# TWAIN Errata

## For Version 2.1

**August 11th, 2009**

# Purpose

The Errata Document identifies omissions or mistakes in the TWAIN Specification. This information may change before being ratified into a future version of the TWAIN Specification.

Items marked [TBD] need clarification about content or positioning in the Specification.

# History

| Date | Comment |
|---|---|
| June 29th, 2009 | Initial version |
| August 11th, 2009 | First pass, converting notes into Spec-ready format for discussion at the upcoming technical meeting. |

# Contents


**Capability Ordering** ..... **4**

**ICAP_ROTATION, ICAP_ORIENTATION Affect on ICAP_FRAMES, DAT_IMAGELAYOUT, DAT_IMAGEINFO** ..... **5**

**TW_USERINTERFACE.ShowUI, CAP_INDICATORS and TWCC_OPERATORERR** ..... **6**

**CAP_CAMERAENABLED** ..... **8**

**CAP_CAMERAORDER** ..... **9**

**ICAP_EXTIMAGEINFO** ..... **10**

**ICAP_AUTOMATICCOLORENABLED** ..... **11**

**ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE** ..... **12**

**CAP_POWERSAVETIME CAP_POWERDOWNTIME** ..... **13**

**TW_PENDINGXFERS** ..... **14**

**DG_CONTROL / DAT_CAPABILITY / MSG_SET** ..... **15**

**DG_CONTROL / DAT_PARENT / MSG_OPENDSM** ..... **16**

**Appendix A Capability Default-Values Table** ..... **17**

**DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ & MSG_XFERREADY** ..... **18**

**Extended Image Attribute Capabilities** ..... **19**

**Deprecated Items** ..... **20**

**CAP_CAMERASIDE** ..... **21**

**DAT_IDENTITY / MSG_GET and DATIDENTITY / MSG_GETCURRENT** ..... **23**

**DEVICEEVENT** ..... **24**

Description ..... 24

Application ..... 25

Source ..... 25

Values ..... 25

Required By ..... 26

Source Required Operations ..... 26

See Also ..... 26

**MSG_GET / MSG_GETCURRENT / MSG_GETDEFAULT for Array** ..... **27**

**MSG_RESET / MSG_GETDEFAULT / MSG_RESETALL / APPENDIX A Defaults** ..... **28**

**OS Errata** ..... **29**

## Capability Ordering [White Paper is done, needs to be added to Spec]

*Update Appendix A, Capability Ordering, to reflect changes in the CapOrder Flowchart White paper.*

*Add the missing capabilities that have been inserted into the Flowchart.*

*Move ICAP_JPEGPIXELTYPE to its correct location after ICAP_COMPRESSION.*

*ICAP_LIGHTSOURCE and ICAP_LIGHTPATH have been moved between CAP_FEEDERTYPE and ICAP_NATIVERESOLUTION to accommodate different available physical sizes based on light path.*

*ICAP_IMAGEFILEFORMAT has been moved between ICAP_PIXELTYPE and ICAP_ICCPROFILE to accommodate ACD and make ICAP_COMPRESSION dependent on both ICAP_BITDEPTH and ICAP_IMAGEFILEFORMAT*

The impact of moving ICAP_IMAGEFILEFORMAT should be low based on:

- File transfer is not implemented or used as much as Native or Memory transfer.
- Most DS do not constrain PixelType based on FileFormat. ( even Inspector TWAIN does not enforcing this rule any more. As of ver 3.0.9 4/1/2009 )
- Applications have second chance to state what FileFormat will be used during DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET.
- The Spec states the main purpose of negotiating ICAP_IMAGEFILEFORMAT is to constrain the DS to only a subset of file formats. The second purpose is to set the context for other capability negotiations. ICAP_PIXELTYPE is not mentioned.

> Informs the application which file formats the Source can generate (MSG_GET). Tells the Source which file formats the application can handle (MSG_SET).
>
> Use this ICAP to determine which formats are available for file transfers, and set the context for other capability negotiations such as ICAP_COMPRESSION.
>
> Be sure to use the DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET operation to specify the format to be used for a particular acquisition.

Spec changes from FileFormat being moved:

- The only place in the spec that suggests that FileFormat should be negotiated before pixel type is in Appendix A p653 (A-27) **General Capability Negotiation**. This will need to be updated with the new cap order and missing caps.

### ICAP_ROTATION, ICAP_ORIENTATION Affect on ICAP_FRAMES, DAT_IMAGELAYOUT, DAT_IMAGEINFO

*When an application sets ICAP_FAMES ICAP_ORIENTATION can be ignored.*

*Origin not defined consistently. In ImageInfo it talks about page origin not scanner origin. Clarify that scanner origin is what is meant. Image Info probably has not been updated after center feed scanners.*

## TW_USERINTERFACE.ShowUI, CAP_INDICATORS and TWCC_OPERATORERR

*This commentary is best positioned in the Application Implementation and Source Implementation sections. There should be references to it from TW_USERINTERFACE, CAP_INDICATORS and the entire Return Code and Condition Code section. With the proposed changes there is no way of knowing if the App will be able to retrieve error messages from the DS if they are suppressed by turning off CAP_INDICATORS. It is possible for an application to get the message using TW_STATUSUFT8. We need to add a capability so an application can indicate it will handle displaying all messages. The entire spec needs to be scrubbed to reflect these changes.*

### User Interface

Sources that report TRUE for CAP_UICONTROLLABLE must allow acquisition with the UI disabled, and they must support CAP_INDICATORS.

If the Application sets ShowUI to TRUE before calling MSG_ENABLEDS, then the Source displays its user interface. CAP_INDICATORS is ignored. A progress indicator is displayed during acquisition and transfer, and errors can result in the Source showing a dialog to the user.

If the Application sets ShowUI to FALSE, but CAP_INDICATORS to TRUE before calling MSG_ENABLEDS, then the Source does not display its user interface. But a progress indicator is still displayed during acquisition and transfer, and an error can result in the Source showing a dialog to the user.

If the Application sets ShowUI to FALSE and CAP_INDICATORS to FALSE before calling MSG_ENABLEDS, then the Source is not allowed to display any kind of user interface, progress indicator or error dialog. All UI activity must be suppressed.

*These additional items need to be fixed…*

### 4-37 Alternatives to Using the Source's User Interface

o The Source still displays a progress indicator during the acquisition. The application can suppress this by setting CAP_INDICATORS to FALSE, if the Source specified CAP_UICONTROLLABLE as TRUE.

• The Source still displays errors and other messages related to the operation of its device. The application can suppress this by setting CAP_INDICATORS to FALSE, if the Source specified CAP_UICONTROLLABLE as TRUE.

### 5-8 Error and Device Control Indicators

The Source knows what is happening with the device it controls. Therefore, the Source is responsible for determining when and what information regarding errors and device controls (ex. "place paper in document feeder") should be presented to the user. Error information should be placed by the Source on top of either the application's or Source's user interface. Do not present error messages regarding capability negotiation to the user since this should be transparent.

### Progress Indicators

The Source should display appropriate progress indicators for the user regarding the acquisition and/or transfer processes. The Source must provide this information regardless of whether or not its user interface is displayed (ShowUI equals TRUE or FALSE). To suppress the indicators when the user interface is not displayed, the application should negotiate the CAP_INDICATORS capability to be FALSE.

7-103 **Note:** While the Source's user interface is raised, the Source is responsible for presenting the user with appropriate progress indicators regarding the acquisition and transfer processes unless the application has set CAP_INDICATORS to FALSE.

## 9-52 CAP_INDICATORS

### Description

If TRUE, the Source displays a progress indicator during acquisition and transfer, regardless of whether the Source's user interface is active. If FALSE, the progress indicator is suppressed if the Source's user interface is inactive.

The Source displays device-specific instructions and error messages if either the user interface or progress indicator is turned on. In this case it returns TWCC_OPERATIONERROR to alert the application that it handled the error, and communicated the problem to the user.

If both the user interface and progress indicator are turned off, then the Source never displays any message to the user, even if TWCC_OPERATIONERROR is returned. Messages to the user are under the sole control of the Application.

## CAP_CAMERAENABLED

*MSG_SET would turn on SDMI mode for the DS, MSG_RESET would turn off SDMI mode for the DS*

*In SDMI mode the DS would ignore the duplexenabled capability.*

*Setting all pixel types of all sides to false should do what?*

*Ambiguity exist in the differences between DAT_FILESYSTEM and CAP_CAMERAxxx ( don't use both, if both used keep them in sync)*

*Default missing from the Table at beginning of Capabilities 9-3*

*Either CAP_CAMERAxxx or DAT_FILESYSTEM can be used for SDMI.*

*DS can support both CAP_CAMERAxxx and DAT_FILESYSTEM*

### Description

When set to true the device will deliver images from the current camera. Use this to control bottom (rear) only scanning, or Single Document Multiple Images (SDMI) mode.

### Application

This feature depends on "camera addressing". TWAIN offers more than one way to do this. DAT_FILESYSTEM must be used when setting SDMI mode, it can also be used to control rear only scanning.

CAP_CAMERASIDE is easier to use, but is only suitable to control rear only scanning. The quickest way to make this work is to set ICAP_PIXELTYPE to the desired camera, then set CAP_CAMERASIDE to the top camera, and finally set CAP_CAMERAENABLED to false.

Avoid using ICAP_PIXELTYPE **after** setting CAP_CAMERAENABLED. CAP_PIXELTYPE implicitly sets CAP_CAMERAENABLED to true for both sides of the current pixel type, and sets all other cameras to false. This supports legacy behavior. An application can always reasonably expect that setting ICAP_PIXELTYPE to TWPT_RGB and then scanning (simple or duplex) will result in getting color images.

The application is not allowed to turn off CAP_CAMERAENABLED for all cameras.

### Source

The default value of CAP_CAMERAENABLED depends on the default value of ICAP_PIXELTYPE. The cameras for that default value should be set to true. All other cameras should set to false.

When ICAP_PIXELTYPE is set or reset the source sets the current camera(s) to true and sets all others to false.

If the application attempts to set all CAP_CAMERAENABLED values to false, the source returns a status of TWRC_FAILURE / TWCC_CAPSEQERROR. At least one camera must be enabled at all times.

## CAP_CAMERAORDER

*Clarification needed. CAP_CAMERAORDER does not constrain the possible pixel types and the array containing the order of pixel types should contain all possible pixel types supported.*

### Description

This capability selects the order of output for Single Document Multiple Image (SDMI) mode based on an array of pixel types, it does not constrain the allowed pixel types.

For example, if the scanner is set up to deliver color and bitonal documents on the top (front) camera, then an array of {TWPT_RGB, TWPT_BW} will deliver first the color image, then the bitonal image, while an array of {TWPT_BW, TWPT_RGB} will deliver first the bitonal image, then the color image.

### Application

Some sources support independent ordering of color, grayscale and bitonal, while other sources may link color and grayscale together. This can be detected by setting CAP_CAMERAORDER to all of the available ICAP_PIXELTYPE values {ex: TWPT_RGB, TWPT_GRAY, TWPT_BW} followed by a MSG_GET to examine the result. In this example a source that supports full, independent control will return back exactly the same list it was set to, while a source that links pixel types together will return a reduced list, such as {TWPT_RGB, TWPT_BW}.

### Source

Camera ordering only applies when CAP_CAMERAENABLED is set for more than one pixel type on the same camera side, putting the scanner into SDMI mode.

CAP_CAMERAORDER does not control the enabling or disabling of SDMI, it has no meaning if SDMI is not turned on, therefore it should return TWRC_FAILURE / TWCC_CAPSEQERROR if SDMI is off.

The setting applies to both the top (front) and the bottom (rear), the source is not allowed to have one ordering for the top and different ordering for the bottom.

If not supported, return TWRC_FAILURE / TWCC_CAPUNSUPPORTED.

## ICAP_EXTIMAGEINFO

*Does not explain why MSG_SET is available*

*Update to reflect the default is FALSE and application checks to see if TRUE is available.*

*If TRUE is available, the source will support the DG_IMAGE/DAT_EXTIMAGEINFO/MSG_GET message.*

### Description

Allows the application to query the data source to see if it supports the operation triplet DG_IMAGE/ DAT_EXTIMAGEINFO/ MSG_GET. Support is only available if the capability is supported and the value TRUE is allowed.

When set to TRUE, the source supports the DG_IMAGE /DAT_EXTIMAGEINFO / MSG_GET message, and data will be returned by this call for any supported TWEI_ items.

When set to FALSE then the application is indicating that it will make no calls to DG_IMAGE/ DAT_EXTIMAGEINFO/ MSG_GET. FALSE is the default.

**Note:** The TWAIN API allows for an application to query the results of many advanced device/manufacturer operations. The responsibility of configuring and setting up each advanced operation lies with the device's data source user interface. Since the configuration of advanced device/manufacturer-specific operations varies from manufacturer to manufacturer, placing the responsibility for setup and configuration of advanced operations allows the application to remain device independent.

### Application

Set this capability to FALSE if there is no intent to use _IMAGE /DAT_EXTIMAGEINFO / MSG_GET. This may improve performance, since the Source is not required to collect that information from the device.

## ICAP_AUTOMATICCOLORENABLED

*The table of defaults. The default value is set as TRUE when it should be FALSE (pg 10-96)*

### Description

The Source automatically detects the pixel type of the image and returns either a color image or a non-color image specified by ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE.

### Application

When the Application sets this capability to TRUE, it must be prepared to receive a mixture of color and non-color images.

### Source

When this capability is TRUE the Source automatically determines the pixel type.

### Values

| | |
|---|---|
| **Type:** | TW_BOOL |
| **Default Value:** | FALSE |
| **Allowed Values:** | TRUE, FALSE |
| **Container for *MSG_GET:*** | TW_ENUMERATION, TW_ONEVALUE |
| ***Container of MSG_SET:*** | TW_ENUMERATION, TW_ONEVALUE |
| ***Container for*** MSG_QUERYSUPPORT: | TW_ONEVALUE |

## ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE

*P 377 8-79 ICAP missing from* ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE

| | |
|---|---|
| 2.1 ICAP_AUTOMATICCOLORENABLED | 0x1159 |
| 2.1 ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE | 0x115A |
| 2.1 ICAP_COLORMANAGEMENTENABLED | 0x115B |

## CAP_POWERSAVETIME
## CAP_POWERDOWNTIME

*These may be one and the same. POWERSAVETIME was added for TWAIN1.8 but removed after twain 1.8 because it was not described. POWERDOWNTIME is described but not defined. POWERSAVETIME was never added to a deprecated list when it was removed. The define value was reassigned to CAP_CAMERSIDE.*

CAP_POWERSAVETIME will be newly defined in the next update. That number has not been determined. Scanner manufactures may have added their own custom capabilites for negotiating power save time.

## TW_PENDINGXFERS

*The following EJ patch codes are listed but not described.*

### Patch Codes for TW_PENDINGXFERS

**Defined as Value**

```
TWEJ_NONE          0x0000
TWEJ_MIDSEPERATOR  0x0001
TWEJ_PATCH1        0x0002
TWEJ_PATCH2        0x0003
TWEJ_PATCH3        0x0004
TWEJ_PATCH4        0x0005
TWEJ_PATCH6        0x0006
TWEJ_PATCHT        0x0007
```

Page 232 (7-84) **DG_CONTROL / DAT_PENDINGXFERS / MSG_GET** EOJ is not discussed.

These items report if the image had a patch on it and if so what kind of patch it had. It's not clear why it's on the MSG_ENDXFER, the only reason I could think to have it here is if there was no image data to transfer and therefore no image metadata, but that's a chicken-n-egg problem, because the caller would have to try it to find out that they can't get it.

The numbers correspond to specific patch images (I guess these were semi-standardized by Kodak as part of their microfilm stuff). The transfer patch is a functional patch, it's equivalent to patch 4, and we tend to use it to programmatically switch between patch2 and patch3 behavior.

I have no clear recollection what the "midseperator" is.

## DG_CONTROL / DAT_CAPABILITY / MSG_SET

It was made available for TWAIN 2.0 for DS to return TW_ENUMERATION for type TW_BOOL. The intent was to make this mandatory for TWAIN 2.0 DS but this was not made clean enough.
Starting with TWAIN 2.1 Capabilities of type TW_BOOL require the datasource to use TW_ENUMERATION when talking to TWAIN 2.x applications but continue to use TW_ONEVALUE when talking to 1.x applications. Chapter 8 for each capability of type TW_BOOL should explain it needs to return TW_ONEVALUE when talking to 1.x and TW_ENUMERATIONS when talking to TWAIN 2.x applications.

Ideally, the Specification should be completely free of version and operating system dependencies. The reality is that we have several dependencies, and that in terms of the Spec they are scattered across the document. I recommend that we remove all version and OS dependent comments, and attempt to consolidate them into two sections within the Spec.

I think a new chapter is appropriate: Version and Operating System Dependencies. Any parts of the main spec that currently talk about dependencies of this sort should either be removed or referenced to this chapter.

**Container for *MSG_GET:*** TW_ONEVALUE, **// for backwards compatibility with 1.x only**
TW_ENUMERATION **// Mandatory for 2.1 and higher**

## DG_CONTROL / DAT_PARENT / MSG_OPENDSM

This triplet needs to be updated to reflect both 32bit and 64bit

**Call**

DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_PARENT, MSG_OPENDSM, pParent);

**On Windows -** pParent = points to the window handle (hWnd) that will act as the Source's "parent". The variable is of type ~~TW_INT32~~ TW_MEMREF and ~~the low word of this variable~~ must contain the window handle.

**On Macintosh -** pParent = should be a ~~32 bit~~ NULL value.

## **Appendix A** Capability Default-Values Table

Update the table with the following changes.

CAP_DUPLEXENABLED Preferred / User Read-only value
CAP_DUPLEXENABLED Preferred / No default

CAP_FEEDERALIGNMENT n/a Read-only value
CAP_FEEDERALIGNMENT n/a No default

## DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ & MSG_XFERREADY

Does not mention callback.

Upon receiving this triplet, the Source Manager posts a private message to the application's event/message loop. Since the application is forwarding all events/messages to the Source while the Source is enabled, this creates a communication device needed by the Source. When this private message is received by the Source Manager (via the DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT operation), the Source Manager will insert the MSG_XFERREADY into the TWMessage field on behalf of the Source.

## Extended Image Attribute Capabilities

The Value types are TW_xxx when they should all be TWTY_xxx

Example
**Value Type:** TW_HANDLE
Should become
**Value Type:** TWTY_HANDLE

## Deprecated Items

Add

```
#define CAP_PAGEMULTIPLEACQUIRE   0x1023    /* Added 1.8 */
#define CAP_PAPERBINDING          0x102f    /* Added 1.8 */
#define CAP_PASSTHRU              0x1031    /* Added 1.8 */
#define CAP_POWERDOWNTIME         0x1034    /* Added 1.8 */  //0x1034 is reused by CAP_CAMERASIDE
```

## CAP_CAMERASIDE

request: TWAIN 2.0 Spec, Edit/Errata suggestion:
Chapter 9, CAP_CAMERASIDE

The Description section gives the purpose or motivation for the capability, but doesn't actually describe it!

Suggestion, and it's just a suggestion:
"TWAIN models a duplex scanner as conceptually having two 'cameras' - a 'top' camera that captures the front of each page, and a 'bottom' camera that captures the back. Some devices allow these two logical cameras to operate with different settings for certain capabilities. CAP_CAMERASIDE provides a simple way to address the cameras individually: The value of CAP_CAMERASIDE determines whether subsequent capability negotiation is directed to one camera or the other, or to both."

The Application section starts with this paragraph:
"The application changes which camera it is addressing any capability that allows independent values for the top and bottom. Please note that top refers to the camera that captures the top of the sheet of paper (sometimes referred to as the front )."

First sentence is ungrammatical and I don't know how to save it, I would delete it if the Description can be turned into a cogent description of this capability.
The comment about "top" belongs in the description, it is not an Application-specific comment.
Plus, surely 'front' and 'back' are the common terms, more accurate and descriptive? Top and bottom imply that we are referring to the physical arrangement of cameras, which I assume is not at all what we are doing? Whereas 'front' and 'back' are familiar terms for the arrangement of information on paper, which I assume is the intent?
That is, if I have a scanner with an ADF designed to be loaded face-down, does the 'bottom' camera capture the front of the pages??

The Application section has this paragraph:
"The value of CAP_DUPLEXENABLED does not impact the ability of the application to negotiate the top and bottom values. That is, if CAP_DUPLEXENABLED is FALSE CAP_CAMERASIDE can still be set to TWCS_BOTTOM (if CAP_DUPLEX is TRUE)."

a) Does 'impact' mean 'affect'?
How about "CAP_CAMERASIDE and CAP_DUPLEXENABLED are independent and have no effect on each other."

b) The interaction of CAP_CAMERASIDE and CAP_DUPLEXENABLED is not an Application programming issue, it's at least as important to Source developers - so this should go in the Description or (in my opinion) in the Source section.

c) And CAP_DUPLEX isn't a boolean, so "(if CAP_DUPLEX is TRUE)" should be deleted. It would be clearer to recommend that simplex sources either not implement CAP_CAMERASIDE, or return TWCC_BADVALUE on an attempt to set TWCS_BOTTOM.

The Source section requires DAT_FILESYSTEM to keep "in sync" with CAP_CAMERASIDE. Is that actually explicit enough for a Source writer to understand what they have to do? I bet what we really want to say is "Mixing camera selection using DAT_FILESYSTEM and CAP_CAMERASIDE is not recommended, and may

produce unexpected results."

What is the intended effect of doing a MSG_SET with an enumeration? I'm hoping it isn't expected to constrain the UI to only allow the user to select and manipulate capabilities for a subset of the sides/cameras...
It would be a kindness to Source developers to say that MSG_SET with an enumeration just takes the current value from the Enum and sets that as the current value of the cap.

Many thanks for taking time to read this,

## DAT_IDENTITY / MSG_GET and DATIDENTITY / MSG_GETCURRENT

*From application to DSM during State 3*

*From application to DS during state 4 and higher.*

DAT_IDENTITY / MSG_GETCURRENT is not documented or mentioned anywhere.

DAT_IDENTITY / MSG_GET is currently consumed by the DS and not the DSM. If an application is going to send this command in state 3 to be consumed by the DSM then it should be documented.

## DEVICEEVENT

*This is an update to the capability that is already defined.*

## Capabilities in Categories of Functionality

### Asynchronous Device Events

| | |
|---|---|
| CAP_DEVICEEVENT | MSG_SET selects which events the application wants the source to report; MSG_RESET ~~returns the preferred settings of the source~~ resets the capability to the empty array (no events set). |

## CAP_DEVICEEVENT

### Description

MSG_SET selects which events the Application wants the Source to report. MSG_GET gets available settings. MSG_GETCURRENT gets the current setting. MSG_RESET resets the capability to the empty array (no events set).

| | |
|---|---|
| TWDE_CHECKAUTOMATICCAPTURE: | The automatic capture settings on the device have been changed by the user. |
| TWDE_CHECKBATTERY: | The status of the battery has changed. |
| TWDE_CHECKFLASH: | The flash setting on the device has been changed by the user. |
| TWDE_CHECKPOWERSUPPLY: | The power supply has been changed (for instance, the user may have just connected AC to a device that was running on battery power). |
| TWDE_CHECKRESOLUTION: | The x/y resolution setting on the device has been changed by the user. |
| TWDE_DEVICEADDED: | The user has added a device (for instance a memory card in a digital camera). |
| TWDE_DEVICEOFFLINE: | A device has become unavailable, but has not been removed. |
| TWDE_DEVICEREADY: | The device is ready to capture an image. |
| TWDE_DEVICEREMOVED: | The user has removed a device. |
| TWDE_IMAGECAPTURED: | The user has captured an image to the device's internal storage. |
| TWDE_IMAGEDELETED: | The user has removed an image from the device's internal storage. |
| TWDE_PAPERDOUBLEFEED: | Two or more sheets of paper have been fed together. |
| TWDE_PAPERJAM: | The device's document feeder has jammed. |
| TWDE_LAMPFAILURE: | The device's light source has failed. |
| TWDE_CHECKDEVICEONLINE: | The device has been turned off and on. |
| TWDE_POWERSAVE: | The device has powered down to save energy. |

| | |
|---|---|
| TWDE_POWERSAVENOTIFY: | The device is about to power down to save energy. |
| TWDE_CUSTOMEVENTS: | Baseline for events specific to a given Source. |

### Application

Set all values and process the TWRC_FAILURE / TWCC_CHECKSTATUS (if returned) to identify those items supported by the Source. MSG_GET to get a list of supported items. MSG_GETCURRENT to get a list of currently enabled items.

### Source

The startup default must be an empty array. Generate TWRC_FAILURE / TWCC_CHECKSTATUS and remove unsupported events when an Application requests events not supported by the Source.

If not supported, return TWRC_FAILURE / TWCC_CAPUNSUPPORTED.

If Operation is not supported, return TWRC_FAILURE, TWCC_CAPBADOPERATION. (See DG_CONTROL /DAT_CAPABILITY/ MSG_QUERYSUPPORT)

Please note that the actions of an Application must never directly generate a device event. For instance, if the user deletes an image using the controls on the device, then the Source should generate an event. If, however, an Application deletes an image in the device (using DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE), then the Source must not generate an event.

### Values

| | |
|---|---|
| ***Type:*** | TW_UINT16 |
| ***Default Value:*** | (empty array) |
| ***Allowed Values:*** | TWDE_CHECKAUTOMATICCAPTURE<br>TWDE_CHECKBATTERY<br>TWDE_CHECKDEVICEONLINE<br>TWDE_CHECKFLASH<br>TWDE_CHECKPOWERSUPPLY<br>TWDE_CHECKRESOLUTION<br>TWDE_DEVICEADDED<br>TWDE_DEVICEOFFLINE<br>TWDE_DEVICEREADY<br>TWDE_DEVICEREMOVED<br>TWDE_IMAGECAPTURED<br>TWDE_IMAGEDELETED<br>TWDE_PAPERDOUBLEFEED<br>TWDE_PAPERJAM<br>TWDE_LAMPFAILURE<br>TWDE_POWERSAVE<br>TWDE_POWERSAVENOTIFY<br>TWDE_CUSTOMEVENTS 0x8000 |
| ***Container for*** MSG_GET***:*** | TW_ARRAY |
| ***Container for*** MSG_SET***:*** | TW_ARRAY |

***Container for*** MSG_QUERYSUPPORT*:* TW_ONEVALUE

## Required By

None

## Source Required Operations

None

## See Also

DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT
DG_CONTROL / DAT_DEVICEEVENT / MSG_GET

Device Events Article

## MSG_GET / MSG_GETCURRENT / MSG_GETDEFAULT for Array

*There are several capabilities that return TW_ARRAY and they have different expectations for MSG_GET*

*MSG_GET on a TW_ARRAY should return a list of all Available items ( ICAP_FILTERS & CAP_EXTENDEDCAPS)*

*MSG_GETCURRENT on a TW_ARRAY should return a list of the current items. On power on this is the Mandatory list.*

*MSG_GETDEFAULT on a TW_ARRAY should return a list of all preferred items*

*MSG_RESET sets the current to the back to the Mandatory list.*

*This is reflected in Chap 4-3 (p78) by the note that MSG_GET would indicate all available.*
To set the Current Value:
Use DG_CONTROL / DAT_CAPABILITY / MSG_SET and one of the following containers:
- TWON_ONEVALUE: Place the desired value in TW_ONEVALUE.Item.
- TWON_ARRAY: Place only the desired items in TW_ARRAY.ItemList.

These must be a subset of the items returned by the Source from a MSG_GET operation.

*And also reflected in CAP_EXTENDEDCAPS chap 10-47(p469)*
MSG_GETCURRENT provides a list of all capabilities which the Source and application have agreed to negotiate in States 5 and 6.
MSG_GET provides a list of all capabilities the Source is willing to negotiate in States 5 and 6.
MSG_SET specifies which capabilities the application wants to negotiate in States 5 and 6.

*This is contradicted in Chap 7-12 (p160) DG_CONTROL / DAT_CAPABILITY / MSG_GET*
Set ConType to the container type your Source uses for this capability. For container types of TWON_ARRAY and TWON_ONEVALUE provide the Current Value. For container types TWON_ENUMERATION and TWON_RANGE provide the Current, Default and Available Values.

*ICAP_FILTERS should be updated to reflect that MSG_GET returns a list of supported*

*If MSG_GET is to act like MSG_GETCURRENT then to handle getting all supported values for an array add new messages for just arrays*

*MSG_GETSUPPORTED on a TW_ARRAY should return a list of all Available items*

*MSG_SETSUPPORTED on a TW_ARRAY should constrain the a list of all Available items*

## MSG_RESET / MSG_GETDEFAULT / MSG_RESETALL / APPENDIX A Defaults

*The Defaults section in the Appendix goes into greater detail and describes three types of defaults. The rest of the document should be scrubbed to reflect these details. This section in the Appendix should be moved to Chap 4 Capabilities. These values should be referred to the as the power-on mandatory and preferred current values. Calling them defaults adds to the confusion.*

*Two interpellation s of the spec when there are times when the preferred value for the scanner is different than the recommended or mandatory value of the spec, when it should be first interpellation.*

1. *Current should be the Mandatory value, default should be the preferred value. MSG_RESET and MSG_RESETALL will return the capability to this state.*

2. *Current should be the Mandatory value, default should be the preferred value. MSG_RESET and MSG_RESETALL will return both current and default to the Mandatory value.*

*MSG_RESETALL*

**Description**

This command resets all of the current values to their mandatory or preferred values and removes all constraints returning them to their defaults for all of the negotiable capabilities supported by the driver.

## OS Errata

**Page 9, (1-3)**

*Remove. Make this part of the new chapter. All 32-bit and 64-bit versions of Windows are supported. Go back to TWAIN 1.9 for 16-bit Windows support. All versions of Macintosh OSX are supported. Go back to 1.9 for Macintosh version 9 and earlier. Not sure what to say about OS/2, drop it? Not sure what to say about Windows CE, in theory we'll work on any 32/64-bit processor, there are just no drivers.*

**Note:** As of this writing TWAIN is supported on the following operating systems: all versions of Apple Macintosh, Microsoft Windows 3.x / 9x / NT /Me /XP and Windows 2000. TWAIN is not available on Windows CE. TWAIN is available on IBM OS/2, but the binaries for the Source Manager were not built or distributed by the TWAIN Working Group.

**Page 10, (1-4)**

*Update. This entire page talks about some pretty old stuff (like the agreement with Microsoft). I think this particular part could stay as-is, though Motif should be removed and Linux added.*

**Extensibility** — The architecture must include the flexibility to embrace multiple windowing environments spanning various host platforms (Macintosh, Microsoft Windows, Motif, etc.) and facilitate the exchange of various data types between Source devices and destination applications. Currently, only the raster image data type is supported but suggestions for future extensions include text, facsimile, vector graphics, and others.

**Page 16, (2-6)**

*Update. This is the current format. We'd like to remove code references from the Spec, but this is one place where it's appropriate to leave it. The new format is below in red, and it should replace both the Windows and Macintosh formats.*

```
TW_UINT16 FAR PASCAL DSM_Entry
( pTW_IDENTITY pOrigin,        // source of message
  pTW_IDENTITY pDest,          // destination of message
  TW_UINT32 DG,                // data group ID: DG_xxxx
  TW_UINT16 DAT,               // data argument type: DAT_xxxx
  TW_UINT16 MSG,               // message ID: MSG_xxxx
  TW_MEMREF pData              // pointer to data
);

TW_UINT16 TW_CALLINGSTYLE DSM_Entry
( pTW_IDENTITY pOrigin,        // source of message
  pTW_IDENTITY pDest,          // destination of message
  TW_UINT32 DG,                // data group ID: DG_xxxx
  TW_UINT16 DAT,               // data argument type: DAT_xxxx
  TW_UINT16 MSG,               // message ID: MSG_xxxx
  TW_MEMREF pData              // pointer to data
);
```

**Page 17, (2-7)**

*Update. We're in the same situation here as on page 16.*

```
TW_UINT16 FAR PASCAL DS_Entry
(       pTW_IDENTITY pOrigin,       // source of message
        TW_UINT32 DG,               // data group ID: DG_xxxx
        TW_UINT16 DAT,              // data argument type: DAT_xxxx
        TW_UINT16 MSG,              // message ID: MSG_xxxx
        TW_MEMREF pData             // pointer to data
);

TW_UINT16 TW_CALLINGSTYLE DS_Entry
(       pTW_IDENTITY pOrigin,       // source of message
        TW_UINT32 DG,               // data group ID: DG_xxxx
        TW_UINT16 DAT,              // data argument type: DAT_xxxx
        TW_UINT16 MSG,              // message ID: MSG_xxxx
        TW_MEMREF pData             // pointer to data
);
```

**Page 17, (2-7)**

*Update. This information is important, it could stay here, but it will have to be repeated in the OS Specific Chapter. Remove the portion in red, it only applies to 16-bit Windows systems.*

### On Windows

• The Source Manager for Windows is a Dynamic Link Library (TWAINDSM.DLL).
• The Source Manager can manage simultaneous sessions between many applications with many Sources. That is, the same instance of the Source Manager is shared by multiple applications.

### On Macintosh

• The Source Manager for Macintosh is a Mach-O framework (TWAIN.framework).

### On Linux

• The Source Manager is a shared library (libtwain.so).

**Page 18, (2-8)**

*Update. This information is important, it could stay here, but it will have to be repeated in the OS Specific Chapter. Remove the portion in red, it only applies to 16-bit Windows systems.*

### On Windows

• The Source is a Dynamic Link Library (DLL) so applications share the same copy of each element.

### On Macintosh

• The Source is implemented as a bundle (preferably Mach-O).

### On Linux

• The Source is a shared library (.so).

**Page 19, (2-9)**

*Update. This part is a problem. The new callback scheme works for all operating systems, and is preferred because it allows for better integration with State 4. But TWAIN has a long history of processing TWRC_DSEVENT. My recommendation is to describe the new scheme that works on all platforms and move the platform specific content to the new OS dependencies chapter. Remove crossed out items, add items in red.*

### Applications

TWAIN Applications ~~running on Linux or Apple Macintosh OS X must~~ that detect the presence of the DF_DSM2 flag inside of TW_IDENTITY.SupportedGroups use DG_CONTROL / DAT_CALLBACK / MSG_REGISTERCALLBACK to register to receive asynchronous notifications for events like MSG_XFERREADY.

~~TWAIN Applications on Microsoft Windows that detect the presence of the DF_DSM2 flag inside of TW_IDENTITY.SupportedGroups are encouraged to use DAT_CALLBACK instead of processing TWRC_DSEVENT from DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT. The legacy TWAIN 1.x behavior is still supported by the Data Source Manager.~~

TWAIN Applications ~~on Microsoft Windows~~ using older versions of the Data Source Manager (no DF_DSM2 flag detected) must use ~~the~~ legacy behavior. Please refer to the chapter on Operating System Dependencies for more information.

Please note that TWAIN Applications are advised to return as soon as possible from a callback function. Events like MSG_XFERREADY should initiate the image transfer in a different thread so that the callback can return immediately.

### Sources

~~TWAIN sources running on Apple Macintosh OS X that use a Data Source Manager that does not report DF_DSM2 in TW_IDENTITY.SupportedGroups must use DG_CONTROL / DAT_CALLBACK / MSG_INVOKECALLBACK to return events like MSG_XFERREADY.~~

~~All other~~ TWAIN sources that detect the presence of the DF_DSM2 flag inside of TW_IDENTITY.SupportedGroups ~~regardless of version or operating system must~~ use DG_CONTROL / DAT_NULL with the appropriate message to return events like MSG_XFERREADY.

TWAIN Source using older versions of the Data Source Manager (no DF_DSM2 flag detected) must use legacy behavior. Please refer to the chapter on Operating System Dependencies for more information.

**Page 20, (2-10)**

*Update. This should be migrated to the chapter on Operating System Dependencies. We can keep a placeholder here to direct the reader to go to that chapter.*

## Installation of the Data Source Manager

### Microsoft Windows

Please refer to the TWAIN 1.9 Specification for support of versions of Microsoft Windows prior

to Windows 2000.

Microsoft provides TWAIN_32.DLL for access to the legacy 1.x Data Source Manager on 32-bit systems and SYSWOW64.

TWAIN_32.DLL is not available for native 64-bit Applications. A native 64-bit TWAIN session must use TWAINDSM.DLL.

Applications that wish to use TWAINDSM.DLL, for access to the new open source Data Source Manager, must install it themselves. Please refer to the TWAIN website http://www.twain.org to obtain this file, and for installation instructions. This DSM is fully backwards compatible with all versions of TWAIN.

### Apple Macintosh

Please refer to the TWAIN 1.9 Specification for support of operating systems prior to Apple Mac OS X.

Apple provides /System/Library/Frameworks/TWAIN.framework for access to the legacy 1.x Data Source Manager.

The TWAIN 2.0 open source Data Source Manager is not currently available on any version of Apple Mac OS X. TWAIN 2.0 compliant Application and Source writers are strongly encouraged to include and check for the DF_APP2, DF_DSM2 and DF_DS2 flags, and to handle DAT_CALLBACK when these flags are detected, so they are ready when the TWAIN 2.0 Data Source Manager appears.

### Linux

Please check the TWAIN website http://www.twain.org to see if your distro is represented, and if not, please consider making a submission to the TWAIN Working Group.

Note that TWAIN 2.0 compliant Sources must support TW_USERINTERFACE.ShowUI being set to 0 (no UI), which in concert with CAP_INDICATORS set to FALSE is expected to prevent the

**Page 20, (2-10)**

*This should be migrated to the chapter on Operating System Dependencies. We can keep a placeholder here to direct the reader to go to that chapter.*

For 1.x TWAIN drivers refer to the chapter on Operating System Dependencies.

*Move this content to the Operating System Dependencies chapter.*

This applies when a 1.x Application or Source is detected.

These functions correspond to the WIN32 Global Memory functions mentioned in previous versions of the TWAIN Specification:

GlobalAlloc, GlobalFree, GlobalLock, GlobalUnlock

On MacOS/X these functions call NewPtrClear and DisposePtr. The lock and unlock

functions are no-ops, but they still must be called.

TWAIN 2.0 compliant Applications and Sources must use these calls on all platforms (Windows, MacOS/X and Linux).

The Source Manager takes the responsibility to make sure that all components are using the same memory management API's.

**Page 31, (2-21)**

*Update. This needs to be discussed in the chapter on Operating System Dependencies. Change the items in red..*

**Native**

Every Source must support this transfer mode. It is the default mode and is the easiest for an application to implement. However, it is restrictive (i.e. limited to the DIB or TIFF formats and limited by available memory).

The format of the data is platform-specific:
- Windows: DIB (Device-Independent Bitmap)
- Macintosh/Linux: A handle to a TIFF image in memory

The Source allocates a single block of memory and writes the image data into the block. It passes a pointer to the application indicating the memory location. The application is responsible for freeing the memory after the transfer.

**Page 33, (3-1)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

Mac OS X includes two high-level native development environments that you can use for your application's graphical user interface: Carbon, and Cocoa. These environments are full-featured development environments in their own right, and you can write TWAIN applications and TWAIN Data Sources in any one of these environments.

Because both Carbon and Cocoa change the event handling mechanism (no WaitNextEvent loops), these paragraphs update and extend the section of the previous specification that describes how to modify the application event loop to support TWAIN.

Carbon and Cocoa-based Mac OS X TWAIN applications are required to supply an event handler callback function that the TWAIN DSM will call. Carbon applications using the Classic Event Manager (WaitNextEvent)should continue to route all events through the Data Source of the Specification. However, Data Sources on Mac OS X can no longer use the Classic Event Manager.

**Page 34, (3-2)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

# Installation of the Source Manager Software

The TWAIN Source Manager for Microsoft Windows consists of four binaries that are owned by the TWAIN Working Group (TWG). These binaries are built and distributed by the TWG for Windows 3.x / 9x / NT, and built and distributed by Microsoft (as protected system files) for all versions of Windows 2000. These files are as follows:

TWAIN_32.DLL The 32-bit Source Manager. This is the DLL that 32-bit applications must use to communicate with TWAIN.

TWAIN.DLL The 16-bit Source Manager. This is the DLL that 16-bit applications must use to communicate with TWAIN.

TWUNK_32.EXE This program works invisibly under the hood to allow 16-bit applications to communicate with 32-bit Sources.

TWUNK_16.EXE This program works invisibly under the hood to allow 32-bit applications to communicate with 16-bit Sources.

Note that 16-bit Sources will not run correctly on Windows NT systems.

For a TWAIN-compliant application or Source to work properly, a Source Manager **must** be installed on the host system. To guarantee that a Source Manager is available, ship a copy of the latest Source Manager on your product's distribution disk and provide the user with an installer or installation instructions as suggested below. To ensure that the most recent version of the Source Manager is available to you and your user on their computer, you must do the following:

1. Look for a Source Manager:
**On Windows systems**:
Look for the file names TWAIN.DLL, TWAIN_32.DLL, TWUNK_16.EXE, and TWUNK_32.EXE in the Windows directory (this is typically C:\Windows on Windows 3.1/95/98, and C:\Winnt on Windows NT).
**On Macintosh systems**:
There's no need to install anything on Mac OS X version 10.2 and later. The system will always have the latest TWAIN.framework. For version 10.1.5 the source manager is supported only and you will need to install it seperately.


2. If no Source Manager is currently installed, install the Source Manager sent out with your application.

3. If a Source Manager already exists, check the version of the installed Source Manager. If the version provided with your application is more recent, rename the existing one as follows and install the Source Manager you shipped. To rename the existing Source Manager:
**On Windows systems**:
Rename the four files to be TWAIN.BAK, TWAIN_32.BAK, TWUNK_16.BAK, and TWUNK_32.BAK.
**On Macintosh systems**:
Move the Source Manager to the Extensions (Disabled) folder.

### How to Install the Source Manager on Microsoft Windows Systems

To allow the comparison of Source Manager versions, the Microsoft Windows Source Manager DLL has version information built into it which conforms to the Microsoft File Version Stamping specification. Application developers are strongly encouraged to take advantage of this in their installation programs. Microsoft provides the File Version Stamping Library, VER.DLL, which should be used to install the Source Manager.

VER.DLL, VER.LIB and VER.H are included in this Toolkit; VER.DLL may be freely copied and distributed with your installation program. Of course, your installation program will have to link to this DLL to use it. Documentation on the File Version Stamping Library API can be found on the Microsoft Windows SDK.

The following code fragment demonstrates how the VerInstallFile( ) function provided in VER.DLL can be used to install the Source Manager into the user's Windows directory. The following example assumes that your installation floppy disk is in the A: drive and the Source Manager is in the root of the installation disk.

```
#include "windows.h"
#include "ver.h"
#include "stdio.h"
// Max file name length is based on 8 dot 3 file name convention.
#define MAXFNAMELEN 12
// Max path name length is based on GetWindowsDirectory()
// documentation.
#define MAXPATHLEN 144
VOID InstallWinSM ( VOID )
{
DWORD dwInstallResult;
WORD wTmpFileLen = MAXPATHLEN;
WORD wLen;
char szSrcDir[MAXPATHLEN];
char szDstDir[MAXPATHLEN];
```



```
char szCurDir[MAXPATHLEN];
char szTmpFile[MAXPATHLEN];
wLen = GetWindowsDirectory( szDstDir, MAXPATHLEN );
if (!wLen || wLen>MAXPATHLEN)
{
return; // failure getting Windows dir
}
strcpy( szCurDir, szDstDir );
strcpy( szSrcDir, "a:\\" );
dwInstallResult = VerInstallFile( VIFF_DONTDELETEOLD,
"TWAIN_32.DLL",
"TWAIN_32.DLL",
szSrcDir,
szDstDir,
szCurDir,
szTmpFile,
&wTmpFileLen );
// If VerInstallFile() left a temporary copy of the new
// file in DstDir be sure to delete it. This happens
// when a more recent version is already installed.
if ( dwInstallResult & VIF_TEMPFILE &&
```

```
((wTmpFileLen - MAXPATHLEN) > MAXFNAMELEN) )
{
// when dst path is root it already ends in '\'
if (szDstDir[wLen-1] != '\\')
{
strcat( szDstDir, "\\" );
}
strcat( szDstDir, szTmpFile );
remove( szDstDir );
}
}
```

You should enhance the above code so that it handles the other three files (TWAIN.DLL, TWUNK_16.EXE, and TWUNK_32.EXE), as well as fixing it to handle low memory and other error conditions, as indicated by the dwInstallResult return code. Also note that the above code does not leave a backup copy of the user's prior Source Manager on their disk, but you should do this. Copy the older versions to TWAIN.BAK, TWAIN_32.BAK, TWUNK_16.BAK, and TWUNK_32.BAK.

*TWAIN 2.1 Specification 3-5*

### How to Install the Source Manager on Macintosh Systems

For Mac OS X version 10.2 and later, the Source Manager is installed automatically with the OS and developers should not install or modify TWAIN.framework. For Mac OS X version 10.1.5, developers will need to follow these instructions to install the Source Manager.

The file TWAIN Source Manager should be installed in the Extensions folder of the active System Folder, if the version being installed is newer than the existing version, or there is no previous version of this file.

The folder TWAIN Data Sources should be created in the Extensions folder if it does not exist. If you are a scanner vendor, install your scanner data sources into the Extensions:TWAIN Data Sources: folder you created.

The file Source Manager should be installed in the Preferences:TWAIN: folder if it does not exist, or if its version number is higher than the existing file.

The last step is very important. The file you are installing is the 68k shim file that routes calls made by older applications to the new DSM. Without this file, older applications will not be able to use the TWAIN DSM properly.

**Page ??, (3-6)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Alter the Application's Event Loop

Events include activities such as key clicks, mouse events, periodic events, accelerators, etc. Every TWAIN-compliant application on Windows needs an event loop. (On Windows, these actions are called messages but that can be confusing because TWAIN uses the term messages to describe the third parameter of an operation triplet. Therefore, we will refer to these key clicks, etc. as events in this section generically.) During a TWAIN session, the application opens one or more Sources. However, even if several Sources are open, the application should only have one

Source enabled at any given time. That is the Source from which the user is attempting to acquire data.

**When this is selected: The application does this:**

Select Source... The application requests that the Source Manager's Select Source Dialog Box appear (or it may display its own version).
After the user selects the Source they want to use, control returns to the application.

Acquire... The application requests that the Source display its user interface. (Again, the application can create its own version of a user interface or display no user interface.)



Altering the event loop serves three purposes:

- Passing events from the application to the Source so it can respond to them
- Notifying the application when the Source is ready to transfer data or have its user interface disabled
- Notifying the application when a device event occurs.

### Event Loop Modification - Events in State 4

Please note that with TWAIN 1.8 and the addition of the DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT message, it is possible to receive events after the Source has been opened but before it has been enabled (State 4). However, these events will not be sent from the Source to the Application unless the Application has negotiated for specific events using CAP_DEVICEEVENTS. Events posted in this way must use the hWnd passed to them by the DG_CONTROL / DAT_PARENT / MSG_OPENDS message. Sources are required to have all device events turned off when they are opened to support backward compatibility with older TWAIN applications.

### Event Loop Modification - Passing events (The first purpose)

While a Source is enabled, all events are sent to the application's event loop. Some of the events may belong to the application but others belong to the enabled Source. To ensure that the Source receives and processes its events, the following changes are required:

The application **must** send all events that it receives in its event loop to the Source as long as the Source is enabled. The application uses:

DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT

The TW_EVENT data structure used looks like this:

```
typedef struct {
TW_MEMREF pEvent; /* Windows pMSG */
TW_UINT16 TWMessage; /* TW message from Source to */
/* the application */
} TW_EVENT, FAR *pTW_EVENT;
```

The pEvent field points to the message structure.

The Source receives the event from the Source Manager and determines if the event belongs to it.

- **If it does**, the Source processes the event. It then sets the Return Code to TWRC_DSEVENT to indicate it was a Source event. In addition, it should set the TWMessage field of the TW_EVENT structure to MSG_NULL.
- **If it does not**, the Source sets the Return Code to TWRC_NOTDSEVENT meaning it is not a Source event. In addition, it should set the TWMessage field of the TW_EVENT structure to MSG_NULL. The application receives this information from DSM_Entry and should process the event in its event loop as normal.



### Event Loop Modification - Notifications from Source to application (The second and third purpose)

When the Source has data ready for a data transfer or it wishes to request that its user interface be disabled, it needs to communicate this information to the application asynchronously. These notifications appear in the application's event loop. They are contained in the

TW_EVENT.TWMessage field. The four notices of interest are:

- MSG_XFERREADY to indicate data is ready for transfer
- MSG_CLOSEDSREQ to request that the Source's user interface be disabled
- MSG_CLOSEDSOK to request that the Source's user interface be disabled (special case for use with DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY).
- MSG_DEVICEEVENT to report that a device event has occurred.

Therefore, the application's event loop must always check the TW_EVENT.TWMessage field following a DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT call to determine if it is the simple MSG_NULL or critical MSG_XFERREADY or MSG_CLOSEDSREQ. Information about how the application should respond to these two special notices is detailed later in this chapter in the "Controlling a TWAIN Session from Your Application" on page 3-15."

### How to Modify the Event Loop for Microsoft Windows

This section illustrates typical modifications needed in an Microsoft Windows application to support TWAIN-connected Sources.

```
TW_EVENT twEvent;
TW_INT16 rc;
while (GetMessage ( (LPMSG) &msg, NULL, 0, 0) ) {
rc = TWRC_NOTDSEVENT;
if Source is enabled {
twEvent.pEvent = (TW_MEMREF)&msg;
twEvent.TWMessage = MSG_NULL;
rc = (*pDSM_Entry) (pAppId,
pSourceId,
DG_CONTROL,
DAT_EVENT,
MSG_PROCESSEVENT,
(TW_MEMREF)&twEvent);
// check for message from Source
switch (twEvent.TWMessage) {
case MSG_XFERREADY:
SetupAndTransferImage(NULL);
break;
case MSG_CLOSEDSREQ:
DisableAndCloseSource(NULL);
break;
case MSG_CLOSEDSOK:
DisableAndCloseSource(NULL);
```



```
GetCustomDsData();
break;
case MSG_NULL:
// no message returned from the source
break;
}
}
// Source didn't process it, so we will
if (rc == TWRC_NOTDSEVENT) {
TranslateMessage( (LPMSG) &msg);
DispatchMessage( (LPMSG) &msg);
}
}
```

**Note:** Source writers are advised to keep stack space usage to a minimum. Application writers should also be aware that, in the Windows environment, sources run in their calling application's data space. They depend upon the application to reserve enough stack space for the source to be able to perform its various functions. For this reason, applications should define enough stack space in their linker DEF files for the sources

that they might use.
Two new TWAIN triplets exist to support the new event handling mechanism:
• DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK
• DG_CONTROL / DAT_CALLBACK / MSG_INVOKE_CALLBACK
A Cocoa or Carbon Event Manager application will register the callback after opening the DS using the DG_CONTROL/ DAT_CALLBACK/ MSG_REGISTER_CALLBACK triplet, and a Carbon DS will invoke the callback using the DG_CONTROL/ DAT_CALLBACK/ MSG_INVOKE_CALLBACK triplet.
The callback function should look like this:

```
TW_UINT16 TWAIN_callback(pTW_IDENTITY pOrigin,
pTW_IDENTITY pDest,
TW_UINT32 DG,
TW_UINT16 DAT,
TW_UINT16 MSG,
TW_MEMREF pData)
{
// process message the same as if it was received through
// the event loop
return TWRC_SUCCESS; // or failure etc
}
```

An application would register the callback function with this code:

```
TW_CALLBACK callback = { 0 };
callback.CallBackProc = TWAIN_callback;
Result = DSM_Entry(&appIdentity, NULL,
DG_CONTROL, DAT_CALLBACK, MSG_REGISTER_CALLBACK,
```



```
(TW_MEMREF)&callback);
```

The DS would invoke the callback thus:

```
TW_CALLBACK callback = { 0 };
callback.Message = MSG_XFERREADY;
Result = DSM_Entry(&appIdentity, NULL,
DG_CONTROL, DAT_CALLBACK, MSG_INVOKE_CALLBACK,
(TW_MEMREF)&callback);
```

DSM_Entry Call and Available Operation Triplets
As described in the Chapter 2, "Technical Overview," all actions that the application invokes on the Source Manager or Source are routed through the Source Manager. The application passes the request for the action to the Source Manager via the DSM_Entry function call which contains an operation triplet describing the requested action. In code form, the DSM_Entry function looks like this:
**On Windows:**

```
TW_UINT16 FAR PASCAL DSM_Entry
( pTW_IDENTITY pOrigin, // source of message
pTW_IDENTITY pDest, // destination of message
TW_UINT32 DG, // data group ID: DG_xxxx
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
TW_MEMREF pData // pointer to data
);
```

**On Macintosh:**

```
FAR PASCAL TW_UINT16 DSM_Entry
( pTW_IDENTITY pOrigin, // source of message
pTW_IDENTITY pDest, // destination of message
TW_UINT32 DG, // data group ID: DG_xxxx
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
```

TW_MEMREF pData // pointer to data
);
The DG, DAT, and MSG parameters contain the operation triplet. The parameters must follow these rules:

**pOrigin**
References the application's TW_IDENTITY structure. The contents of this structure must not be changed by the application from the time the connection is made with the Source Manager until it is closed.

**pDest**
Set to NULL if the operation's final destination is the Source Manager.
Otherwise, set to point to a valid TW_IDENTITY structure for an open Source.

**DG_xxxx**
Data Group of the operation. Currently, only DG_CONTROL, DG_IMAGE, and DG_AUDIO are defined. Custom Data Groups can be defined.



**DAT_xxxx**
Designator that uniquely identifies the type of data *object* (structure or variable) referenced by pData.

**MSG_xxxx**
Message specifies the action to be taken.

**pData**
Refers to the TW_xxxx structure or variable that will be used during the operation. Its type is specified by the DAT_xxxx. This parameter should always be typecast to TW_MEMREF when it is being referenced.

**Page ??, (3-18)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### State 1 to 2 - Load the Source Manager and Get the DSM_Entry

The application must load the Source Manager before it is able to call its DSM_Entry point.

#### Operations Used:

No TWAIN operations are used for this transition. Instead,



**On Windows:**
Load TWAIN_32.DLL using the LoadLibrary( ) routine.
Get the DSM_Entry by using the GetProcAddress( ) call.

**On Macintosh:**
Link against TWAIN.framework.

#### On Windows:

The application can load the Source Manager by doing the following:

```
DSMENTRYPROC pDSM_Entry;
HANDLE hDSMLib;
char szSMDir;
OFSTRUCT of;
// check for the existence of the TWAIN_32.DLL file in the Windows
// directory
GetWindowsDirectory (szSMDir, sizeof(szSMDir));
/*** Could have been networked drive with trailing '\' ***/
if (szSMDir [(lstrlen (szSMDir) - 1)] != '\\')
{ lstrcat( szSMDir, "\\" );
}
```

```
if ((OpenFile(szSMDir, &of, OF_EXIST) != -1)
{
// load the DLL
if (hDSMDLL = LoadLibrary("TWAIN_32.DLL")) != NULL)
{
// check if library was loaded
if (hDSMDLL >= (HANDLE)VALID_HANDLE)
{
if (lpDSM_Entry = (DSMENTRYPROC)GetProcAddress(hDSMDLL,
MAKEINTRESOURCE (1))) != NULL)
{
if (lpDSM_Entry )
FreeLibrary(hDSMDLL);
}
}
}
}
```

Note, the code appends TWAIN_32.DLL to the end of the Windows directory and verifies that the file exists before calling LoadLibrary( ). Applications are strongly urged to perform a dynamic run-time link to DSM_Entry( ) by calling LoadLibrary( ) rather than statically linking to TWAIN_32.LIB via the linker. If the TWAIN_32.DLL is not installed on the machine, Microsoft Windows will fail to load an application that statically links to TWAIN_32.LIB. If the Application has a dynamic link, however, it will be able to give users a meaningful error message, and perhaps continue with image acquisition facilities disabled.

After getting the DSM_Entry, the application must check pDSM_Entry. If it is NULL, it means that the Source Manager has not been installed on the user's machine and the application cannot provide any TWAIN services to the user. If NULL, the application must not attempt to call *pDSM_Entry as this would result in an Unrecoverable Application Error (UAE).



### On Macintosh:

The Source Manager is a mach-o framework (TWAIN.framework).

When building your application, you should link against TWAIN.framework. There should be no need to check for an existing Source Manager - beginning with Mac OS X 10.2, the TWAIN.framework is part of Mac OS X.

### State 2 to 3 - Open the Source Manager

The Source Manager has been loaded. The application must now open the Source Manager.

One Operation is Used:

DG_CONTROL / DAT_PARENT / MSG_OPENDSM

**pOrigin**

The application must allocate a structure of type TW_IDENTITY and fill in all fields except for the Id field. Once the structure is prepared, this pOrigin parameter should point at that structure.

During the MSG_OPENDSM operation, the Source Manager will fill in the Id field with a unique identifier of the application. The value of this identifier is only valid while the application is connected to the Source Manager.

The application must save the entire structure. From now on, the structure will be referred to by the pOrigin parameter to identify the application in every call the application makes to DSM_Entry( ).

The TW_IDENTITY structure is defined in the TWAIN.H file but for quick reference, it looks like this:

```
/* DAT_IDENTITY Identifies the program/library/code */
/* resource. */
typedef struct {
TW_UINT32 Id; /* Unique number for identification*/
```

```
TW_VERSION Version;
TW_UINT16 ProtocolMajor;
TW_UINT16 ProtocolMinor;
TW_UINT32 SupportedGroups;/*Bit field OR combination */
/*of DG_constants found in */
/*the TWAIN.H file */
TW_STR32 Manufacturer;
TW_STR32 ProductFamily;
TW_STR32 ProductName;
} TW_IDENTITY, FAR *pTW_IDENTITY;
```

**pDest**

Set to NULL indicating the operation is intended for the Source Manager.

pData

Typically, you would expect to see this point to a structure of type TW_PARENT but this is not the case. This is an exception to the usual situation where the DAT field of the triplet identifies the data structure for pData.



- **On Windows:** pData points to the window handle (hWnd) that will act as the Source's "parent". The variable is of type TW_INT32. For 16 bit Microsoft Windows, the handle is stored in the low word of the 32 bit integer and the upper word is set to zero. If running under the WIN32 environment, this is a 32 bit window handle. The Source Manager will maintain a copy of this window handle for posting messages back to the application.
- **On Macintosh:** pData should be a 32-bit NULL value.

### How to Initialize the TW_IDENTITY Structure

Here is a Windows example of code used to initialize the application's TW_IDENTITY structure.

```
TW_IDENTITY AppID; // App's identity structure
AppID.Id = 0; // Initialize to 0 (Source Manager
// will assign real value)
AppID.Version.MajorNum = 3; //Your app's version number
AppID.Version.MinorNum = 5;
AppID.Version.Language = TWLG_ENGLISH_USA;
AppID.Version.Country = TWCY_USA;
lstrcpy (AppID.Version.Info, "Your App's Version String");
AppID.ProtocolMajor = TWON_PROTOCOLMAJOR;
AppID.ProtocolMinor = TWON_PROTOCOLMINOR;
AppID.SupportedGroups = DF_APP2 | DG_IMAGE | DG_CONTROL;
lstrcpy (AppID.Manufacturer, "App's Manufacturer");
lstrcpy (AppID.ProductFamily, "App's Product Family");
lstrcpy (AppID.ProductName, "Specific App Product Name");
```

**On Windows: Using DSM_Entry to open the Source Manager**

```
TW_UINT16 rc;
rc = (*pDSM_Entry) (&AppID,
NULL,
DG_CONTROL,
DAT_PARENT,
MSG_OPENDSM,
(TW_MEMREF) &hWnd);
```

where AppID is the TW_IDENTITY structure that the application set up to identify itself and hWnd is the application's main window handle.

**On Macintosh: Using DSM_Entry to open the Source Manager**

```
rc = DSM_Entry(&AppID,
NULL,
DG_CONTROL,
DAT_PARENT,
```

MSG_OPENDSM,
NULL);
If your data source requires resources, it is responsible for loading and unloading them at run time. The Source Manager no longer manages resources automatically.

**Page ??, (3-25)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Set the Capability to Specify the Number of Images the Application can Transfer

The capability that specifies how many images an application can receive during a TWAIN session is CAP_XFERCOUNT. All Sources must support this capability. Possible values for CAP_XFERCOUNT are:
The default value allows multiple images to be transferred. The following is a simple code example illustrating the setting of a capability and specifically showing how to limit the number of images to one. Notice there are differences between the code for Windows and Macintosh applications. Both versions are included here with ifdef statements for MSWIN versus MAC.

```
TW_CAPABILITY twCapability;
TW_INT16 count;
TW_STATUS twStatus;
TW_UINT16 rc;
#ifdef _MSWIN_
pTW_ONEVALUE pval;
#endif
#ifdef _MAC_
TW_HANDLE h;
pTW_INT16 pInt16;
#endif
//-----Setup for MSG_SET for CAP_XFERCOUNT
twCapability.Cap = CAP_XFERCOUNT;
twCapability.ConType = TWON_ONEVALUE;
#ifdef _MSWIN_
twCapability.hContainer = GlobalAlloc(GHND, sizeof(TW_ONEVALUE));
pval = (pTW_ONEVALUE) GlobalLock(twCapability.hContainer);
pval->ItemType = TWTY_INT16;
pval->Item = 1; //This app will only accept 1 image
GlobalUnlock(twCapability.hContainer);
#endif
#ifdef _MAC_
twCapability.hContainer = (TW_HANDLE)h =
NewHandle(sizeof(TW_ONEVALUE));
((TW_ONEVALUE*)(*h))->ItemType = TWTY_INT16;
```

**Value: Description:**

1 Application wants to receive a single image.
greater than 1 Application wants to receive this specific number of images.
-1 Application can accept any arbitrary number of images during the session. This is the default for this capability.
0 This value has no legitimate meaning and the application should not set the capability to this value. If a Source receives this value during a MSG_SET operation, it should maintain the Current Value without change and return TWRC_FAILURE and TWCC_BADVALUE.

```
count = 1; //This app will only accept 1 image
pInt16 = ((TW_ONEVALUE*)(*h))->Item;
*pInt16 = count;
#endif
//-----Set the CAP_XFERCOUNT
rc = (*pDSM_Entry) (&AppID,
&SourceID,
DG_CONTROL,
DAT_CAPABILITY,
MSG_SET,
(TW_MEMREF)&twCapability);
#ifdef _MSWIN_
GlobalFree((HANDLE)twContainer.hContainer);
#endif
#ifdef _MAC_
DisposHandle((HANDLE)twContainer.hContainer);
#endif
//-----Check Return Codes
//SUCCESS
if (rc == TWRC_SUCCESS)
//the value was set
//APPROXIMATION MADE
else if (rc == TWRC_CHECKSTATUS)
{
//The value could not be matched exactly
//MSG_GET to get the new current value
twCapability.Cap = CAP_XFERCOUNT;
twCapability.ConType = TWON_DONTCARE16; //Source will specify
twCapability.hContainer = NULL; //Source allocates and fills
container
rc = (*pDSM_Entry) (&AppID,
&SourceID,
DG_CONTROL,
DAT_CAPABILITY,
MSG_GET,
(TW_MEMREF)&twCapability);
//remember current value
#ifdef _MSWIN_
pval = (pTW_ONEVALUE) GlobalLock(twCapability.hContainer);
count = pval->Item;
//free hContainer allocated by Source
GlobalFree((HANDLE)twCapability.hContainer);
#endif
#ifdef _MAC_
pInt16 = ((TW_ONEVALUE*)(*h))->Item;
count = *pInt16;
```



```
//free hContainer allocated by Source
DisposeHandle((HANDLE)twCapability.hContainer);
#endif
}
//MSG_SET FAILED
else if (rc == TWRC_FAILURE)
{
//check Condition Code
```

```
rc = (*pDSM_Entry) (&AppID,
&SourceID,
DG_CONTROL,
DAT_STATUS,
MSG_GET,
(TW_MEMREF)&twStatus);
switch (twStatus.ConditionCode)
{
TWCC_BADCAP:
TWCC_CAPUNSUPPORTED:
TWCC_CAPBADOPERATION:
TWCC_CAPSEQERROR:
//Source does not support setting this cap
//All Sources must support CAP_XFERCOUNT
break;
TWCC_BADDEST:
//The Source specified by pSourceID is not open
break;
TWCC_BADVALUE:
//The value set was out of range for this Source
//Use MSG_GET to determine what setting was made
//See the TWRC_CHECKSTATUS case handled earlier
break;
TWCC_SEQERROR:
//Operation invoked in invalid state
break;
}
}
```

**Page ??, (3-28)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### State 4 to 5 - Request the Acquisition of Data from the Source

The Source device is open and capabilities have been negotiated. The application now enables the Source so it can show its user interface, if requested, and prepare to acquire data.

**One Operation is Used:**

**DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS**

**pOrigin**

Points to the application's TW_IDENTITY structure.

**pDest**

Points to the Source's TW_IDENTITY structure.

pData

Points to a structure of type TW_USERINTERFACE.

The definition of TW_USERINTERFACE is:

```
typedef struct {
TW_BOOL ShowUI;
TW_BOOL ModalUI;
TW_HANDLE hParent;
} TW_USERINTERFACE, FAR *pTW_USERINTERFACE;
```

Set the ShowUI field to TRUE if you want the Source to display its user interface. Otherwise, set to FALSE.

The Application will set the ModalUI field to TRUE if it wants the Source to run modal, and

FALSE if it wants the Source to run modeless. Please note that to successfully run modal, it may be necessary for the application to disable inputs to its windows while the Source's GUI is running.
The application sets the hParent field differently depending on the platform on which the application runs.
• **On Windows** - The application should place a handle to the Window that is acting as the Source's parent.
• **On Macintosh** - The application sets this field to NULL.
In response to the user choosing the application's Acquire menu option, the application sends this operation to the Source to enable it. The application typically requests that the Source display the Source's user interface to assist the user in acquiring data. If the Source is told to display its user interface, it will display it when it receives the operation triplet. Modal and Modeless interfaces are discussed in Chapter 4, "Advanced Application Implementation" and Chapter 5, "Source Implementation." Sources **must** check the ShowUI field and return an error if they cannot support the specified mode. In other words it is unacceptable for a source to ignore a ShowUI = FALSE request and still activate its user interface. The application may

develop its own user interface instead of using the Source's. This is discussed in Advanced Application Implementation."
**Note:** Once the Source is enabled via the DG_CONTROL / DAT_USERINTERFACE/ MSG_ENABLEDS operation, all events that enter the application's main event loop must be immediately forwarded to the Source. The explanation for this was given earlier in this chapter when you were instructed to modify the event loop in preparation for a TWAIN session.

**Page ??, (3-29)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### State 5 to 6 - Recognize that the Data Transfer is Ready

The Source is now working with the user to arrange the transfer of the desired data. Unlike all the earlier transitions, the Source, not the application, controls the transition from State 5 to State 6.

#### No Operations (from the application) are Used:

This transition is not triggered by the application sending an operation. The Source causes the transition.
Remember while the Source is enabled, the application is forwarding all events in its event loop to the Source by using the DG_CONTROL /DAT_EVENT / MSG_PROCESSEVENT operation. The TW_EVENT data structure associated with this operation looks like this:

```
typedef struct {
TW_MEMREF pEvent; /*Windows pMSG or MAC pEvent */
TW_UINT16 TWMessage;/*TW message from the Source to the
application*/
} TW_EVENT, FAR *pTW_EVENT;
```

The Source can set the TWMessage field to signal when the Source is ready to transfer data. Following each DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT operation, the application must check the TWMessage field. If it contains MSG_XFERREADY, the session is in State 6 and the Source will wait for the application to request the actual transfer of data.

**Page ??, (3-32)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### State 6 to 7 - Start and Perform the Transfer

The Source indicated it is ready to transfer data. It is waiting for the application to inquire about the image details, initiate the actual transfer, and, hence, transition the session from State 6 to 7. If the initiation (DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET) fails, the session does not transition to State 7 but remains in State 6.

#### Two Operations are Used:

DG_IMAGE / DAT_IMAGEINFO / MSG_GET

**pOrigin**

Points to the application's TW_IDENTITY structure.

**pDest**

Points to the Source's TW_IDENTITY structure.



pData

Points to a structure of type TW_IMAGEINFO. The definition of TW_IMAGEINFO is:

```
typedef struct {
TW_FIX32 XResolution;
TW_FIX32 YResolution;
TW_INT32 ImageWidth;
TW_INT32 ImageLength;
TW_INT16 SamplesPerPixel;
TW_INT16 BitsPerSample[8];
TW_INT16 BitsPerPixel;
TW_BOOL Planar;
TW_INT16 PixelType;
TW_UINT32 Compression;
} TW_IMAGEINFO, FAR *pTW_IMAGEINFO;
```

The Source will fill in information about the image that is to be transferred. The application uses this operation to get the information regardless of which transfer mode (Native, Disk File, or Buffered Memory) will be used to transfer the data.

**DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET**

**pOrigin**

Points to the application's TW_IDENTITY structure.

**pDest**

Points to the Source's TW_IDENTITY structure.

**pData**

Points to a TW_UINT32 variable. This is an exception from the typical pattern.

- **On Windows:** This is a pointer to a handle variable. For 16 bit Microsoft Windows, the handle is stored in the low word of the 32-bit integer and the upper word is set to zero. If running under the WIN32 environment, this is a 32 bit window handle. The Source will set pHandle to point to a device-independent bitmap (DIB) that it allocates.
- **On Macintosh:** This is a pointer to a PicHandle. The Source will set pHandle to point to a PicHandle that the Source allocates.

In either case, the application is responsible for deallocating the memory block holding the Native-format image.

The application may want to inquire about the image data that it will be receiving. The DG_IMAGE / DAT_IMAGEINFO / MSG_GET operation allows this. Other operations, such as

DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET, provide additional information. This information can be used to determine if the application actually wants to initiate the transfer. To actually transfer the data in the Native mode, the application invokes the DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET operation. The Native mode is the default transfer mode and will be used unless a different mode was negotiated via capabilities in State 4. For the Native mode transfer, the application only invokes this operation once per image. The Source returns the TWRC_XFERDONE value when the transfer is complete. This type of transfer cannot be aborted by the application once initiated. (Whether it can be aborted from the Source's User Interface depends on the Source.) Use of the other transfer modes, Disk File and Buffered Memory, are discussed in Chapter 4, "Advanced Application Implementation."

*TWAIN 2.1 Specification 3-31*

The following code illustrates how to get information about the image that will be transferred and how to actually perform the transfer. This code segment is continued in the next section (State 7 to 6 to 5).

```
// After receiving MSG_XFERREADY
TW_UINT16 TransferNativeImage()
{
TW_IMAGEINFO twImageInfo;
TW_UINT16 rc;
TW_UINT32 hBitmap;
TW_BOOL PendingXfers = TRUE;
while (PendingXfers)
{
rc = (*pDSM_Entry)(&AppId,
&SourceId,
DG_IMAGE,
DAT_IMAGEINFO,
MSG_GET,
(TW_MEMREF)&twImageInfo);
if (rc == TWRC_SUCCESS)
Examine the image information
// Transfer the image natively
hBitmap = NULL;
rc = (*pDSM_Entry)(&AppId,
SourceId,
DG_IMAGE,
DAT_IMAGENATIVEXFER,
MSG_GET,
(TW_MEMREF)&HbITMAP);
// Check the return code
switch(rc)
{
case TWRC_XFERDONE:
// Notes: hBitmap points to a valid image Native image (DIB or
// PICT)
// The application is now responsible for deallocating the
memory.
// The source is currently in state 7.
// The application must now acknowledge the end of the transfer,
// determine if other transfers are pending and shut down the
data
// source.
PendingXfers = DoEndXfer(); //Function found in
code
//example in next section
break;
```

```
case TWRC_CANCEL:
// The user canceled the transfer.
// hBitmap is an invalid handle but memory was allocated.
// Application is responsible for deallocating the memory.
```

```
// The source is still in state 7.
// The application must check for pending transfers and shut down
// the data source.
PendingXfers = DoEndXfer(); //Function found in code
//example in next section
break;
case TWRC_FAILURE:
// The transfer failed for some reason.
// hBitmap is invalid and no memory was allocated.
// Condition code will contain more information as to the cause of
// the failure.
// The state transition failed, the source is in state 6.
// The image data is still pending.
// The application should abort the transfer.
DoAbortXfer(MSG_RESET); //Function in next section
PendingXfers = FALSE;
break;
}
}
}
//Check the return code
switch (rc)
{
case TWRC_XFERDONE:
//hBitMap points to a valid Native Image (DIB or PICT)
//The application is responsible for deallocating the memory
//The source is in State 7
//Acknowledge the end of the transfer
goto LABEL_DO_ENDXFER //found in next section
break;
case TWRC_CANCEL:
//The user canceled the transfer
//hBitMap is invalid
//The source is in State 7
//Acknowledge the end of the transfer
goto LABEL_DO_ENDXFER //found in next section
break;
case TWRC_FAILURE:
//The transfer failed
//hBitMap is invalid and no memory was allocated
//Check Condition Code for more information
//The state transition failed, the source is in State 6
//The image data is still pending
//To abort the transfer
goto LABEL_DO_ENDXFER //found in code example for
//the next section
```

```
break;
}
```

**Page ??, (3-37)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

**pData**
Typically, you would expect to see this point to a structure of type TW_PARENT but this is not the case. This is an exception to the usual situation where the DAT field of the triplet identifies the data structure for pData.
**On Windows:** pData points to the window handle (hWnd) that acted as the Source's "parent". The variable is of type TW_INT32. For 16 bit Microsoft Windows, the handle is stored in the low word of the 32 bit integer and the upper word is set to zero. If running under the WIN32 environment, this is a 32 bit window handle.
**On Macintosh:** pData should be a 32-bit NULL value.

To Move from State 2 to State 1
*Chapter 3*
*3-38 TWAIN 2.1 Specification*
Once the Source Manager has been closed, the application must unload the DLL (on Windows) from memory before continuing.
**On Windows:**
Use FreeLibrary( hDSMLib); where hDSMLib is the handle to the Source Manager DLL returned from the call to LoadLibrary( ) seen earlier (in the State 1 to 2 section).
**On Macintosh:**
No action is necessary.

**Page ??, (4-17)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

This indicates the transfer was completed and the session is in State 7. However, it does not guarantee that the Source did not clip the image to make it fit. Even if the application issued a DG_IMAGE / DAT_IMAGEINFO / MSG_GET operation prior to the transfer to determine the image size, it cannot assume that the ImageWidth and ImageLength values returned from that operation really apply to the image that was ultimately transferred. If the dimensions of the image are important to the application, the application should always check the actual transferred image size after the transfer is completed. To do this:
1. Execute a DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER operation to move the session from State 7 to State 6 (or 5).
2. Determine the actual size of the image that was transferred:
a. **On Windows** - Read the DIB header
b. **On Macintosh** - Check the picFrame in the Picture

**Page ??, (4-18)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Disk File Mode Transfer

Beginning with version 1.9, there are now two file transfer mechanisms available. Windows

developers may continue to use the TWSX_FILE option. Macintosh developers must use TWSX_FILE2, instead of TWSX_FILE, in order to correctly address image and audio files in the newer versions of the operating system.
Determine if a Source Supports the Disk File Mode
• Use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation.
• Set the TW_CAPABILITY's Cap field to ICAP_XFERMECH.
• The Source returns information about the transfer modes it supports in the container structure pointed to by the hContainer field of the TW_CAPABILITY structure. The disk file mode is identified as TWSX_FILE or TWSX_FILE2. Sources are not required to support Disk File Transfer so it is important to verify its support.

**After Verifying Disk File Transfer is Supported, Set Up the Transfer**

**During State 4:**
• Set the ICAP_XFERMECH to TWSX_FILE or TWSX_FILE2. Use the DG_CONTROL / DAT_CAPABILITY / MSG_SET operation.
• Use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation to determine which file formats the Source can support. Set TW_CAPABILITY. Cap to ICAP_IMAGEFILEFORMAT and execute the MSG_GET. The Source returns the supported format identifiers which start with TWFF_ and may include TWFF_PICT, TWFF_BMP, TWFF_TIFF, etc. They are listed in the TWAIN.H file and in the Constants section of Chapter 8, "Data Types and Data Structures."

**During States 4, 5, or 6:**
To set up the transfer the DG_CONTROL / DAT_SETUPFILEXFER operation of MSG_GET, MSG_GETDEFAULT, and MSG_SET can be used.
The data structure used in the DSM_Entry call is a TW_SETUPFILEXFER structure (for DAT_SETUPFILEXFER, on Windows and pre-1.9 Macintosh Sources and Applications):

```
typedef struct {
TW_STR255 FileName; /* File to contain data */
TW_UINT16 Format; /* A TWFF_xxxx constant */
TW_HANDLE VrefNum; /* Used for Macintosh only */
} TW_SETUPFILEXFER, FAR *pTW_SETUPFILEXFER;
```

**Page ??, (5-1)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

Mac OS X includes two high-level native development environments that you can use for your application's graphical user interface: Carbon, and Cocoa. These environments are full-featured development environments in their own right, and you can write TWAIN applications and TWAIN Data Sources in any one of these environments.
Because both Carbon and Cocoa change the event handling mechanism (no WaitNextEvent loops), these paragraphs update and extend the section of the previous specification that describes how to modify the application event loop to support TWAIN.
Carbon and Cocoa-based Mac OS X TWAIN applications are required to supply an event handler callback function that the TWAIN DSM will call. Carbon applications using the Classic Event Manager (WaitNextEvent) should continue to route all events through the Data Source. However, Data Sources on Mac OS X can no longer use the Classic Event Manager.

**Page ??, (5-2)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

# The Structure of a Source

The following sections describe the structure of a source.

### On Windows

#### Implementation

The Source is implemented as a Dynamic Link Library (DLL). Sources should link to TWAIN.LIB at link time. The Source will not run stand-alone. The DLL typically runs within the (first) calling application's heap although DLLs may be able to allocate their own heap and stack space. There is only one copy of the DLL's code and data loaded at run-time, even if more than one application accesses the Source. For more information regarding DLLs on Win32s, Windows95, and Windows NT please refer to Microsoft documents.

#### Naming and Location

The DLL's file name must end with a .DS extension. The Source Manager recursively searches for your Source in the TWAIN sub-directory of the Windows directory (typically C:\WINDOWS on Windows 95/98, or C:\WINNT on Windows NT). To reduce the chance for naming collisions, each Source should create a sub-directory beneath TWAIN, giving it a name relevant to their product. The Source DLLs are placed there. Supporting files may be placed there as well, but since this is a system directory which may only be modifiable by the System Administrator, Sources must not write any information into this directory after the installation.

#### Entry Points and Segment Attributes

• Every Source is required to have a single entry point called DS_Entry (see Chapter 6, "Entry Points and Triplet Components"). For 16-bit sources only, in order to speed up the Source Manager's ability to identify Sources, the Source entry point DS_Entry( ) and the code to respond to the DG_CONTROL / DAT_IDENTITY / MSG_GET operation must reside in a segment marked as PRELOAD. All other segments should be marked as LOADONCALL (with the exception of any interrupt handler that may exist in the Source which needs to be in a FIXED code segment).

#### Resources

• **Version Information** - The Microsoft VER.DLL is included with the TWAIN toolkit for use by your installation program, if you have one, to validate the version of the currently installed Source Manager. Sources should be marked with the Version information capability defined in Microsoft Windows 3.1. To do this, you can use the resource compiler from the version 3.1 SDK. VER.DLL and the version stamping are also compatible with Microsoft Windows version 3.0.

• **Icon Id** - Future versions of the TWAIN Source Manager may display the list of available Sources using a combination of the ProductName (in the Source's TW_IDENTITY structure) and an Icon (as the Macintosh version currently does). Therefore, it is recommended that you add this icon into your Source resource file today. This will allow your Source to be immediately compatible with any upcoming changes. The icon should be identified using TWON_ICONID from the TWAIN.H file.



#### General Notes

• **GlobalNotify -** Microsoft Windows allows only one GlobalNotify handler per task. As the Source resides in the application heap, the Source should **not** use the GMEM_NOTIFY flag on the memory blocks allocated as this may disrupt the correct behavior of the application that uses GlobalNotify.

• **Windows Exit Procedure (WEP) -** During the WEP, the Source is being unloaded by

Microsoft Windows. The Source should make sure all the resources it allocated and owns get released whether or not the Source was terminated properly.

### On Macintosh

#### Implementation

A Source on a Macintosh is implemented as a bundle. The Source will not run standalone. A separate copy of the Source's code will be made for each application that opens the Source.

#### Naming and Location

The extension for a Source is ds. The Source Manager will search for bundles with this extension in the /Library/Image Capture/TWAIN Data Sources/ folder. It is recommended that each Source bundle contains any other files it may require.

#### Compatibility with Older Data Sources

Pre Mac OS X Data Source are not compatible with the TWAIN implementation on Mac OS X.

**Page ??, (5-5)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

# Sources and the Event Loop

### Handling Events

On Windows, when a Source is enabled (i.e. States 5, 6, and 7), the application must pass all events (messages) to the Source. Since the Source runs subservient to the application, this ensures that the Source will receive all events for its window. The event will be passed in the TW_EVENT data structure that is referenced by a DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT command.

On Mac OS X, the Data source either uses Carbon or Cocoa. A Carbon based Data Source has to install Carbon Event Handler for all UI elements. A Cocoa Data Source gets the UI event handling for free.

**Note:** Starting with TWAIN 1.8, it is now possible for events to be managed in State 4 only to support CAP_DEVICEEVENTS. This is a fundamental change from previous TWAIN behavior that has been added to allow the Source to notify the Application of important changes in the state of the Source even while in State 4. Note also that the default value for CAP_DEVICEEVENTS (if supported) must be an empty TW_ARRAY, indicating the



event reporting is turned off. This is essential to allow backward compatibility with pre-1.8 Applications.

Routing all messages to all connected Sources while they are enabled places a burden on the application and creates a potential performance bottleneck. Therefore, the Source must process the incoming events as quickly as possible. The Source should examine each incoming operation before doing anything else. Only one operation's message field says MSG_PROCESSEVENT so always look at the message field first. If it indicates MSG_PROCESSEVENT then:

**Immediately** determine if the event belongs to the Source.

**If it does**

Set the Return Code for the operation to TWRC_DSEVENT

Set the TWMessage field to MSG_NULL

Process the event

**Else**

Set the Return Code to TWRC_NOTDSEVENT

Set the TWMessage field to MSG_NULL

Return to the application immediately

If the Source developer fails to process events with this high priority, the user may see degraded performance whenever the Source is frontmost which reflects poorly on the Source.

On Windows, the code fragment looks like the following:

```
TW_UINT16 CALLBACK DS_Entry(pTW_IDENTITY pSrc,
TW_UINT32 DG,
TW_UINT16 DAT,
TW_UINT16 MSG,
TW_MEMREF pData)
{
TWMSG twMsg;
TW_UINT16 twRc;
//Valid states 5 – 7 (or 4 – 7 if CAP_DEVICEEVENTS has been
// negotiated to anything other than its default value of an empty
// TW_ARRAY). As soon as the application has enabled the
// Source it must being sending the Source events. This allows the
// Source to receive events to update its user interface and to
// return messages to the application. The app sends down ALL
// message, the Source decides which ones apply to it.
if (MSG == MSG_PROCESSEVENT)
```



```
{
if (hImageDlg && IsDialogMessage(hImageDlg,
(LPMSG)(((pTW_EVENT)pData)->pEvent)))
{
twRc = TWRC_DSEVENT;
// The source should, for proper form, return a MSG_NULL for
// all Windows messages processed by the Data Source
((pTW_EVENT)pData)->TWMessage = MSG_NULL;
}
else
{
// notify the application that the source did not
// consume this message
twRc = TWRC_NOTDSEVENT;
((pTW_EVENT)pData)->TWMessage = MSG_NULL;
}
}
else
{
// This is a Twain message, process accordingly.
// The remainder of the Source's code follows...
}
return twRc;
}
```

The Windows IsDialogMessage( ) call is used in this example. Sources can also use other Windows calls such as TranslateAccelerator( ) and TranslateMDISYSAccel( ).

**Page ??, (5-6)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

These notices are sent differently on Windows versus Macintosh systems.

**On Windows**

The Source creates a call to DSM_Entry (the entry point in the Source Manager) and fills the

destination with the TW_IDENTITY structure of the application. The Source uses one of the following triplets:
DG_CONTROL / DAT_NULL / MSG_XFERREADY
DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ
The Source Manager, on Windows, recognizes the notice and makes sure the message is received correctly by the application.

**On Macintosh**

Please refer to the callback mechanism described in Chapter 3, "Application Implementation."

**Page ??, (5-8)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Modal versus Modeless Interfaces

As stated in Chapter 4, "Advanced Application Implementation", the Source's user interface may be modal or modeless. The modeless approach gives the user more control and is recommended whenever practical. Refer to the information following this table for specifics about Windows and Macintosh implementation.

**Page ??, (5-9)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Implementing Modal and Modeless User Interfaces

#### On Windows

You cannot use the modal dialog creation call DialogBox( ) to create the Source's user interface main window. To allow event processing by both the application and the Source, this call cannot be used. Modal user interfaces in Source are not inherently bad, however. If a modal user interface makes sense for your Source, use either the CreateDialog( ) or CreateWindow( ) call.

**Modal (App Modal)**

It is recommended that the Source's main user interface window be created with a modeless mechanism. Source writers can still decide to make their user interface behave modally if they choose. It is even appropriate for a very simple "click and go" interface to be implemented this way.

This is done by first specifying the application's window handle (hWndParent) as the parent window when creating the Source's dialog/window and second by enabling/disabling the parent window during the MSG_ENABLEDS / MSG_DISABLEDS operations. Use EnableWindow(hWndParent, FALSE) to disable the application window and EnableWindow(hWndParent, TRUE) to re-enable it.

**Modeless**

If implementing a modeless user interface, specify NULL as the parent window handle when creating the Source's dialog/window. Also, it is suggested that you call BringWindowToTop( ) whenever a second request is made by the same application or another application requesting access to a Source that supports multiple application connections.

#### On Macintosh

It is recommended that the Source's main user interface window be created with a modeless mechanism. Source writers can still decide to make their user interface behave modally if they choose. It is even appropriate for a very simple "click and go" interface to be implemented this way.

**Page ??, (5-12)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Transfer Modes

All Sources must support Native and Buffered Memory data transfers. It is strongly suggested that they support Disk File mode, too. The default mode is Native. To select one of the other modes, the application must negotiate the ICAP_XFERMECH capability (whose default is TWSX_NATIVE). Sources must support negotiation of this capability. The native format for Microsoft Windows is DIB. For Macintosh, the native format is a PICT. The version of PICT to be transferred is the latest version available on the machine on which the application is running (usually PICT II for machines running 32-bit/color QuickDraw and PICT I for machines running black and white QuickDraw).

**Page ??, (5-12)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Native Mode Transfers

On Native mode transfers, the data parameter in the DSM_Entry call is a pointer to a variable of type TW_UINT32.

*TWAIN 2.1 Specification 5-13*

#### On Windows

The low word of this 32-bit integer is a handle variable to a DIB (Device Independent Bitmap) located in memory.

#### On Macintosh

This 32-bit integer is a handle to a Picture (a PicHandle). It is a Quick Draw picture located in memory.

**Page ??, (5-16)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### DAT_EVENT Handling Errors

One of the most common problems between a data source and application is the management of DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT. The symptoms are not immediately obvious, so it is worth mentioning them to assist new developers in quickly identifying and solving the problem.

**Cannot use TAB or Keyboard Shortcuts to Navigate TWAIN Dialog**

The cause of this can be one of two things. Either the application is not forwarding all messages to TWAIN through the DAT_EVENT mechanism, or the data source is not properly processing the DAT_EVENT messages. (Windows: calling IsDialogMessage for each forwarded message with TWAIN Dialog handle)

**TWAIN Dialog Box Combo Boxes cannot be opened, Edit boxes produce multiple chars per keystroke**

This case is caused by processing TWAIN Dialog Messages twice. Either the data source has not returned the proper return code in response to DAT_EVENT calls (Windows: TWRC_DSEVENT when IsDialogMessage returns TRUE), or the application is ignoring the return code.

**This is not a problem when data source operates through TWAIN Thunker**

Problems with the application handling of these messages are not often detected if the data source is operating through the TWAIN Thunking mechanism. This is because the Thunker process has a separate Window and Message pump that properly dispatch DAT_EVENT messages to the data source. Any mistake in application handling will pass without notice since all DAT_EVENT calls will return TWRC_NOTDSEVENT. (with the exception of important messages such as MSG_XFERREADY.)

**Problem seems erratic, keyboard shortcuts and Tab key work for Message Boxes, but not TWAIN Dialog**

This observation often further confuses the issue. In Windows, a standard Message box is Modal, and operates from a local message pump until the user closes it. All messages are properly dispatched to the message box since it does not rely on the application message pump. The TWAIN Dialog is slightly different since it is implemented Modeless. There is no easy way to duplicate Modal behavior for the TWAIN Dialog.

**Page ??, (5-17)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

# Memory Management

### Windows Specifics

On 16-bit Windows systems, a single copy of the Source Manager and Source(s) services all applications wishing to access TWAIN functionality. If the Source can connect to more than one application, it will probably need to maintain a separate execution frame for each connected application. The Source does not have unlimited memory available to it so be conservative in its use.

On 32-bit Windows systems, a new in-memory copy of the Source Manager and Source(s) is

created in the Application's calling space. In addition, a call may be made to the Windows On Windows (WOW) system, to support the thunking mechanism. For more information on the thunker, refer to Chapter 3, "Application Implementation."
It is valid for an application to open a Source and leave it open between several acquires. Therefore, Sources should minimize the time and resources required to load and remain open (in State 4). Also, Sources should allow a reasonable number of connections to ensure they can handle more than one application using the Source in this manner (leaving it open between acquires).

### Macintosh Specifics

Each application that loads the Source Manager has a private copy of the Source. Each Source that is connected also runs as a private copy. It is important for the Source writer to recognize that their Source will be using the memory heap of the host application, not in its own heap. Therefore, the Source should be conscientious with allocation and deallocation of memory.

**Page ??, (6-1)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Programming Basics

• Upon entry, the parameters must be ordered on the stack in Pascal form. Be sure that your code expects this ordering rather than the reverse order that C uses.
• The keyword FAR is included in the entry point syntax to accommodate the 16-bit Windows segmented addressing scheme. It has no value for any other operating system, and is defined as an empty value for everything, except 16-bit Windows.

**Page ??, (6-2)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Declaration of DSM_Entry( )

Written in C code form, the declaration looks like this:

#### On Windows

```
TW_UINT16 FAR PASCAL DSM_Entry
( pTW_IDENTITY pOrigin, // source of message
pTW_IDENTITY pDest, // destination of message
TW_UINT32 DG, // data group ID: DG_xxxx
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
TW_MEMREF pData // pointer to data
);
```

#### On Macintosh

```
FAR PASCAL TW_UINT16 DSM_Entry
( pTW_IDENTITY pOrigin, // source of message
pTW_IDENTITY pDest, // destination of message
TW_UINT32 DG, // data group ID: DG_xxxx
```

```
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
TW_MEMREF pData // pointer to data
);
```

**Page ??, (6-3)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

### Declaration of DS_Entry( )

DS_Entry is only called by the Source Manager. Written in C code form, the declaration looks like this:

#### On Windows

```
TW_UINT16 FAR PASCAL DS_Entry
( pTW_IDENTITY pOrigin, // source of message
TW_UINT32 DG, // data group ID: DG_xxxx
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
TW_MEMREF pData // pointer to data
);
```

#### On Macintosh

```
FAR PASCAL TW_UINT16 DS_Entry
( pTW_IDENTITY pOrigin, // source of message
TW_UINT32 DG, // data group ID: DG_xxxx
TW_UINT16 DAT, // data argument type: DAT_xxxx
TW_UINT16 MSG, // message ID: MSG_xxxx
```



```
TW_MEMREF pData // pointer to data
);
```

**Page ??, (6-5)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

| | | |
|---|---|---|
| DAT_AUDIONATIVEXFER | DG_AUDIO | TW_UINT32<br>On Windows - low word = WAV handle<br>On Macintosh - audio handle |
| DAT_PARENT | DG_CONTROL | TW_INT32<br>On Windows - low word=Window handle<br>On Macintosh - Set to NULL |

**Page ??, (6-6)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

| | | |
|---|---|---|
| DAT_IMAGENATIVEXFER | DG_IMAGE | TW_UINT32;<br>On Windows - low word=DIB handle<br>On Macintosh – PicHandle |

**Page ??, (7-9)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_AUDIO / DAT_AUDIONATIVEXFER / MSG_GET

### Call

DSM_Entry (pOrigin, pDest, DG_AUDIO, DAT_AUDIONATIVEXFER, MSG_GET, pHandle);

pHandle = A pointer to a variable of type TW_UINT32

**On Windows** - This 32 bit integer is a handle variable to WAV data located in memory.

**On Macintosh** - This 32-bit integer is a handle to AIFF data

**Page ??, (7-10)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_CALLBACK / MSG_INVOKE_CALLBACK

### Call

DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_CALLBACK, MSG_INVOKE_CALLBACK, (TW_MEMREF)&callback);

### Valid States

4, 5, 7 (depending on the message)

### Description

This triplet is sent by the DS to the DSM, which in turn calls the application's registered callback function. The last argument is a pointer to an initialized TW_CALLBACK structure, which contains the message to be processed.

The TW_CALLBACK structure should be initialized as follows:

Msg Initialized to any valid DG_CONTROL / DAT_NULL message.

The message specified will be processed in the same manner as the DAT_NULL mechanism employed by the Windows version. These are:

MSG_XFERREADY

MSG_CLOSEDSREQ

MSG_CLOSEDSOK

MSG_DEVICEEVENT

MSG_INVOKE_CALLBACK is the **only** way for a Mac OS X DS to inform the application of these events.

### Return Codes

TWRC_FAILURE

### See Also

DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK

**Page ??, (7-41)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL/ DAT_EVENT / MSG_PROCESSEVENT

### Call

DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_EVENT, MSG_PROCESSEVENT, pEvent);

pEvent = A pointer to a TW_EVENT structure.

### Valid States

5 through 7

### Description

This operation supports the distribution of events from the application to Sources so that the Source can maintain its user interface and return messages to the application. Once the application has enabled the Source, it **must immediately** begin sending to the Source all events that enter the application's main event loop. This allows the Source to update its user interface in real-time and to return messages to the application which cause state transitions. Even if the application overrides the Source's user interface, it must forward all events once the Source has been enabled. The Source will tell the application whether or not each event belongs to the Source.

**Note:** Events only need to be forwarded to the Source while it is enabled.

The Source should be structured such that identification of the event's "owner" is handled before doing anything else. Further, the Source should return **immediately** if the Source isn't the owner. This convention should minimize performance concerns for the application (remember, these events are only sent while a Source is enabled—that is, just before and just after the transfer is taking place).

### Application

On Windows: Make pEvent->pEvent point to the message structure.

On Macintosh: SeeChapter 3, "Application Implementation."

**Note:** On return, the application should check the Return Code from DSM_Entry() for TWRC_DSEVENT or TWRC_NOTDSEVENT. If TWRC_DSEVENT is returned, the application should not process the event—it was consumed by the Source. If TWRC_NOTDSEVENT is returned, the application should process the event as it normally would.

With either of these Return Codes, the application should also check the pEvent->TWMessage and switch on the result. This is the mechanism used by the Source to notify the application that a data transfer is ready or that it should close the Source. The Source can return one of the following messages:

```
MSG_XFERREADY /* Source has one or more images */
/* ready to transfer */
```



```
MSG_CLOSEDSREQ /* Source wants to be closed, */
/* usually initiated by a */
/* user-generated event */
MSG_NULL /* no message for application */
```

### Source

Process this operation **immediately** and return to the application **immediately** if the event doesn't belong to you. Be aware that the application will be sending *thousands* of messages to you. Consider in-line processing and global flags to speed implementation.

### Return Codes

```
TWRC_DSEVENT /* Source consumed event--application */
/* should not process it */
```

TWRC_NOTDSEVENT /* Event belongs to application - */
/* process as usual */
TWRC_FAILURE
TWCC_BADDEST /* No such Source in-session */
/* with application */
TWCC_SEQERROR /* Operation invoked in invalid */
/* state */

**See Also**

DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ
DG_CONTROL / DAT_NULL / MSG_XFERREADY
Event loop information (in Chapter 7, "Operation Triplets.")

**Page ??, (7-59)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS (from Application to Source Manager)

**Call**

DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_CLOSEDS, pSourceIdentity);
pSourceIdentity = A pointer to a TW_IDENTITY structure.

**Valid States**

**4 only** (Transitions to State 3, if successful)

**Description**

When an application is finished with a Source, it must formally close the session between them using this operation. This is necessary in case the Source only supports connection with a single application (many desktop scanners will behave this way). A Source such as this cannot be accessed by other applications until its current session is terminated.

**Application**

Reference pSourceIdentity to the application's copy of the TW_IDENTITY structure for the Source whose session is to be ended. The application needs to unload the Source from memory after it is closed. The process for unloading the Source is similar to that used to unload the Source Manager.

**Source Manager**

**On Macintosh only**—Closes the Source and removes it from memory, following receipt of TWRC_SUCCESS from the Source.
**On Windows only**—Checks its internal counter to see whether any other applications are accessing the specified Source. If so, the Source Manager takes no other action. If the closing application is the last to be accessing this Source, the Source Manager closes the Source (forwards this triplet to it) and removes it from memory, following receipt of TWRC_SUCCESS from the Source.
Upon receiving the request from the Source Manager, the Source *immediately* prepares to terminate execution.

**Return Codes**

TWRC_SUCCESS
TWRC_FAILURE
TWCC_SEQERROR /* Operation invoked in invalid state */

**See Also**

DG_CONTROL / DAT_IDENTITY / MSG_OPENDS

**Page ??, (7-69)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Source Manager to Source)

### Call

DS_Entry(pOrigin, DG_CONTROL, DAT_IDENTITY, MSG_OPENDS, pSourceIdentity);
pSourceIdentity = A pointer to a TW_IDENTITY structure.

### Valid States

Source is loaded but not yet open (approximately State 3.5, session transitions to State 4, if successful).

### Description

Opens the Source for operation.

### Source Manager

pSourceIdentity is filled in from a previous DG_CONTROL / DAT_IDENTITY / MSG_GET and the Id field should be filled in by the Source Manager.

### Source

Initializes any needed internal structures, performs necessary checks, and loads all resources needed for normal operation.

**Windows only:** Source should record a copy of *pOrigin, the application's TW_IDENTITY structure, whose Id field maintains a unique number identifying the application that is calling. Sources that support only a single connection should examine pOrigin->Id for each operation to verify they are being called by the application they acknowledge being connected with. All requests from other applications should fail (TWRC_FAILURE / TWCC_MAXCONNECTIONS). The Source is responsible for managing this, not the Source Manager (the Source Manager does not know in advance how many connections the Source will support).

**Macintosh Note:** Since the Source(s) and the Source Manager connected to a particular application live within that application's heap space, and are not shared with any other application, the discussion about multiply-connected Sources and verifying which application is invoking an operation is not relevant. A Source or Source Manager on the Macintosh can only be connected to a single application, though multiple copies of a Source or the Source Manager may be active on the same host simultaneously. These instances simply exist in different applications' heaps. If the instances need to communicate with one another, they might use a special file, Gestalt selector, or other IPC mechanism.

### Return Codes

TWRC_SUCCESS
TWRC_FAILURE
*Chapter 7*
*7-70 TWAIN 2.1 Specification*
TWCC_LOWMEMORY /* not enough memory to */
/* open the Source */
TWCC_MAXCONNECTIONS /* Source cannot support */
/* another connection */
TWCC_OPERATIONERROR /* internal Source error;*/
/* handled by the Source */

### See Also

DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS
DG_CONTROL / DAT_IDENTITY / MSG_GET

**Page ??, (7-72)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT

### Call

DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_USERSELECT, pSourceIdentity);

pSourceIdentity = A pointer to a TW_IDENTITY structure.

### Valid States

3 through 7

### Description

This operation should be invoked when the user chooses **Select Source...** from the application's **File** menu (or an equivalent user action). This operation causes the Source Manager to display the Select Source dialog. This dialog allows the user to pick which Source will be used during subsequent **Acquire** operations. The Source selected becomes the system default Source. This default persists until a different Source is selected by the user. The system default Source may be overridden by an application (the override is local to only that application). Only Sources that can supply data matching one or more of the application's SupportedGroups (from the application's identity structure) will be selectable. All others will be unavailable for selection.

### Application

If the application wants a particular Source, other than the system default, to be highlighted in the Select Source dialog, it should set the ProductName field of the structure pointed to by pSourceIdentity to the ProductName of that Source. This information should have been obtained from an earlier operation using DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST, MSG_GETNEXT, or MSG_USERSELECT. Otherwise, the application should set the ProductName field in pSourceIdentity to the null string ("\0"). In either case, the application should set the Id field in pSourceIdentity to zero.

If the Source Manager can't find a Source whose ProductName matches that specified by the application, it will select the system default Source (the default that matches the SupportedGroups of the application). This is not considered to be an error condition. No error will be reported. The application should check the ProductName field of pSourceIdentity following this operation to verify that the Source it wanted was opened.

### Source Manager

The Source Manager displays the Select Source dialog and allows the user to select a Source. When the user clicks the "OK" button ("Select" button in the Microsoft Windows Source Manager) in the Select Source dialog, the system default Source (maintained by the Source Manager) will be changed to the selected Source. This Source's identifying information will be written into pSourceIdentity.

The "Select" button ("OK" button in the Macintosh Source Manager) will be grayed out if there are no Sources available matching the SupportedGroups specified in the application's identity structure, pOrigin. The user must click the "Cancel" button to exit the Select Source dialog. The application cannot discern from this Return Code whether the user simply canceled the



selection or there were no Sources for the user to select. If the application really wants to know whether any Sources are available that match the specified SupportedGroups it can invoke a MSG_GETFIRST operation and check for a successful result.

It copies the TW_IDENTITY structure of the selected Source into pSourceIdentity.

**Suggestion for Source Developers:** The string written in the Source's TW_IDENTITY.ProductName field should clearly and unambiguously identify your product or the Source to the user (if the Source can be used to control more than one device). ProductName contains the string that will be placed in the Select Source dialog (accompanied, on the Macintosh, with an icon from the Source's resource file representing the Source). It is

further suggested that the Source's disk file name approximate the ProductName to assist the user in equating the two.

**Return Codes**

TWRC_SUCCESS
TWRC_CANCEL /* User clicked cancel button - maybe there */
/* were no Sources */
TWRC_FAILURE
TWCC_LOWMEMORY /* not enough memory to perform this */
/* operation */

**See Also**

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT
DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST
DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS

**Page ??, (7-74)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ (from Source to Application - Windows only)

**Call**

DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_CLOSEDSREQ, NULL);
This operation requires no data (NULL).

**Valid States**

5 through 7 (This operation causes the session to transition to State 5.)

**Description**

While the Source is enabled, the application is sending all events/messages to the Source. The Source will use one of these events/messages to indicate to the application that it needs to be closed.

**On Windows**, the Source sends this DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ to the Source Manager to cause the Source Manager to post a private message to the application's event/message loop. This guarantees that the application will have an event/message to pass to the Source Manager so it will be able to communicate the Source's Close request back to the application.

**On Macintosh**, refer to Chapter 3, "Application Implementation."

**Source (on Windows only)**

Source creates this triplet with NULL data and sends it to the Source Manager via the Source Manager's DSM_Entry point.
pDest is the TW_IDENTITY structure of the application.

**Source Manager (on Windows only)**

Upon receiving this triplet, the Source Manager posts a private message to the application's event/message loop. Since the application is forwarding all events/messages to the Source while the Source is enabled, this creates a communication device needed by the Source. When this private message is received by the Source Manager (via the DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT operation), the Source Manager will insert a MSG_CLOSEDSREQ into the TWMessage field on behalf of the Source.

**Return Codes**

TWRC_SUCCESS
TWRC_FAILURE
TWCC_SEQERROR /* Operation invoked in invalid state */

TWCC_BADDEST /* No such application in session with*/
/* Source */
*TWAIN 2.1 Specification 7-75*

**See Also**

DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT
DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS

**Page ??, (7-77)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_NULL / MSG_XFERREADY (from Source to Application - applies to Windows only)

**Call**

DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_XFERREADY, NULL);
This operation requires no data (NULL).

**Valid States**

5 only (This operation causes the transition to State 6.)

**Description**

While the Source is enabled, the application is sending all events/messages to the Source. The Source will use one of these events/ messages to indicate to the application that the data is ready to be transferred.

**On Windows**, the Source sends this DG_CONTROL / DAT_NULL / MSG_XFERREADY to the Source Manager to cause the Source Manager to post a private message to the application's event/message loop. This guarantees that the application will have an event/message to pass to the Source and the Source will be able to communicate its "transfer ready" announcement back to the application.

**On Macintosh**, refer to Chapter 3, "Application Implementation."

**Source (on Windows only)**

Source creates this triplet with NULL data and sends it to the Source Manager via the Source Manager's DSM_Entry point.
pDest is the TW_IDENTITY structure of the application.

**Source Manager**

Upon receiving this triplet, the Source Manager posts a private message to the application's event/message loop. Since the application is forwarding all events/messages to the Source while the Source is enabled, this creates a communication device needed by the Source. When this private message is received by the Source Manager (via the DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT operation), the Source Manager will insert the MSG_XFERREADY into the TWMessage field on behalf of the Source.

**Return Codes**

TWRC_SUCCESS
TWRC_FAILURE
TWCC_SEQERROR /* Operation invoked in invalid state */
TWCC_BADDEST /* No such application in session with*/
/* Source */
*Chapter 7*
*7-78 TWAIN 2.1 Specification*

**See Also**

DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET

DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

**Page ??, (7-79)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM

### Call

DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_PARENT, MSG_CLOSEDSM, pParent);

**On Windows -** pParent = points to the window handle (hWnd) that will act as the Source's "parent". The variable is of type TW_INT32 and the low word of this variable must contain the window handle.

**On Macintosh -** pParent = should be a 32-bit NULL value.

### Valid States

3 **only** (causes transition back to State 2, if successful)

### Description

When the application has closed all the Sources it had previously opened, and is finished with the Source Manager (the application plans to initiate no other TWAIN sessions), it must close the Source Manager. The application should unload the Source Manager DLL or code resource after the Source Manager is closed—unless the application has immediate plans to use the Source Manager again.

### Application

References the same pParent parameter that was used during the "open Source Manager" operation. If the operation returns TWRC_SUCCESS, the application should unload the Source Manager from memory.

### Source Manager

Does any housekeeping needed to prepare for being unloaded from memory. This housekeeping is transparent to the application.

**Windows only**—If the Source Manager is open to at least one other application, it will clean up just activities relative to the closing application, then return TWRC_SUCCESS. The application will attempt to unload the Source Manager DLL. Windows will tell the application that the unload was successful, but the Source Manager will remain active and connected to the other application(s).

### Return Codes

TWRC_SUCCESS
TWRC_FAILURE
TWCC_SEQERROR /* Operation invoked in invalid state */

### See Also

DG_CONTROL / DAT_PARENT / MSG_OPENDSM

**Page ??, (7-80)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_PARENT / MSG_OPENDSM

### Call

DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_PARENT, MSG_OPENDSM, pParent);
**On Windows -** pParent = points to the window handle (hWnd) that will act as the Source's "parent". The variable is of type TW_INT32 and the low word of this variable must contain the window handle.
**On Macintosh -** pParent = should be a 32-bit NULL value.

### Valid States

2 **only** (causes transition to State 3, if successful)

### Description

Causes the Source Manager to initialize itself. This operation **must** be executed before any other operations will be accepted by the Source Manager.

### Application

**Windows only**—The application should set the pParent parameter to point to a window handle (hWnd) of an open window that will remain open until the Source Manager is closed. If application can't open the Source Manager DLL, Windows displays an error box (this error box can be disabled by a prior call to SetErrorMode (SET_NOOPENFILEERRORBOX)).
**Macintosh only**—Set pParent to NULL.

### Source Manager

Initializes and prepares itself for subsequent operations. Maintains a copy of pParent.
**Windows only**—If Source Manager is already open, Source Manager won't reinitialize but will retain a copy of pParent.

### Return Codes

TWRC_SUCCESS
TWRC_FAILURE
TWCC_LOWMEMORY /* not enough memory to perform */
/* this operation */
TWCC_SEQERROR /* Operation invoked in invalid */
/* state */

### See Also

DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM

**Page ??, (7-89)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET

### Call

DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
MSG_GET, pSetupFile);
pSetupFile = A pointer to a TW_SETUPFILEXFER structure

### Valid States

4 through 6

### Description

Returns information about the file into which the Source has or will put the acquired DG_IMAGE or DG_AUDIO data.

### Application

No special set up or action required.

### Source

Set the following:
pSetupFile->Format = format of destination file

(DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
(DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
pSetupFile->FileName = name of file
(on Windows, include the complete path name)
pSetupFile->VRefNum = volume reference number
**(Macintosh only)**

**Return Codes**

TWRC_SUCCESS
TWRC_FAILURE
TWCC_BADDEST /* No such Source in-session with application */
TWCC_BADPROTOCOL /* Source does not support file transfer */
TWCC_SEQERROR /* Operation invoked in invalid state */

**See Also**

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET

DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
Capabilities -ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

**Page ??, (7-91)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT

**Call**

DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER, MSG_GETDEFAULT, pSetupFile);
pSetupFile = A pointer to a TW_SETUPFILEXFER structure

**Valid States**

4 through 6

**Description**

Returns information for the default DG_IMAGE or DG_AUDIO file.

**Application**

No special set up or action required.

**Source**

Set the following:
pSetupFile->Format = format of destination file
(DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
(DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
pSetupFile->FileName = name of file
(on Windows, include the complete path name)
pSetupFile->VRefNum = volume reference number
**(Macintosh only)**

**Return Codes**

TWRC_SUCCESS
TWRC_FAILURE
TWCC_BADDEST /* No such Source in-session with application */
TWCC_BADPROTOCOL /* Source does not support file transfer */

TWCC_SEQERROR /* Operation invoked in invalid state */

**See Also**
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
Capabilities -ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

**Page ??, (7-93)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET

**Call**
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER, MSG_RESET, pSetupFile);
pSetupFile = A pointer to a TW_SETUPFILEXFER structure
**Valid States**
4 only
**Description**
Resets the current file information to the DG_IMAGE or DG_AUDIO default file information and returns that default information..
**Application**
No special set up or action required.
**Source**
Set the following:
pSetupFile->Format = format of destination file
(DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
(DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
pSetupFile->FileName = name of file
(on Windows, include the complete path name)
pSetupFile->VRefNum = volume reference number
**(Macintosh only)**
**Note:** VRefNum should be set to reflect the default file only if it already exists). Otherwise, set this field to NULL.
**Return Codes**
TWRC_SUCCESS
TWRC_FAILURE
TWCC_BADDEST /* No such Source in-session with application */
TWCC_BADPROTOCOL /* Source does not support file transfer */

TWCC_SEQERROR /* Operation invoked in invalid state */
**See Also**
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET

DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
Capabilities -ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT,
ACAP_XFERMECH

**Page ??, (7-95)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET

### Call

DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
MSG_SET, pSetupFile);
pSetupFile = A pointer to a TW_SETUPFILEXFER structure

### Valid States

4 through 6

### Description

Sets the file transfer information for the next file transfer. The application is responsible for verifying that the specified file name is valid and that the file either does not currently exist (in which case, the Source is to create the file), or that the existing file is available for opening and read/write operations. The application should also assure that the file format it is requesting can be provided by the Source (otherwise, the Source will generate a TWRC_FAILURE / TWCC_BADVALUE error).

### Application

Set the following:
pSetupFile->Format = format of destination file
(DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
(DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
pSetupFile->FileName = name of file
(on Windows, include the complete path name)
pSetupFile->VRefNum = volume reference number
**(Macintosh only)**
**Note:** ICAP_XFERMECH or ACAP_XFERMECH (depending on the value of DAT_XFERGROUP) must have been set to TWSXdata) and return TWRC_FAILURE with TWCC_BADVALUE. If the format and file name are OK, but a file error occurs when trying to open the file (other than "file does not existö), return TWCC_BADVALUE and set up to use the default file. If the specified file does not exit, create it. If the file exists and has data in it, overwrite the existing data starting with the first byte of the file.

### Return Codes

TWRC_SUCCESS
TWRC_FAILURE
TWCC_BADDEST /* No such Source in-session with application */
TWCC_BADPROTOCOL /* Source does not support file transfer */

TWCC_BADVALUE /* Source cannot comply with one of the */
/* settings */
TWCC_SEQERROR /* Operation invoked in invalid state */

### See Also

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
Capabilities -ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

**Page ??, (7-105)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

**Windows only**—The application should place a handle (hWnd) to the window acting as the Source's parent into pUserInterface->hParent.
**Macintosh only**—Set pUserInterface->hParent to NULL.

**Page ??, (7-134)**

*Remove. This needs to be moved to the chapter on Operating System Dependencies.*

## DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

### Call

DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGENATIVEXFER, MSG_GET, pHandle);
pHandle = A pointer to a variable of type TW_UINT32.
**Windows -** This 32 bit integer is a handle variable to a DIB (Device Independent Bitmap) located in memory.
**Macintosh -** This 32-bit integer is a handle to a Picture (a PicHandle). It is a QuickDraw picture located in memory.

### Valid States

6 only (Transitions to State 7, if successful. Remains in State 7 until MSG_ENDXFER operation).

### Description

Causes the transfer of an image's data from the Source to the application, via the Native transfer mechanism, to begin. The resulting data is stored in main memory in a single block. The data is stored in Picture (PICT) format on the Macintosh and as a device-independent bitmap (DIB) under Microsoft Windows. The size of the image that can be transferred is limited to the size of the memory block that can be allocated by the Source.
**Note:** This is the default transfer mechanism. All Source's support this mechanism. The Source will use this mechanism unless the application explicitly negotiates a different transfer mechanism with ICAP_XFERMECH.

### Application

The application need only invoke this operation once per image. The Source allocates the largest block available and transfers the image into it. If the image is too large to fit, the Source may resize the image. Read the DIB header or check the picFrame in the Picture to determine if

this happened. The application is responsible for deallocating the memory block holding the Native-format image.

**Windows only**—Set pHandle pointing to a handle to a device-independent bit map (DIB) in memory. The Source will allocate the image buffer and return the handle to the address specified..

**Macintosh only**—Set pHandle pointing to a handle to a Picture in memory. The Source will allocate the image buffer at the memory location referenced by the handle.

**Note:** This odd combination of pointer and handle to reference the image data block was used to assure that the allocated memory object would be relocatable under Microsoft Windows, Macintosh, and UNIX. A handle was required for this task on both the Macintosh and under Microsoft Windows; though pointers are inherently relocatable under UNIX. Rather than disturb the entry points convention that the data object is



always referenced by a pointer, it was decided to have that pointer reference the relocatable handle. A handle in UNIX is typecast to a pointer.

### Source

Allocate a single block of memory to hold the image data and write the image data into it using the appropriate format for the operating environment. The source must assure that the allocated block will be accessible to the application. Place the handle of the allocated block in the TW_UINT32 pointed to by pHandle.

Microsoft Windows: Format the data block as a DIB. Use GlobalAlloc or equivalent under windows. Under 16 bit Microsoft Windows, place the handle in the low word of the TW_UINT32. The following assignment will work in either Win16 or Win32:

```
(HGLOBAL FAR *) pHandle = hDIB;
```

See the Windows SDK documentation under Structures: BIMAPINFO, BITMAPINFOHEADER, RGBQUAD. See also "DIBs and their use" by Ron Gery, in the Microsoft Development Library (MSDN CD).

Notes:

- Do not use BITMAPCOREINFO or BIMAPCOREHEADER as these are for OS/2 compatibility only.
- Always follow the BITMAPINFOHEADER with the color table and only save 1, 4, or 8 bit DIBs
- Color table entries are RGBQUADs, which are stored in memory as BGR not RGB.
- For 24 bit color DIBs, the "pixels" are also stored in BGR order, not RGB.
- DIBs are stored 'upside-down' - the first pixel in the DIB is the lower-left corner of the image, and the last pixel is the upper-right corner.
- DIBs can be larger than 64K, but be careful, a 24 bit pixel can straddle a 64K boundary!
- Pixels in 1, 4, and 8 bit DIBs are "always" color table indices, you must index through the color table to determine the color value of a pixel.

Macintosh: Format the data block as a PICT, preferably using standard system calls.

Microsoft Windows and Macintosh: If the allocation fails, it is recommended that you allow the user the option to re-size the image to fit within available memory or to cancel the transfer (assuming that the Source user interface is displayed). If the user chooses to cancel the transfer, return TWRC_CANCEL. If the user wants to re-size the image, the Source might choose to blindly crop the image, clip a selection region to the maximum supported size for the current memory configuration, or allow the user to re-acquire the image altogether. The user will usually feel more in control if you provide one or both of the last two options, but the first may make the most sense for your Source.

If the allocation fails and the image cannot be clipped, return TWRC_FAILURE and remain in State 6. Set the pHandle to NULL. The image whose transfer failed is still pending transfer. Do not decrement TW_PENDINGXFERS.Count.



### Return Codes

```
TWRC_XFERDONE /* Source done transferring the */
/* specified block */
```

TWRC_CANCEL /* User aborted the transfer */
/* within the Source */
TWRC_FAILURE
TWCC_BADDEST /* No such Source in session */
/* with application */
TWCC_LOWMEMORY /* Not enough memory for */
/* image--cannot crop to fit */
TWCC_OPERATIONERROR /* Failure in the Source-- */
/* transfer invalid */
TWCC_SEQERROR /* Operation invoked in */
/* invalid state */
/* The following introduced for 2.0 or higher */
TWCC_INTERLOCK /* Cover or door is open */
TWCC_DAMAGEDCORNER /* Document has a damaged corner */
TWCC_FOCUSERROR /* Focusing error during document capture */
TWCC_DOCTOOLIGHT /* Document is too light */
TWCC_DOCTOODARK /* Document is too dark */
TWCC_NOMEDIA /* Source has nothing to capture */

**See Also**

DG_IMAGE / DAT_IMAGEINFO / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
Capability - ICAP_XFERMECH

**Page ??, (8-4)**

*Leave. This is TWAIN.H stuff.*

**Page ??, (8-7)**

*Leave. This is TWAIN.H stuff.*

**Page ??, (8-22)**

*Leave? Why is it useful?.*

## TW_ELEMENT8

typedef struct {
TW_UINT8 Index;
TW_UINT8 Channel1;
TW_UINT8 Channel2;
TW_UINT8 Channel3;
} TW_ELEMENT8, FAR * pTW_ELEMENT8;

**Used by**

Embedded in the TW_GRAYRESPONSE, TW_PALETTE8 and TW_RGBRESPONSE structures

**Description**

This structure holds the tri-stimulus color palette information for TW_PALETTE8 structures. The order of the channels shall match their alphabetic representation. That is, for RGB data, R shall be channel 1. For CMY data, C shall be channel 1. This allows the application and Source to maintain consistency. Grayscale data will have the same values entered in all three channels.

**Field Descriptions**

Index Value used to index into the color table. Especially useful on the Macintosh.
Channel1 First tri-stimulus value (e.g. Red).
Channel2 Second tri-stimulus value (e.g. Green).
Channel3 Third tri-stimulus value (e.g. Blue).

**Page ??, (8-24)**

*Leave.. Details should be in Operating System Dependencies Chapter.*

## TW_EVENT

```
typedef struct {
TW_MEMREF pEvent;
TW_UINT16 TWMessage;
} TW_EVENT, FAR * pTW_EVENT;
```

**Used by**

DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT

**Description**

Used to pass application events/messages from the application to the Source. The Source is responsible for examining the event/message, deciding if it belongs to the Source, and returning an appropriate return code to indicate whether or not the Source owns the event/message. This process is covered in more detail in the Event Loop section of Chapter 3, "Application Implementation."

**Field Descriptions**

pEvent A pointer to the event/message to be examined by the Source.
Under Microsoft Windows, pEvent is a pMSG (pointer to a Microsoft Windows MSG struct). That is, the message the application received from GetMessage().
On the Macintosh, pEvent is a pointer to an EventRecord.
TWMessage Any message (MSG_xxxx) the Source needs to send to the application in response to processing the event/message. The messages currently defined for this purpose are MSG_NULL, MSG_XFERREADY and MSG_CLOSEDSREQ.

**Page ??, (8-48)**

*Leave.. Details should be in Operating System Dependencies Chapter.*

## TW_MEMORY

```
typedef struct {
TW_UINT32 Flags;
TW_UINT32 Length;
TW_MEMREF TheMem;
} TW_MEMORY, FAR * pTW_MEMORY;
```

**Used by**

Embedded in the TW_IMAGEMEMXFER and TW_JPEGCOMPRESSION structures

**Description**

Provides information for managing memory buffers. Memory for transfer buffers is allocated by the application--the Source is asked to fill these buffers. This structure keeps straight which entity is responsible for deallocation.

**Field Descriptions**

Flags Encodes which entity releases the buffer and how the buffer is referenced. The ownership flags must be used:

- when transferring Buffered Memory data as tiles
- when transferring Buffered Memory that is compressed
- in the TW_JPEGCOMPRESSION structure

When transferring Buffered Memory data as uncompressed strips, the application allocates the buffers and is responsible for setting the ownership flags.

This field is used to identify how the memory is to be referenced. The memory is always referenced by a Handle on the Macintosh and a Pointer under UNIX. It is referenced by a Handle or a pointer under Microsoft Windows.

Use TWMF_xxxx constants, bit-wise OR'd together to fill this field.

Flag Constants:

TWMF_APPOWNS 0x1
TWMF_DSMOWNS 0x2
TWMF_DSOWNS 0x4
TWMF_POINTER 0x8
TWMF_HANDLE 0x10

Length The size of the buffer in bytes. Should always be an even number and wordaligned.

TheMem Reference to the buffer. May be a Pointer or a Handle (see Flags field to make this determination). You must typecast this field before referencing it in your code.

**Page ??, (8-50)**

*Leave. This is TWAIN.H stuff.*

## TW_UINTPTR

**On Windows:**

```
typedef UINT_PTR TW_UINTPTR;
```

**On Macintosh and Unix:**

```
//32 bit GNU
typedef unsigned long TW_UINTPTR;
//64 bit GNU
typedef unsigned long long TW_UINTPTR;
```

**Used by**

Embedded in the TW_INFO structure.

**Description**

Integer pointer references are specific to each operating system. TWAIN defines TW_UINTPTR to be the integer pointer reference type supported by the operating system.

**Field Descriptions**

See definitions above

**Page ??, (8-57)**

*Leave. This is TWAIN.H stuff.*

## TW_SETUPFILEXFER

```
typedef struct {
TW_STR255 FileName;
TW_UINT16 Format;
TW_INT16 VRefNum;
} TW_SETUPFILEXFER, FAR * pTW_SETUPFILEXFER;
```

**Used by**

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET

**Description**

Describes the file format and file specification information for a transfer through a disk file.

**Field Descriptions**

FileName A complete file specifier to the target file. On Windows, be sure to include the complete pathname.

Format The format of the file the Source is to fill. Fill with the correct constant—as negotiated with the Source—of type TWFF_xxxx.

VRefNum The volume reference number for the file. This applies to Macintosh only. On Windows, fill the field with TWON_DONTCARE16.

**Page ??, (8-64)**

*Leave. This is TWAIN.H stuff.*

**DAT_PARENT**

Used by the DG_CONTROL / DAT_PARENT / MSG_OPENDSM and MSG_CLOSEDSM operations.

**On Windows:** They act on a variable of type TW_INT32. Prior to the operation, the application must write, a window handle to the application's window that acts as the "parent" for the Source's user interface. In Win 3.1 this would be in the low word, in Win 95 it will fill the entire field. (This must be done whether or not the Source's user interface will be used. The Source Manager uses this window handle to signal the application when data is ready for transfer (MSG_XFERREADY) or the Source needs to

be closed (MSG_CLOSEDSREQ)).
**On Macintosh:** These act on NULL data.

**Page ??, (8-101)**

*Leave. This is deprecated stuff.*

**Page ??, (10-119)**

*Leave? This is a definition.*

TWCP_NONE All Sources must support this.
TWCP_PACKBITS Macintosh PackBits format, (can be used with TIFF or PICT)
TWCP_GROUP31D,

**Page ??, (10-139)**

*Leave? This is a definition.*

TWFF_TIFF Used for document imaging
TWFF_PICT Native Macintosh format
TWFF_BMP Native Microsoft format
TWFF_XBM Used for document imaging

**Page ??, (11-1)**

*Leave? This is a definition.*

The TWAIN protocol defines no dynamic messaging system through which the application might determine, in real-time, what is happening in either the Source Manager or a Source. Neither does the protocol implement the native messaging systems built into the operating environments that TWAIN is defined to operate under (Microsoft Windows and Macintosh). This decision was made due to issues regarding platform specificity and higher-than-desired implementation costs.

**Page ??, (A-35)**

*Leave? This is a definition.*

| | | |
|---|---|---|
| ICAP_IMAGEFILEFORMAT | Preferred / User No default | TWFF_BMP (Windows)<br>TWFF_PICT (Macintosh) |

**Page ??, (B-1)**

*Leave. This is contact information.*