PAPER_USEXECUTIVE = 10
Global Const PAPER_A3 = 11
Global Const PAPER_B3 = 12
Global Const PAPER_A6 = 13
Global Const PAPER_C4 = 14
Global Const PAPER_C5 = 15
Global Const PAPER_C6 = 16
```

-------- Document Feeder -------

TWAIN_HasFeeder() cLong

Return TRUE(1) if the source indicates it has a document feeder.
Note: A FALSE(0) is returned if the source does not handle this inquiry.

TWAIN_IsFeederSelected() cLong

Return TRUE(1) if the document feeder is selected.

TWAIN_SelectFeeder(fYes cLong) cLong

(Try to) select or deselect the document feeder.
Return TRUE(1) if successful, FALSE(0) otherwise.

TWAIN_IsAutoFeedOn() cLong

Return TRUE(1) if automatic feeding is enabled, otherwise FALSE(0).
Make sure the feeder is selected before calling this function.

TWAIN_SetAutoFeed(fYes cLong) cLong

```
(Try to) turn on/off automatic feeding thru the feeder.
Return TRUE(1) if successful, FALSE(0) otherwise.
This function selects the feeder, so to start scanning
pages from an ADF, just do: TWAIN_SetAutoFeed(TRUE).
```

TWAIN_IsFeederLoaded() cLong

```
Return TRUE(1) if there are documents in the feeder.
Make sure the feeder is selected before calling this function.
```

```
-------- Duplex Scanning -------
```

TWAIN_GetDuplexSupport() cLong

```
Query the device for duplex scanning support.
  Return values:
0    = no support (or error, or query not recognized)
1    = 1-pass duplex
2    = 2-pass duplex
```

TWAIN_EnableDuplex(fYes cLong) cLong

```
Enable (fYes=1) or disable (fYes=0) duplex scanning.
Returns TRUE(1) if successful, FALSE(0) otherwise.
```

TWAIN_IsDuplexEnabled() cLong

```
Returns TRUE(1) if the device supports duplex scanning
and duplex scanning is enabled.  FALSE(0) otherwise.
```

--------- Other exotic capabilities --------

TWAIN_HasControllableUI() cLong

```
Return 1 if source claims UI can be hidden (see SetHideUI above)
Return 0 if source says UI *cannot* be hidden
Return -1 if source (pre TWAIN 1.6) cannot answer the question.
```

TWAIN_SetIndicators(bVisible As Boolean) cLong

```
Tell the source to show (hide) progress indicators during acquisition.
```

TWAIN_Compression() cLong

TWAIN_SetCompression(compression cLong) cLong

Set/Return compression style for transferred data

```
TWAIN_Tiled() cLong

TWAIN_SetTiled(bTiled As Boolean) cLong
```

Set/Return whether source does memory xfer via strips or tiles.
bTiled = TRUE if it uses tiles for transfer.

```
TWAIN_PlanarChunky() cLong

TWAIN_SetPlanarChunky(shape As Integer) cLong
```

Set/Return current pixel shape (TWPC_CHUNKY or TWPC_PLANAR), or -1.

```
Global Const CHUNKY_PIXELS = 0
Global Const PLANAR_PIXELS = 1
```

```
TWAIN_PixelFlavor() cLong

TWAIN_SetPixelFlavor(flavor cLong) cLong
```

Set/Return pixel flavor - whether 0 is black or white:

```
Global Const CHOCOLATE_PIXELS = 0   zero pixel represents darkest shade
Global Const VANILLA_PIXELS = 1     zero pixel represents lightest shade
```

```
TWAIN_SetLightPath(bTransmissive As Boolean) cLong
```

Tries to select transparent or reflective media on the open source.
A parameter of TRUE(1) means transparent, FALSE(0) means reflective.
A return of TRUE(1) implies success, FALSE(0) means that
the data source refused the request.

```
TWAIN_SetAutoBright(bOn As Boolean) cLong

TWAIN_SetGamma(dGamma cDouble) cLong

TWAIN_SetShadow(d cDouble) cLong        0..255

TWAIN_SetHighlight(d cDouble) cLong     0..255
```

--------- Image Layout (Region of Interest) --------

```
TWAIN_SetImageLayout(L cDouble, t cDouble, r cDouble, b cDouble) cLong
```

Specify the area to scan, sometimes called the ROI (Region of Interest)
This call is like a capability negotiation, and is only valid in
State 4. L, T, R, B = distance to left, top, right, and bottom
edge of area to scan, measured in the current unit of measure,
from the top-left corner of the original page (TWAIN 1.6 8-22)

--------- Fun With Containers --------

Capability values are passed thru the TWAIN API in blocks of global memory called containers . EZTWAIN represents these containers with an opaque handle (an integer) called HCONTAINER. The following functions should provide reasonably comprehensive access to the contents of containers. See also: GetCap, SetCap.

There are four shapes of containers, which I call formats . Depending on its format, a container holds some items - these are simple data values, all the same type in a given container.

Container formats, same codes as in TWAIN.H

```
Global Const twCONTAINER_Array = 3
Global Const twCONTAINER_Enumeration = 4
Global Const twCONTAINER_OneValue = 5
Global Const twCONTAINER_Range = 6
```

CONTAINER_Free(hcon cLong)

Free the memory and resources of a capability container.

CONTAINER_Copy(hcon cLong) cLong

Create an exact copy of the container.

CONTAINER_Format(hcon cLong) cLong

Return the format of this container: CONTAINER_ONEVALUE, etc.

CONTAINER_ItemType(hcon cLong) cLong

Return the item type (what exact kind of values are in the container.)
See the TWTY_* definitions in TWAIN.H

CONTAINER_ItemCount(hcon cLong) cLong

Return the number of values in the container.
For a ONEVALUE, return 1.

CONTAINER_FloatValue(hcon cLong, n cLong) cDouble

CONTAINER_IntValue(hcon cLong, n cLong) cLong

Return the value of the nth item in the container.
n is forced into the range 0 to ItemCount(hcon)-1.

CONTAINER_ValuePtr(hcon cLong, n cLong) As Byte

```
CONTAINER_ContainsValue(hcon cLong, d cDouble) cLong
```

Return 1 (TRUE) if the value d is one of the items in the container.

```
CONTAINER_FindValue(hcon cLong, d cDouble) cLong
```

Return the index of the value d in the container, or -1 if not found.

```
CONTAINER_CurrentValue(hcon cLong) cDouble

CONTAINER_DefaultValue(hcon cLong) cDouble
```

Return the container s current or power-up (default) value.
Array containers do not have these and will return -1.0.
OneValue containers always return their (one) value.

```
CONTAINER_DefaultIndex(hcon cLong) cLong

CONTAINER_CurrentIndex(hcon cLong) cLong
```

Return the index of the Default or Current value, in an Enumeration.
Return -1 if the container is not an Enumeration.

```
CONTAINER_MinValue(hcon cLong) cDouble

CONTAINER_MaxValue(hcon cLong) cDouble

CONTAINER_StepSize(hcon cLong) cDouble
```

Return the three parameters that define a Range container.
Returns -1.0 if the container is not a Range.

The following four functions create containers from scratch:
nItemType is one of the TWTY_* item types from TWAIN.H
nItems is the number of items, in an array or enumeration.
dMin, dMax, dStep are the beginning, ending, and step value of a range.

```
CONTAINER_OneValue(nItemType cLong, dVal cDouble) cLong

CONTAINER_Range(nItemType cLong, dMin cDouble, dMax cDouble, dStep cDouble) cLong

CONTAINER_Array(nItemType cLong, nItems cLong) cLong

CONTAINER_Enumeration(nItemType cLong, nItems cLong) cLong
```

```
CONTAINER_SetItem(hcon cLong, n cLong, dVal cDouble) cLong

CONTAINER_SetItemString(hcon cLong, n cLong, pzVal cPtr) cLong

CONTAINER_SetItemFrame(hcon cLong, n cLong, L cDouble, t cDouble, r cDouble, b cDouble) cLong
```

```
Set the nth item of the container to dVal or pzText, or frame(l,t,r,b).
NOTE: A OneValue is treated as an array with 1 element.
Return 1 (TRUE) if successful. 0 (FALSE) for failure:
   The container is not an array, enumeration, or onevalue
   n < 0 or n >= CONTAINER_ItemCount(hcon)
   the value cannot be represented in this container s ItemType.
```

CONTAINER_SelectDefaultValue(hcon cLong, dVal cDouble) cLong

CONTAINER_SelectDefaultItem(hcon cLong, n cLong) cLong

CONTAINER_SelectCurrentValue(hcon cLong, dVal cDouble) cLong

CONTAINER_SelectCurrentItem(hcon cLong, n cLong) cLong

```
Select the current or default value within an enumeration or range,
by specifying either the value, or its index.
Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.
This will fail if:
   The container is not an enumeration or range.
   dVal is not one of the values in the container
   n < 0 or n >= CONTAINER_ItemCount(hcon)
```

CONTAINER_DeleteItem(hcon cLong, n cLong) cLong

```
Delete the nth item from an Array or Enumeration container.
Returns 1 (TRUE) for success, 0 (FALSE) otherwise. Failure causes:
  invalid container handle
  container is not an array or enumeration
  n < 0 or n >= ItemCount(hcon)
```

CONTAINER_InsertItem(hcon cLong, n cLong, dVal cDouble) cLong

```
Insert an item with value dVal into the container at position n.
If n = -1, the item is inserted at the end of the container.
```

--------- Low-level Capability Negotiation Functions --------

```
Setting a capability is valid only in State 4 (TWAIN_SOURCE_OPEN)
Getting a capability is valid in State 4 or any higher state.
```

TWAIN_Get(uCap cLong) cLong

```
Issue a DAT_CAPABILITY/MSG_GET to the open source.
Return a capability  container  - the  MSG_GET  value of the capability.
Use CONTAINER_* functions to examine and modify the container object.
Use CONTAINER_Free to release it when you are done with it.
A return value of 0 indicates failure:  Call GetConditionCode
or ReportLastError above.
```

TWAIN_GetDefault(uCap cLong) cLong

TWAIN_GetCurrent(uCap cLong) cLong

Issue a DAT_CAPABILITY/MSG_GETDEFAULT or MSG_GETCURRENT. See Get above.

TWAIN_Set(uCap cLong, hcon cLong) cLong

Issue a DAT_CAPABILITY/MSG_SET to the open source,
using the specified capability and container.
Return value as for TWAIN_DS

TWAIN_Reset(uCap cLong) cLong

Issue a MSG_RESET on the specified capability.
State must be 4. Returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

TWAIN_GetCapBool(cap cLong, bDefault As Boolean) As Boolean

Issue a DAT_CAPABILITY/MSG_GETCURRENT on the specified capability,
assuming the value type is TW_BOOL.
If successful, return the returned value. Otherwise return bDefault.
This is only valid in State 4 (TWAIN_SOURCE_OPEN) or higher.

TWAIN_GetCapFix32(cap cLong, dDefault cDouble) cDouble

TWAIN_GetCapUint16(cap cLong, nDefault As Integer) cLong

TWAIN_SetCapFix32(cap cLong, dVal cDouble) cLong

TWAIN_SetCapOneValue(cap cLong, ItemType cLong, ItemVal cLong) cLong

Do a DAT_CAPABILITY/MSG_SET, on capability Cap (e.g. ICAP_PIXELTYPE,
CAP_AUTOFEED, etc.) using a TW_ONEVALUE container with the given item type
and value. Use SetCapFix32 for capabilities that take a FIX32 value,
use SetCapOneValue for the various ints and uints. These functions
do not support FRAME or STR items.
Return Value: TRUE (1) if successful, FALSE (0) otherwise.

TWAIN_SetCapFix32R(cap cLong, Numerator cLong, Denominator cLong) cLong

Just like TWAIN_SetCapFix32, but uses the value Numerator/Denominator
This is useful for languages that make it hard to pass double parameters.

TWAIN_GetCapCurrent(cap cLong, ItemType cLong, ByRef pVal As Any) cLong

Do a DAT_CAPABILITY/MSG_GETCURRENT on capability Cap .
Copy the current value out of the returned container into *pVal.
If the operation fails (the source refuses the request), or if the
container is not a ONEVALUE or ENUMERATION, or if the item type of the
returned container is incompatible with the expected TWTY_ type in nType,
returns FALSE. If this function returns FALSE, *pVal is not touched.

```
TWAIN_ToFix32(d cDouble) cLong
```

Convert a floating-point value to a 32-bit TW_FIX32 value that can be passed to e.g. TWAIN_SetCapOneValue

```
TWAIN_ToFix32R(Numerator cLong, Denominator cLong) cLong
```

Convert a rational number to a 32-bit TW_FIX32 value.
Returns a TW_FIX32 value that approximates Numerator/Denominator

```
TWAIN_Fix32ToFloat(nfix cLong) cDouble
```

Convert a TW_FIX32 value (as returned from some capability inquiries)
to a double (floating point) value.

--------- Lowest-level functions for TWAIN protocol --------

```
TWAIN_DS(DG cLong, DAT cLong, MSG cLong, ByRef pdata As Any) cLong
```

Pass the triplet (DG, DAT, MSG, pData) to the open data source if any.
Returns 1 (TRUE) if the result code is TWRC_SUCCESS, 0 (FALSE) otherwise.
The last result code can be retrieved with TWAIN_GetResultCode(), and the
corresponding condition code can be retrieved with TWAIN_GetConditionCode().
If no source is open this call will fail, result code TWRC_FAILURE,
condition code TWCC_NODS.

```
TWAIN_Mgr(DG cLong, DAT cLong, MSG cLong, ByRef pdata As Any) cLong
```

Pass a triplet to the Data Source Manager (DSM).
Returns 1 (TRUE) for success, 0 (FALSE) otherwise.
See GetResultCode, GetConditionCode, and ReportLastError functions
for diagnosing and reporting a TWAIN_Mgr failure.
If the Source Manager is not open, this call fails setting result code
TWRC_FAILURE, and condition code=TWCC_SEQERROR (triplet out of sequence).

--------- Constants From Twain.h --------

```
Global Const CAP_XFERCOUNT = &H1
Global Const ICAP_COMPRESSION = &H100
Global Const ICAP_PIXELTYPE = &H101
Global Const ICAP_UNITS = &H102
Global Const ICAP_XFERMECH = &H103
Global Const CAP_AUTHOR = &H1000
Global Const CAP_CAPTION = &H1001
Global Const CAP_FEEDERENABLED = &H1002
Global Const CAP_FEEDERLOADED = &H1003
Global Const CAP_TIMEDATE = &H1004
Global Const CAP_SUPPORTEDCAPS = &H1005
Global Const CAP_EXTENDEDCAPS = &H1006
Global Const CAP_AUTOFEED = &H1007
Global Const CAP_CLEARPAGE = &H1008
Global Const CAP_FEEDPAGE = &H1009
Global Const CAP_REWINDPAGE = &H100A
Global Const CAP_INDICATORS = &H100B
Global Const CAP_SUPPORTEDCAPSEXT = &H100C
Global Const CAP_PAPERDETECTABLE = &H100D
Global Const CAP_UICONTROLLABLE = &H100E
Global Const CAP_DEVICEONLINE = &H100F
Global Const CAP_AUTOSCAN = &H1010
Global Const CAP_THUMBNAILSENABLED = &H1011
Global Const CAP_DUPLEX = &H1012
Global Const CAP_DUPLEXENABLED = &H1013
Global Const CAP_ENABLEDSUIONLY = &H1014
Global Const CAP_CUSTOMDSDATA = &H1015
Global Const CAP_ENDORSER = &H1016
Global Const CAP_JOBCONTROL = &H1017
Global Const CAP_ALARMS = &H1018
Global Const CAP_ALARMVOLUME = &H1019
Global Const CAP_AUTOMATICCAPTURE = &H101A
Global Const CAP_TIMEBEFOREFIRSTCAPTURE = &H101B
Global Const CAP_TIMEBETWEENCAPTURES = &H101C
Global Const CAP_CLEARBUFFERS = &H101D
Global Const CAP_MAXBATCHBUFFERS = &H101E
Global Const CAP_DEVICETIMEDATE = &H101F
Global Const CAP_POWERSUPPLY = &H1020
Global Const CAP_CAMERAPREVIEWUI = &H1021
Global Const CAP_DEVICEEVENT = &H1022
Global Const CAP_PAGEMULTIPLEACQUIRE = &H1023
Global Const CAP_SERIALNUMBER = &H1024
Global Const CAP_FILESYSTEM = &H1025
Global Const CAP_PRINTER = &H1026
Global Const CAP_PRINTERENABLED = &H1027
Global Const CAP_PRINTERINDEX = &H1028
Global Const CAP_PRINTERMODE = &H1029
Global Const CAP_PRINTERSTRING = &H102A
Global Const CAP_PRINTERSUFFIX = &H102B
Global Const CAP_LANGUAGE = &H102C
Global Const CAP_FEEDERALIGNMENT = &H102D
Global Const CAP_FEEDERORDER = &H102E
Global Const CAP_PAPERBINDING = &H102F
Global Const CAP_REACQUIREALLOWED = &H1030
Global Const CAP_PASSTHRU = &H1031
Global Const CAP_BATTERYMINUTES = &H1032
Global Const CAP_BATTERYPERCENTAGE = &H1033
Global Const CAP_POWERDOWNTIME = &H1034
Global Const ICAP_AUTOBRIGHT = &H1100
Global Const ICAP_BRIGHTNESS = &H1101
Global Const ICAP_CONTRAST = &H1103
Global Const ICAP_CUSTHALFTONE = &H1104
Global Const ICAP_EXPOSURETIME = &H1105
Global Const ICAP_FILTER = &H1106
Global Const ICAP_FLASHUSED = &H1107
Global Const ICAP_GAMMA = &H1108
Global Const ICAP_HALFTONES = &H1109
Global Const ICAP_HIGHLIGHT = &H110A
Global Const ICAP_IMAGEFILEFORMAT = &H110C
Global Const ICAP_LAMPSTATE = &H110D
Global Const ICAP_LIGHTSOURCE = &H110E
Global Const ICAP_ORIENTATION = &H1110
Global Const ICAP_PHYSICALWIDTH = &H1111
Global Const ICAP_PHYSICALHEIGHT = &H1112
Global Const ICAP_SHADOW = &H1113
Global Const ICAP_FRAMES = &H1114
Global Const ICAP_XNATIVERESOLUTION = &H1116
Global Const ICAP_YNATIVERESOLUTION = &H1117
Global Const ICAP_XRESOLUTION = &H1118
Global Const ICAP_YRESOLUTION = &H1119
Global Const ICAP_MAXFRAMES = &H111A
Global Const ICAP_TILES = &H111B
Global Const ICAP_CCITTKFACTOR = &H111D
Global Const ICAP_LIGHTPATH = &H111E
Global Const ICAP_PIXELFLAVOR = &H111F
Global Const ICAP_PLANARCHUNKY = &H1120
Global Const ICAP_ROTATION = &H1121
Global Const ICAP_SUPPORTEDSIZES = &H1122
Global Const ICAP_THRESHOLD = &H1123
Global Const ICAP_XSCALING = &H1124
Global Const ICAP_YSCALING = &H1125
Global Const ICAP_BITORDERCODES = &H1126
Global Const ICAP_PIXELFLAVORCODES = &H1127
Global Const ICAP_JPEGPIXELTYPE = &H1128
Global Const ICAP_TIMEFILL = &H112A
Global Const ICAP_BITDEPTH = &H112B
Global Const ICAP_BITDEPTHREDUCTION = &H112C
Global Const ICAP_UNDEFINEDIMAGESIZE = &H112D
Global Const ICAP_IMAGEDATASET = &H112E
Global Const ICAP_EXTIMAGEINFO = &H112F
Global Const ICAP_MINIMUMHEIGHT = &H1130
Global Const ICAP_MINIMUMWIDTH = &H1131
Global Const ICAP_AUTODISCARDBLANKPAGES = &H1134
Global Const ICAP_FLIPROTATION = &H1136
Global Const ICAP_BARCODEDETECTIONENABLED = &H1137
Global Const ICAP_SUPPORTEDBARCODETYPES = &H1138
Global Const ICAP_BARCODEMAXSEARCHPRIORITIES = &H1139
Global Const ICAP_BARCODESEARCHPRIORITIES = &H113A
Global Const ICAP_BARCODESEARCHMODE = &H113B
Global Const ICAP_BARCODEMAXRETRIES = &H113C
Global Const ICAP_BARCODETIMEOUT = &H113D
Global Const ICAP_ZOOMFACTOR = &H113E
Global Const ICAP_PATCHCODEDETECTIONENABLED = &H113F
Global Const ICAP_SUPPORTEDPATCHCODETYPES = &H1140
Global Const ICAP_PATCHCODEMAXSEARCHPRIORITIES = &H1141
Global Const ICAP_PATCHCODESEARCHPRIORITIES = &H1142
Global Const ICAP_PATCHCODESEARCHMODE = &H1143
Global Const ICAP_PATCHCODEMAXRETRIES = &H1144
Global Const ICAP_PATCHCODETIMEOUT = &H1145
Global Const ICAP_FLASHUSED2 = &H1146
Global Const ICAP_IMAGEFILTER = &H1147
Global Const ICAP_NOISEFILTER = &H1148
Global Const ICAP_OVERSCAN = &H1149
Global Const ICAP_AUTOMATICBORDERDETECTION = &H1150
Global Const ICAP_AUTOMATICDESKEW = &H1151
Global Const ICAP_AUTOMATICROTATE = &H1152
```