

---

# TWAIN Errata

**For Version 2.1**

**July 28<sup>th</sup>, 2010**

---



# Contents

Operation Triplets – Application to Source Manager .....	4
Requirements for an Application to be TWAIN-Compliant .....	5
Legacy Issues .....	6
DAT_SETUPFILEXFER2, TW_SETUPFILEXFER2, and TWSX_FILE2.....	7
Alternative User Interfaces.....	9
TW_USERINTERFACE.ShowUI, CAP_INDICATORS and TWCC_OPERATORERR.....	10
Requirements for a Source to be TWAIN-Compliant.....	13
Data Group / Data Argument Type / Message / Page.....	14
Operation Triplets – Source to Application .....	15
Currently Defined Condition Codes .....	16
DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK .....	17
DG_CONTROL / DAT_CAPABILITY / MSG_GETHELP .....	18
DG_CONTROL / DAT_CAPABILITY / MSG_GETLABEL .....	19
MSG_RESET / MSG_GETDEFAULT / MSG_RESETALL / APPENDIX A Defaults .....	20
DG_CONTROL / DAT_NULL / MSG_CLOSEDOK.....	21
DG_CONTROL / DAT_PARENT / MSG_CLOSEDM .....	22
DG_CONTROL / DAT_PARENT / MSG_OPENM .....	23
DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER.....	24
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET .....	25
TW_INFO .....	26
DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET.....	27
TW_HANDLE.....	28
TW_PENDINGXFERS .....	29
Constants.....	30
ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE .....	31
Deprecated Items .....	32
Extended Image Attribute Capabilities (TW_HANDLE fix) .....	33
Extended Image Attribute Capabilities (Pixel fix) .....	34
DEVICEEVENT .....	35
CAP_FEEDERALIGNMENT.....	36
Chapter 10: DG_CONTROL / DAT_CAPABILITY / MSG_GET.....	37
CAP_CAMERAENABLED.....	39
CAP_CAMERAORDER .....	40
CAP_CAMERASIDE.....	41
CAP_DEVICEEVENT .....	42
CAP_DUPLEXENABLED .....	44

<b>CAP_JOBCONTROL</b> .....	<b>45</b>
<b>Rename CAP_POWERSAVETIME to CAP_POWERDOWNTIME</b> .....	<b>46</b>
<b>ICAP_AUTOMATICCOLORENABLED</b> .....	<b>47</b>
<b>ICAP_BITDEPTH</b> .....	<b>48</b>
<b>ICAP_EXTIMAGEINFO</b> .....	<b>49</b>
<b>ICAP_FRAMES</b> .....	<b>50</b>
<b>ICAP_ICCPROFILE</b> .....	<b>51</b>
<b>ICAP_ORIENTATION</b> .....	<b>52</b>
<b>ICAP_SUPPORTEDEXTIMAGEINFO</b> .....	<b>53</b>
<b>Capability Ordering</b> .....	<b>54</b>
<b>PDF Cross References</b> .....	<b>55</b>
<b>Appendix A Capability Default-Values Table</b> .....	<b>56</b>
<b>Internationalization</b> .....	<b>57</b>
<b>The ImageData and Its Layout</b> .....	<b>65</b>
<b>DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET</b> .....	<b>68</b>
<b>DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET</b> .....	<b>69</b>
<b>TW_FRAME</b> .....	<b>70</b>
<b>TW_IMAGELAYOUT</b> .....	<b>71</b>
<b>TWMF_DSOWNS</b> .....	<b>72</b>
<b>NativeTransfer</b> .....	<b>73</b>
<b>Wrong Capability IDs</b> .....	<b>74</b>
<b>Wrong Names</b> .....	<b>76</b>
<b>Missing File Format description</b> .....	<b>78</b>
<b>Missing constants</b> .....	<b>79</b>
<b>Wrong constants</b> .....	<b>80</b>

---

## Operation Triplets – Application to Source Manager

Page 3-11 (PDF page 43)

### Operation Triplets - Application to Source Manager

There are nine operation triplets that can be sent from the application to be consumed by the Source Manager. They all use the `DG_CONTROL` data group and they use three different data argument types: `DAT_IDENTITY`, `DAT_PARENT`, and `DAT_STATUS`. The following table lists the data group, data argument type, and messages that make up each operation. The list is in alphabetical order not the order in which they are typically called by an application. Details about each operation are available in reference format in [Chapter 7, "Operation Triplets."](#)

#### Control Operations from Application to Source Manager

##### **DG\_CONTROL / DAT\_IDENTITY**

<code>MSG_CLOSED</code>	:	Prepare specified Source for unloading
<code>MSG_GETDEFAULT</code>	:	Get identity information of the default Source
<code>MSG_GETFIRST</code>	:	Get identity information of the first available Source
<code>MSG_GETNEXT</code>	:	Get identity of the next available Source
<code>MSG_OPENS</code>	:	Load and initialize the specified Source
<code>MSG_SET</code>	:	Set identity information of the default Source
<code>MSG_USERSELECT</code>	:	Present "Select Source" dialog

---

## Requirements for an Application to be TWAIN-Compliant

Page 3-41 (PDF page 73)

- Add the items below

TWAIN 2.1 Applications must support all TWAIN 2.0 required features and the following:

### TWAIN Data Type

TWTY\_HANDLE if supports DAT\_EXTIMAGEINFO

### TWAIN Condition Codes

TWCC\_NOMEDIA if supports scanning with UI and Indicators suspended.

TWAIN 2.0 Applications must support all TWAIN 1.9 required features and the following:

### TWAIN Data Flag

DF\_APP2, DF\_DS2, DF\_DSM2

TW\_IDENTITY.SupportedGroups |= DF\_APP2

### TWAIN Condition Codes

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 \*/

DG\_CONTROL DAT\_ENTRYPOINT MSG\_GET

DG\_CONTROL / DAT\_CALLBACK / MSG\_REGISTERCALLBACK (Required by Mac OS X and Linux, recommended for Windows)

### Memory Functions

Use the memory functions of the DSM when talking to a TWAIN 2 Source.

---

## Legacy Issues

Page 3-42 (PDF page 74)

- Add this “Legacy Issues” section at the same level as the previous section “Requirements for an Application to be TWAIN Compliant”

### ICAP\_BITDEPTH

#### Data Sources

Report the number-of-channels times the depth-per-channel. For example, a typical value for ICAP\_BITDEPTH when ICAP\_PIXELTYPE is TWPT\_RGB is  $3 \times 8 = 24$ .

#### Applications

Ambiguity in the Specification prior to version 2.2 may result in some Data Sources reporting just the depth-per-channel. In the majority of cases a value of 8 for ICAP\_BITDEPTH when ICAP\_PIXELTYPE is TWPT\_RGB may be treated as if the bit depth is really 24.

### CAP\_DUPLEXENABLED

#### Data Sources

If a Data Source supports one of MSG\_GET, MSG\_GETCURRENT, or MSG\_GETDEFAULT for a capability, it should support all get messages.

#### Applications

Ambiguity in the Specification prior to version 2.2 may result in some Data Sources not supporting MSG\_GET for CAP\_DUPLEXENABLED. The Data Source may only support MSG\_GETCURRENT to determine if duplex option is enabled or not.

### ICAP\_FRAMES

#### Applications

Some scanners may handle having the origin of a frame as 0,0 differently. The spec states that when an application is only interested in the extent of image scanned it can set the origin to 0,0 with MSG\_SET. Some center feed or right feed scanners may scan from the left edge of the scanner. They expect the application to center (or right align) the frame using the physical extent of the scanner.

---

## DAT\_SETUPFILEXFER2, TW\_SETUPFILEXFER2, and TWSX\_FILE2

Page 4-18 (PDF page 92)

- Update text under “Disk File Mode Transfer”

### Disk File Mode Transfer

The disk file mode is identified as TWSX\_FILE. Sources are not required to support Disk File Transfer so it is important to verify its support.

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.

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

#### During State 4:

- Set the ICAP\_XFERMECH to TWSX\_FILE. 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):

```
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 10-11 (PDF page 433)

## ACAP\_XFERMECH

<b>Allowed Values:</b>	TWSX_NATIVE TWSX_FILE
<b>Container for MSG_GET:</b>	TW_ENUMERATION TW_ONEVALUE

## ICAP\_XFERMECH

<b>Allowed Values:</b>	TWSX_NATIVE TWSX_FILE TWSX_MEMORY TWSX_MEMFILE
<b>Container for MSG_GET:</b>	TW_ENUMERATION TW_ONEVALUE

### General Capability Negotiation

ICAP\_XFERMECH selects the way an image is transferred from the Source to an Application, which has an impact on some of the characteristics of an image, which is why this value must be selected first. If TWSX\_NATIVE is selected, then no other action related to image transfer is needed. If TWSX\_FILE is selected, then the application should negotiate ICAP\_IMAGEFILEFORMAT, which will be used when DAT\_SETUPFILEXFER is called. If TWSX\_MEMORY is selected, then DAT\_SETUPMEMXFER will need to be called. The Application may then opt to negotiate ICAP\_TILES.

---

## Alternative User Interfaces

Page 4-37 (PDF page 111)

Displaying a custom selection interface:

1. Use the `DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST` operation to have the Source Manager locate the first Source available. The name of the Source is contained in the `TW_IDENTITY.ProductName` field. Save the `TW_IDENTITY` structure.
2. Use the `DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT` to have the Source Manager locate the next Source. Repeatedly use this operation until it returns `TWRC_ENDOFLIST` indicating no more Sources are available. Save the `TW_IDENTITY` structure.
3. Use the `ProductName` information to display the choices to the user. Once they have made their selection, use the saved `TW_IDENTITY` structure and the `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS` operation to have the Source Manager open the desired Source. (Note, using this approach, as opposed to the `SG_USERSELECT` operation, the Source Manager does not update the system default Source information to reflect your choice.)
4. Use the `DG_CONTROL / DAT_IDENTITY / MSG_SET` to set the system default source.

Transparently selecting a Source:

---

## TW\_USERINTERFACE.ShowUI, CAP\_INDICATORS and TWCC\_OPERATORERR

Page 5-8 (PDF page 124)

- Insert the following under the section “Displaying the User Interface” after the paragraph starting “Sources are not required to allow themselves...”

### User Interface

Sources that report TRUE for CAP\_UICONTROLLABLE must allow acquisition with the UI disabled, and they must support TRUE and FALSE for CAP\_INDICATORS.

If the Application sets ShowUI to TRUE when 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 when 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 when 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.

Page 4-38 (PDF page 112)

### Alternatives to Using the Source’s User Interface

Just as with the Source Manager’s Select Source dialog, the application may ask to not use the Source’s user interface. Certain types of applications may not want to have the Source’s user interface displayed. An example of this can be seen in some text recognition packages that wish to negotiate a few capabilities (i.e. pixel type, resolution, page size) and then proceed directly to acquiring and transferring the data.

Some Sources may display the UI even when ShowUI is set to FALSE. An application can determine whether ShowUI can be set by interrogating the CAP\_UICONTROLLABLE capability. If CAP\_UICONTROLLABLE returns FALSE but the ShowUI input value is set to FALSE in an activation of DG\_CONTROL / DAT\_USERINTERFACE / MSG\_ENABLEDS, the enable DS operation returns TWRC\_CHECKSTATUS but displays the UI regardless. Therefore, an application that requires that the UI be disabled should interrogate CAP\_UICONTROLLABLE before issuing MSG\_ENABLEDS.

To Enable the Source without Displaying its User Interface

- Use the DG\_CONTROL / DAT\_USERINTERFACE / MSG\_ENABLEDS operation.
- Set the ShowUI field of the TW\_USERINTERFACE structure to FALSE.
- When the command is received and accepted (TWRC\_SUCCESS), the Source does not display a user interface but is armed to begin capturing data. For example, in a flatbed scanner, the light bar will light and begin to move. A handheld scanner will be armed and ready to acquire data when the “go” button is pressed on the scanner. Other devices may respond differently but they all will either begin acquisition immediately or be armed to begin acquiring data as soon as the user interacts with the device.

Capability Negotiation is Essential when the Source’s User Interface is not Displayed

- Since the Source’s user interface is not displayed, the Source will not be giving the user the opportunity to select the information to be acquired, etc. Unless default values are acceptable,

current values for all image acquisition and control parameters must be negotiated before the Source is enabled, i.e. while the session is in State 4.

When `TW_USERINTERFACE.ShowUI` is set to `FALSE`:

- A Source that does not support `ShowUI` set to `FALSE` will return `TWRC_CHECKSTATUS` and display the UI regardless.
- The application is still required to pass all events to the Source (via the `DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT` operation) while the Source is enabled.
- The Source must display the minimum possible user interface containing only those controls required to make the device useful in context. In general, this means that no user interface is displayed, however certain devices may still require a trigger to initiate the scan.
- If the Source user interface is not displayed, and the Application sets `CAP_INDICATORS` to `TRUE`, then the Source displays a progress indicator during acquisition and transfer, and an error can result in the Source showing a dialog to the user.
- If the Source user interface is not displayed, and the Application sets `CAP_INDICATORS` to `FALSE`, then the Source is not allowed to display any kind of user interface, progress indicator or error dialog. All UI activity must be suppressed.
- If the Source user interface is displayed then the Source will ignore the setting for `CAP_INDICATORS`. A progress indicator is displayed during acquisition and transfer, and errors can result in the Source showing a dialog to the user.
- The Source still sends the application a `MSG_XFERREADY` notice when the data is ready to be transferred.
- The Source may or may not send a `MSG_CLOSEDREQ` to the application asking to be closed since this is often user-initiated. Therefore, after the Source has returned to State 5 (following the `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` operation and the `TW_PENDINGXFERS.Count = 0`), the application can send the `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDREQ` operation.

Page 5-8 (PDF page 124)

### 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.

Error messages are suppressed when the UI is not displayed and `CAP_INDICATORS` is set to `FALSE`.

### Progress Indicators

- If the Source user interface is not displayed, and the Application sets `CAP_INDICATORS` to `TRUE`, then the Source displays a progress indicator during acquisition and transfer, and an error can result in the Source showing a dialog to the user.
- If the Source user interface is not displayed, and the Application sets `CAP_INDICATORS` to `FALSE`, then the Source is not allowed to display any kind of user interface, progress indicator or error dialog. All UI activity must be suppressed.
- If the Source user interface is displayed then the Source will ignore the setting for `CAP_INDICATORS`. A progress indicator is displayed during acquisition and transfer, and errors can result in the Source showing a dialog to the user.

**Note:** If the application has set ShowUI or CAP\_INDICATORS to TRUE, then the Source is responsible for presenting the user with appropriate progress indicators regarding the acquisition and transfer process. If ShowUI is set to TRUE, CAP\_INDICATORS is ignored and progress and errors are always shown.

---

## Requirements for a Source to be TWAIN-Compliant

Page 5-18 to 5-21 (PDF page 134-137)

- Update the section using the Mandatory White Paper

---

## Data Group / Data Argument Type / Message / Page

Page 7-3 (PDF page 151)

DG_IMAGE	DAT_CIECOLOR	MSG_GET	7-110
DG_IMAGE	DAT_EXTIMAGEINFO	MSG_GET	7-112
DG_IMAGE	DAT_GRAYRESPONSE	MSG_RESET	7-114
		MSG_SET	7-115
DG_IMAGE	DAT_ICCPROFILE	MSG_GET	7-116
DG_IMAGE	DAT_IMAGEFILEXFER	MSG_GET	7-118
DG_IMAGE	DAT_IMAGEINFO	MSG_GET	7-120
DG_IMAGE	DAT_IMAGELAYOUT	MSG_GET	7-122
		MSG_GETDEFAULT	7-124
		MSG_RESET	7-125
		MSG_SET	7-126
DG_IMAGE	DAT_IMAGEMEMFILEXFER	MSG_GET	7-128
DG_IMAGE	DAT_IMAGEMEMXFER	MSG_GET	7-131
<del>DG_IMAGE</del>	<del>DAT_IMAGEFILEXFER</del>	<del>MSG_GET</del>	<del>7-128</del>
DG_IMAGE	DAT_IMAGENATIVEXFER	MSG_GET	7-134

---

## Operation Triplets – Source to Application

Page 7-4 (PDF page 152)

- Add page number for MSG\_CLOSED\_SOK

---

## Currently Defined Condition Codes

Page 7-7 (PDF page 155)

### Return Codes

```
TWRC_CANCEL
TWRC_XFERDONE
TWRC_FAILURE
    TWCC_BADPROTOCOL
    TWCC_OPERATIONERROR
    TWCC_SEQERROR - not state 6.
    /* The following introduced for 2.0 or higher */
    TWCC_FILEWRITEERROR
```

Page 7-93 (PDF page 241)

### 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 */
/* The following introduced for 2.0 or higher */
    TWCC_FILEWRITEERROR
```

Page 7-95 (PDF page 243)

### 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 */
/* The following introduced for 2.0 or higher */
    TWCC_FILEWRITEERROR
```

Page 7-119 (PDF page 267)

### Return Codes

```
TWRC_XFERDONE
TWRC_CANCEL
TWRC_FAILURE
    TWCC_BADDEST          /* No such Source in-session */
                        /* with application */
    TWCC_OPERATIONERROR  /* Failure in the Source -- */
                        /* transfer invalid */
    TWCC_SEQERROR        /* Operation invoked in */
                        /* invalid state */
/* The following introduced for 2.0 or higher */
    TWCC_FILEWRITEERROR
    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 */
```

---

## DG\_CONTROL / DAT\_CALLBACK / MSG\_REGISTER\_CALLBACK

Page 7-11 (PDF page 159)

### Call

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

### Valid States

4

### Description

This triplet is sent to the DSM by the Application to register the application's entry point with the DSM, so that the DSM can use callbacks to inform the application of events generated by the DS.

The last argument is a pointer to an initialized TW\_CALLBACK structure. The TW\_CALLBACK structure should be initialized as follows:

CallBackProc	The callback function's entry point, <b>used by DSM to send DAT_NULL/MSG_XXX</b>
RefCon	An application defined reference constant. <b>Returned as _pData in callback.</b>

**Note:** Application should refrain from assigning a pointer to RefCon if they want the same behavior in 32bit and 64bit. RefCon is not large enough to hold a pointer as 64bit.

### Return Codes

TWRC\_SUCCESS  
TWRC\_FAILURE  
TWCC\_BADVALUE

### See Also

DG\_CONTROL / DAT\_CALLBACK / MSG\_INVOKE\_CALLBACK

---

## DG\_CONTROL / DAT\_CAPABILITY / MSG\_GETHELP

Page 7-21 (PDF page 169)

### Application

The Application frees the handle.

### Source

The Source returns a TW\_ONEVALUE container with a TWTY\_HANDLE item type. The handle points to a string. The encoding of the string is determined by the TW\_IDENTITY.TW\_VERSION.Language reported back by the Source, unless overridden by CAP\_LANGUAGE.

### Return Codes

TWRC\_SUCCESS  
TWRC\_FAILURE  
TWCC\_BADPROTOCOL  
TWCC\_CAPUNSUPPORTED

---

## DG\_CONTROL / DAT\_CAPABILITY / MSG\_GETLABEL

Page 7-22 (PDF page 170)

### Application

The Application frees the handle.

### Source

The Source returns a TW\_ONEVALUE container with a TWTY\_HANDLE item type. The handle points to a string. The encoding of the string is determined by the TW\_IDENTITY.TW\_VERSION.Language reported back by the Source, unless overridden by CAP\_LANGUAGE.

### Return Codes

TWRC\_SUCCESS  
TWRC\_FAILURE  
TWCC\_BADPROTOCOL  
TWCC\_CAPUNSUPPORTED

Page 7-30 (PDF page 178)

**Description**

This command resets all current values back to original power-on defaults. All current values are set to their default value except is the where mandatory values are required. All constraints are removed for all of the negotiable capabilities supported by the driver.

---

## DG\_CONTROL / DAT\_NULL / MSG\_CLOSEDOK

New section, add this after Page 7-73 (PDF page 221)

### Call

`DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_CLOSEDOK, 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 with all changes preserved.

On Windows, the Source sends this `DG_CONTROL / DAT_NULL / MSG_CLOSEDOK` 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_CLOSEDOK` 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 \*/

### See Also

`DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT`

`DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLED`

---

## DG\_CONTROL / DAT\_PARENT / MSG\_CLOSEDISM

Page 7-79 (PDF page 227)

### Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_PARENT, MSG_CLOSEDISM, pParent);
```

*pParent should be the same value used in MSG\_OPENDISM.*

### 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.

*After the Source Manager is closed the unique ID assigned to pOrigin->Id is no longer valid.*

Page 8-70 (PDF page 368)

MSG_OPENDISM	0x0301
MSG_CLOSEDISM	0x0302

---

## DG\_CONTROL / DAT\_PARENT / MSG\_OPENDSM

Page 7-80 (PDF page 228)

### 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\_HANDLE** and must contain the window handle.

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

### Source Manager

Initializes and prepares itself for subsequent operations. Maintains a copy of pParent.

If successfully opened, the Source Manager will assign a unique ID to pParent->Id for this application.

---

## DG\_CONTROL / DAT\_PENDINGXFERS / MSG\_ENDXFER

Page 7-82 (PDF page 230)

- Add to the **Application** section.

When DAT\_XFERGROUP is set to DG\_IMAGE and CAP\_JOBCONTROL is set to other than TWJC\_NONE then check pPendingXfers->EOJ for TWEJ\_XXX Job control value.

Page 7-83 (PDF page 231)

- Add to the **Source** section.

When DAT\_XFERGROUP is set to DG\_IMAGE and CAP\_JOBCONTROL is set to other than TWJC\_NONE then pPendingXfers->EOJ should reflect the current TWEJ\_XXX Job control value.

---

## DG\_CONTROL / DAT\_PENDINGXFERS / MSG\_GET

Page 7-84 (PDF page 232)

- Add to the **Application** section.

When DAT\_XFERGROUP is set to DG\_IMAGE and CAP\_JOBCONTROL is set to other than TWJC\_NONE then check pPendingXfers->EOJ for TWEJ\_XXX Job control value.

- Add to the **Source** When DAT\_XFERGROUP is set to DG\_IMAGE: section.

When CAP\_JOBCONTROL is set to other than TWJC\_NONE then pPendingXfers->EOJ should reflect the current TWEJ\_XXX Job control value.

---

## TW\_INFO

Page 7-112 (PDF page 260)

```
pExtImageInfo->Info[0].ReturnCode = TWRC_INFONOTSUPPORTED;  
pExtImageInfo->Info[0].ReturnCode = TWRC_SUCCESS;
```

Page 8-25 to 8-29 (PDF page 323 to 327)

```
ReturnCode = 0  
ReturnCode = TWRC_SUCCESS
```

Page 8-44 (PDF page 342)

```
TW_UINT16 ReturnCode;
```

**ReturnCode** This is the return code of availability of data for extended image attribute requested. Following is the list of possible condition codes:

---

## DG\_IMAGE / DAT\_EXTIMAGEINFO / MSG\_GET

Page 7-112 (PDF page 260)

If the application requests information that the Source does not recognize, the Source should put `TWRC_INFONOTSUPPORTED` in the `ReturnCode` field of `TW_INFO` structure.

```
pExtImageInfo->Info[0].ReturnCode = TWRC_INFONOTSUPPORTED;
```

If the application requests information that the Source recognizes but is currently not available, the Source should put `TWRC_INFONOTAVAILABLE` in the `ReturnCode` field of `TW_INFO` structure.

```
pExtImageInfo->Info[0].ReturnCode = TWRC_INFONOTAVAILABLE;
```

If you support the capability, fill in the fields allocating extra memory if necessary. For example, for `TWEI_BARCODEX`:

Page 7-113 (PDF page 261)

~~For handle (Application set `TWMF_HANDLE`),~~

---

## TW\_HANDLE

Page 8-35 (PDF page 333)

See “Platform Specific Typedefs” on page 8-4. for information on the actual mapping of this type.

### Used by

Embedded in the TW\_CAPABILITY and TW\_USERINTERFACE structures, and used by TW\_INFO and TW\_ONEVALUE structures when ItemType is TWTY\_HANDLE. When used in a capability TW\_HANDLE must reflect a string. For TW\_INFO, Application writers will need to look at the metadata to determine if the Handle is a string or binary data.

### Description

The typedef of Handles are defined by the operating system. TWAIN defines TW\_HANDLE to be the handle type supported by the operating system. Identified as a TW\_HANDLE by setting ItemType to TWTY\_HANDLE where appropriate.

### Field Descriptions

See definitions above

---

## TW\_PENDINGXFERS

Page 8-54 (PDF page 352)

### Field Descriptions

Count	<p>When <code>DAT_XFERGROUP</code> is set to <code>DG_IMAGE</code>, the number of complete transfers a Source has available for the application it is connected to. If no more transfers are available, set to zero. If an unknown and non-zero number of transfers are available, set to -1.</p> <p>When <code>DAT_XFERGROUP</code> is set to <code>DG_AUDIO</code>, the number of complete audio snippet transfers for a given image a Source has available for the application it is connected to. If no more transfers are available, set to zero. -1 is not a valid value.</p>
EOJ	<p>The application should check this field if the <code>CAP_JOBCONTROL</code> is set to other than <code>TWJC_NONE</code>. If the EOJ is not 0, the application should expect more data from the driver according to <code>CAP_JOBCONTROL</code> settings.</p> <p>The source should fill this value with one of the <code>TWEJ_xxx</code> patch codes if <code>CAP_JOBCONTROL</code> is set to other than <code>TWJC_NONE</code>.</p>
Reserved	Maintained so as not to cause compile time errors for pre-1.7 code.

---

## Constants

Pages 8-65 through 8-101 (PDF pages 363-399)

- Add a “Version” column to each table in this section.

Page 8-67 (PDF page 365)

2.1	TWTY_HANDLE	0x000F //Item is a TW_HANDLE
-----	-------------	------------------------------

---

## ICAP\_AUTOMATICCOLORNONCOLORPIXELTYPE

Page 8-79 (PDF page 377)

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

---

## Deprecated Items

Page 8-102 (PDF page 400)

Capabilities	CAP_SUPPORTEDCAPSEXT	0x100c	
	CAP_FILESYSTEM	0x????	
	CAP_PAGEMULTIPLEACQUIRE	0x1023	/* Added 1.8 */
	CAP_PAPERBINDING	0x102f	/* Added 1.8 */
	CAP_PASSTHRU	0x1031	/* Added 1.8 */
	CAP_POWERSAVETIME	0x1034	/* Added 1.8, deprecated */
			/* 0x1034 has been reused */
			/* by CAP_CAMERASIDE */

---

## Extended Image Attribute Capabilities (TW\_HANDLE fix)

Pages 9-2 (PDF page 404), 9-19 (PDF page 421)

- Example

**Value Type:** TW\_HANDLE

- Should become

**Value Type:** TWTY\_HANDLE

---

## Extended Image Attribute Capabilities (Pixel fix)

Pages 9-3 through 9-17 (PDF pages 405 through 419)

- Change the following items, looking for the word “coordinate”, adding the words “in pixels” after each occurrence.

TWEI\_BARCODEX, TWEI\_BARCODEY, TWEI\_DESHADETOP, TWEI\_DESHADELEFT, TWEI\_DESHADEHEIGHT, TWEI\_DESHADEWIDTH, TWEI\_DESHADESIZE, TWEI\_HORZLINECOORD, TWEI\_HORZLINEYCOORD, TWEI\_HORZLINELENGTH, TWEI\_HORZLINETHICKNESS, TWEI\_VERTLINECOORD, TWEI\_VERTLINEYCOORD, TWEI\_VERTLINELENGTH, TWEI\_VERTLINETHICKNESS, TWEI\_SKEWWINDOWX1, TWEI\_SKEWWINDOWY1, TWEI\_SKEWWINDOWX2, TWEI\_SKEWWINDOWY2, TWEI\_SKEWWINDOWX3, TWEI\_SKEWWINDOWY3, TWEI\_SKEWWINDOWX4, , TWEI\_SKEWWINDOWY4, TWEI\_FORMHORZDOCOFFSET, TWEI\_FORMVERTDOCOFFSET, and TWEI\_FRAME

Example:

### TWEI\_BARCODEX

<b>Description</b>	The X coordinate <b>in pixels</b> of a bar code found on a page.
--------------------	--

---

## DEVICEEVENT

Page 10-2 (PDF page 424)

### Capabilities in Categories of Functionality

#### Asynchronous Device Events

CAP\_DEVICEEVENT

MSG\_SET selects which events the application wants the source to report; MSG\_RESET resets the capability to the empty array (no events set).

---

## CAP\_FEEDERALIGNMENT

Page 10-7 (PDF page 429)

CAP\_FEEDERALIGNMENT Indicates the alignment of the document feeder.

Page 10-48 (PDF page 470)

### Description

Helps the Application determine any special actions it may need to take when negotiating frames with the Source.

TWFA_NONE:	The alignment is free-floating. Applications should assume that the origin for frames is on the left.
TWFA_LEFT:	The alignment is to the left.
TWFA_CENTER:	The alignment is centered. This means that the paper will be fed in the middle of the ICAP_PHYSICALWIDTH of the device. If this is set, then the Application should calculate any frames with a left offset
TWFA_RIGHT:	The alignment is to the right. <b>If this is set, then the Application should calculate any frames with a left offset.</b>

---

## Chapter 10: DG\_CONTROL / DAT\_CAPABILITY / MSG\_GET

Page 10-10 (PDF page 432)

### Containers

<b>MSG_GETCURRENT &amp; MSG_GETDEFAULT:</b>	Acceptable containers for use on MSG_GETCURRENT and MSG_GETDEFAULT operations.
<b>MSG_GET</b>	Acceptable containers for use on MSG_GET operations.
<b>MSG_RESET</b>	Acceptable containers for use on MSG_RESET operations.
<b>MSG_SET</b>	Acceptable containers for use on MSG_SET operations.

### Required By

If a Source or application is required to support the capability.

### Source Required Operations

Operations the Source is required to support.

### TWAIN Version Introduced

Version 2.1

### See Also

#### *Example*

### Values

<b>Type:</b>	TW_BOOL
<b>Default Value:</b>	None
<b>Allowed Values:</b>	TRUE or FALSE

### Containers

<b>MSG_GETCURRENT &amp; MSG_GETDEFAULT:</b>	TW_ONEVALUE	
<b>MSG_GET:</b>	TW_ONEVALUE,	<i>// for backwards compatibility with</i>
<b>1.x applications only</b>		
	TW_ENUMERATION	<i>// mandatory for 2.1 application</i>
<b>and higher</b>		
<b>MSG_RESET:</b>	Not allowed	
<b>MSG_SET:</b>	Not allowed	
<b>MSG_QUERYSUPPORT:</b>	TW_ONEVALUE	

### Required By

None

### Source Required Operations

None

### TWAIN Version Introduced

Version 2.1

Pages 10-16, 10-19, 10-21, 10-25, 10-33, 10-44, 10-50, 10-54, 10-55, 10-57, 10-65, 10-71, 10-88, 10-91, 10-93, 10-95, 10-96, 10-98, 10-99, 10-102, 10-118, 10-124, 10-148, 10-159, 10-186, 10-188

**MSG\_GET:** TW\_ONEVALUE, // for backwards compatibility with 1.x applications only  
TW\_ENUMERATION // mandatory for 2.1 applications and higher

Pages 10-28, 10-35, 10-41, 10-45, 10-51, 10-66, 10-77, 10-79, 10-89, 10-94

**MSG\_GET:** TW\_ONEVALUE,  
TW\_ENUMERATION // allowed for 2.0 applications and higher

---

## CAP\_CAMERAENABLED

Page 10-25 (PDF page 447)

### Description

This feature depends on “camera addressing”, which is the ability to address elements in the device responsible for the color space or location. TWAIN offers DAT\_FILESYSTEM and CAP\_CAMERASIDE to do this.

When set to TRUE the device will deliver images from the current camera. The Current Camera can be selected with either CAP\_CAMERASIDE or DAT\_FILESYSTEM. With CAP\_CAMERASIDE it is possible to enable bottom (rear) only scanning, or have different settings for top and bottom. With DAT\_FILESYSTEM it is possible to enter a Single Document Multiple Images (SDMI) mode in addition to enabling different settings for top and bottom.

### Application

CAP\_CAMERASIDE is easier to use, but cannot be used for SDMI. To enable bottom only scanning, set CAP\_CAMERASIDE to TWCS\_BOTTOM and set CAP\_CAMERAENABLED to TRUE, then set CAP\_CAMERASIDE to TWCS\_TOP and set CAP\_CAMERAENABLED to FALSE.

With DAT\_FILESYSTEM an application can traverse and control all cameras individually.

An application should not use both CAP\_CAMERASIDE and DAT\_FILESYSTEM to address a camera.

Avoid using ICAP\_PIXELTYPE **after** setting CAP\_CAMERAENABLED. ICAP\_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 (simplex or duplex) will result in getting color images.

The application is not allowed to turn off CAP\_CAMERAENABLED for all cameras.

### Source

A Source that supports CAP\_CAMERAENABLED must support DAT\_FILESYSTEM or CAP\_CAMERASIDE or both.

If CAP\_CAMERASIDE is supported, the application can use it to set the driver up for bottom (rear) only scanning. Set CAP\_CAMERASIDE to TWCS\_BOTTOM and set CAP\_CAMERAENABLED to TRUE, then set CAP\_CAMERASIDE to TWCS\_TOP and set CAP\_CAMERAENABLED to FALSE.

If DAT\_FILESYSTEM is supported, then the application may be able to enter Single Document Multiple Images (SDMI) mode. In this mode the application can independently address the color, grayscale, bitonal, top and bottom cameras as supported by the driver. If the application sets CAP\_CAMERAENABLED to TRUE for more than one “pixel type” on the same camera side, (for instance, color and bitonal on the front) then the driver will output multiple images for that side of the document.

When ICAP\_PIXELTYPE is set or reset and CAP\_CAMERASIDE is set to TWCS\_BOTH, 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.

If not supported, return TWRC\_FAILURE / TWCC\_CAPUNSUPPORTED.

**Note:** It is not recommended that applications mix the use of ICAP\_PIXELTYPE with DAT\_FILESYSTEM or CAP\_CAMERASIDE. ICAP\_PIXELTYPE is intended for simple applications that only want to choose color, grayscale or bitonal. Applications that want to provide bottom (rear) only scanning should use DAT\_FILESYSTEM or CAP\_CAMERASIDE. Applications that want to provide Single Document Multiple Images should use DAT\_FILESYSTEM.

---

## CAP\_CAMERAORDER

Page 10-27 (PDF page 449)

### 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. DAT\_FILESYSTEM is used to address each camera.

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, and will be ignored.

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.

---

## CAP\_CAMERASIDE

Page 10-29 (PDF page 451)

### Description

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.

### Application

The application sets which camera it wishes to address with CAP\_CAMERASIDE. The application then sets any capability that allows independent values for the top and bottom.

There is no easy way to determine if a capability supports independent values for the top and bottom, though as a general rule the ICAP\_ capabilities are more likely to allow this. An application can determine support by setting one side, then testing the other side to see if it has changed.

Mixing camera selection using DAT\_FILESYSTEM and CAP\_CAMERASIDE is not recommended, and may produce unexpected results.

### Source

If set to TWCS\_BOTH (the default) then DAT\_CAPABILITY / MSG\_SET and MSG\_RESET operations apply to the top and bottom. MSG\_GET operations get their data from the top camera.

If set to TWCS\_TOP or TWCS\_BOTTOM, and if the capability being negotiated allows separate values for the top and bottom, then only the side addressed by this capability will be changed as part of a MSG\_SET or MSG\_RESET, or returned as part of a MSG\_GET.

If a capability does not allow separate values for the top and bottom (for instance CAP\_DUPLEXENABLED), then the current value of CAP\_CAMERASIDE has no impact on how it is negotiated.

CAP\_CAMERASIDE and CAP\_DUPLEXENABLED are independent and have no effect on each other. That is, if CAP\_DUPLEXENABLED is FALSE CAP\_CAMERASIDE can still be set to TWCS\_BOTTOM.

If DAT\_FILESYSTEM is also supported by the source, it must keep it in sync with the current value of this capability.

---

## CAP\_DEVICEEVENT

Page 10-38 to 10-39 (PDF page 460- 461)

### Description

MSG\_SET selects which events the Application wants the Source to report. MSG\_GET and 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 and 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\_QUERY SUPPORT)

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

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

---

## **CAP\_DUPLEXENABLED**

Page 10-44 (PDF page 466)

### **Application**

Application should send `MSG_GET` or `MSG_GETCURRENT` to determine if the duplex option is enabled or not.

---

## CAP\_JOBCONTROL

Page 10-58 (PDF page 480)

If the application selects options other than none, it should check the **EOJ** field for one of the **TWEJ\_XXXX** patch codes of the **PENDINGXFERS** data.

---

## Rename CAP\_POWERSAVETIME to CAP\_POWERDOWNTIME

Page 10-67 (PDF page 489)

<b>Type:</b>	TW_INT32
<b>Default Value:</b>	No Default
<b>Allowed Values:</b>	>= -1
<b>Container for <i>MSG_GET</i>:</b>	TW_ONEVALUE
<b>Container for <i>MSG_SET</i>:</b>	TW_ONEVALUE
<b>Container for <i>MSG_QUERY SUPPORT</i>:</b>	TW_ONEVALUE

Page A-28 (PDF page 654)

### Power Supply

CAP\_POWERSUPPLY reports which power supply is currently in effect for the Source. CAP\_BATTERYPERCENTAGE, CAP\_BATTERYMINUTES and CAP\_POWERSAVETIME are available at all times, though the values they report may change depending on the current value of CAP\_POWERSUPPLY.

Page 8-79 (PDF page 377)

2.1 CAP\_POWERSAVETIME 0x115F

Twain.h

```
#define CAP_POWERSAVETIME 0x115f /* Added 2.1 */
```

---

## ICAP\_AUTOMATICCOLORENABLED

Page 10-96 (PDF page 518)

### Values

Type:	TW_BOOL
Default Value:	FALSE
Allowed Values:	TRUE, FALSE
Container for <i>MSG_GET</i> :	TW_ENUMERATION, TW_ONEVALUE
Container of <i>MSG_SET</i> :	TW_ENUMERATION, TW_ONEVALUE
Container for <i>MSG_QUERY SUPPORT</i> :	TW_ONEVALUE

---

## ICAP\_BITDEPTH

Page 10-111 (PDF page 533)

### Description

Specifies the pixel bit depths for the Current value of ICAP\_PIXELTYPE.

For example;

ICAP\_PIXELTYPE = TWPT\_GRAY, this capability specifies whether this is 4-bit gray or 8-bit gray

ICAP\_PIXELTYPE = TWPT\_RGB, this capability specifies whether this is 24-bit color or 48-bit color

This depth applies to the total of all the data channels. TW\_IMAGEINFO BitsPerSample is used to identify the number of bits in each channel.

Page 4-8 (PDF page 82)

### Depth of the Pixels (in bits)

A pixel type such as TWPT\_BW allows only 1 bit per pixel (either black or white). The other pixel types may allow a variety of bits per pixel (4-bit or 8-bit gray, 24-bit or 48-bit color). Be sure to set the ICAP\_PIXELTYPE first, then set the ICAP\_BITDEPTH.

---

## ICAP\_EXTIMAGEINFO

Page 10-124 (PDF page 546)

### 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`, 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/manufacture 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/manufacture-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 `DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET`. This may improve performance, since the Source is not required to collect that information from the device. Set this capability to `TRUE` if using `DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET` to ensure all `TWEI_` items are available.

---

## ICAP\_FRAMES

Pages 10-131 (PDF page 553)

### Application

`MSG_GET` returns the size and location of all the frames the Source will acquire image data from when acquiring from each page.

`MSG_GETCURRENT` returns the size and location of the next frame to be acquired.

`MSG_SET` allows the application to specify the frames and their locations to be used to acquire from future pages. If the application isn't interested in setting the origin of the image, set both Top and Left to zero.

Defines the Left, Top, Right, and Bottom coordinates (in `ICAP_UNITS`) of the rectangle enclosing the original image on the original scanner. This ICAP is most useful if the Source supports simultaneous acquisition from multiple frames. Use `ICAP_MAXFRAMES` to establish this ability.

---

## ICAP\_ICCPROFILE

Page 10-135 (PDF page 557)

### Values

<b>Type:</b>	TW_UNIT16
<b>Default Value:</b>	No Default
<b>Allowed Value:</b>	TWIC_NONE, TWIC_EMBED, TWIC_LINK
<b>Container for MSG_GET:</b>	TW_ENUMERATION, TW_ONEVALUE
<b>Container for MSG_SET:</b>	TW_ONEVALUE
<b>Container for MSG_QUERY SUPPORT:</b>	TW_ONEVALUE

---

## ICAP\_ORIENTATION

Pages 10-155 (PDF page 577)

- ~~TWOR\_AUTOxxxxx values were introduced in 2.0 but should not be used with ICAP\_ORIENTATION.~~

### Application

~~TWOR\_ROT0 == TWOR\_PORTRAIT and TWOR\_ROT270 == TWOR\_LANDSCAPE.~~

~~TWOR\_AUTO orients the image according to criteria determined by the source. TWOR\_AUTOTEXT orients the document using text only algorithms. TWOR\_AUTOPICTURE orients the document using image only algorithms.~~

### Values

**Allowed Values:**

TWOR_ROT0	
TWOR_ROT90	
TWOR_ROT180	
TWOR_ROT270	
TWOR_PORTRAIT	(equals TWOR_ROT0)
TWOR_LANDSCAPE	(equals TWOR_ROT270)
<del>TWOR_AUTO</del>	<del>// 2.0 and higher</del>
<del>TWOR_AUTOTEXT</del>	<del>// 2.0 and higher</del>
<del>TWOR_AUTOPICTURE</del>	<del>// 2.0 and higher</del>

### Description

Defines which edge of the “paper” the image’s “top” is aligned with. This information is used to adjust the frames to match the scanning orientation of the paper. For instance, if an ICAP\_SUPPORTEDSIZE of TWSS\_ISO4 has been negotiated, and ICAP\_ORIENTATION is set to TWOR\_LANDSCAPE, then the Source must rotate the frame it downloads to the scanner to reflect the orientation of the paper.

- ICAP\_ORIENTATION affects the values reported by ICAP\_FRAMES when using ICAP\_SUPPORTEDSIZES.
- ICAP\_ORIENTATION is ignored when set using ICAP\_FRAMES or DAT\_IMAGELAYOUT.

Pages A-26 Frame Management (PDF page 652)

ICAP\_ORIENTATION is intended to tell a Source the orientation of a page in the scanner.

ICAP\_ROTATION is a specific request to the scanner to rotate the scanned image the indicated number of degrees. ~~ICAP\_ORIENTATION with ICAP\_SUPPORTEDSIZES will affect ICAP\_FRAMES and DAT\_IMAGELAYOUT. ICAP\_ROTATION should only affect the output from DAT\_IMAGEINFO.~~ The reason for negotiating these values after establishing the frame is that some Sources may reject attempts to rotate data if one of the dimensions exceeds the physical width or height of the scanner.

---

## ICAP\_SUPPORTEDEXTIMAGEINFO

Page 10-179 (PDF page 601)

Replace second paragraph under the Application section.

For instance, if the Source supports ICAP\_BARCODEDETECTIONENABLED, then it may report TW\_EI\_BARCODETEXT as part of this capability. However, if the image that was just captured has no barcode data, or if ICAP\_BARCODEDETECTIONENABLED was disabled, then the Source can return TWRC\_DATANOTAVAILABLE or TWRC\_INFONOTSUPPORTED for that TW\_INFO field, when the Application calls DAT\_EXTIMAGEINFO.

---

## Capability Ordering

Pages A-27 through A-30 (PDF page 650-656)

Insert the Capability Ordering flowchart and text from the current white paper.

---

## PDF Cross References

Make sure the cross-references are resolved when building the PDF file

---

## Appendix A Capability Default-Values Table

Page A-33 (PDF page 659-661)

CAP_CAMERASIDE	Mandatory	TWCS_BOTH
CAP_DUPLEXENABLED	Preferred / User	No default
CAP_FEEDERALIGNMENT	n/a	No default

Page A-34

- Add the following item

ICAP_AUTOMATICCOLORENABLED	n/a	FALSE
----------------------------	-----	-------

Page A-35

- Add the following item

ICAP_ICCPROFILE	n/a	No default
-----------------	-----	------------

---

## Internationalization

Add the Internationalization content from the TWAIN 2.0 Specification as an Appendix in the TWAIN 2.2 specification.

## Internationalization

A TWAIN Source can easily be internationalized despite its 8-bit character interface. A well designed Source should automatically match the locale of the application calling it; passing localized data through the API, and displaying appropriate language text in its user interface. Developers have the option of using UNICODE or MultiByte encodings, the 8-bit interface is not an obstacle to Applications or Sources.

When an Application calls `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`, it provides to the Source its `TW_IDENTITY` data. Internationalized Sources should check the `appIdentity->Version.Language` field, and attempt to match the Application's language (returning the same value in the `dsIdentity` structure). If the Source is incapable of matching the language, then it should attempt to match the User's current locale (on Win32 do this using the `LOCALE_USER_DEFAULT` value returned by the `GetLocaleInfo()` call). In most cases the Application locale and the User locale will be the same, and the Source will have to select the best language it can. For instance, if the Application requested Swiss French, and the Source only has French, then it should offer that. Otherwise, it should resort to some common secondary language, such as English.

Please note that `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS` is the very first opportunity that an Application and Source have to negotiate language. `DG_CONTROL / DAT_IDENTITY / MSG_GET`, when invoked in state 3, does not provide an `appIdentity`. Sources should default to the `LOCALE_USER_DEFAULT` in this instance.

As mentioned above, the TWAIN interface assumes 8-bit characters, this prevents the direct passing of UNICODE data between Sources and Applications, but it does not hinder indirect means that convert data into MultiByte encodings. The remainder of this section shows one way of allowing Sources and Applications to communicate, without worrying about whether they are UNICODE or MultiByte enabled. The best example to illustrate this is to consider a Source and Application, both UNICODE enabled, communicating through the TWAIN interface.

To pass UNICODE string data from the Source to the Application, the Source must convert UNICODE to MultiByte, using the appropriate Code-Page (which is specific to a given set of locales). When the Application receives the data, it converts from MultiByte back to UNICODE.

The process is the same when sending string data from the Application to the Source. The process depends on the Application and Source using the same Code-Page for their conversion. The Win32 functions required to perform the conversions are `WideCharToMultiByte` and `MultiByteToWideChar`. The only limitation to watch out for is the size of the various strings provided by TWAIN. At all times the MultiByte data must fit within the strings described by the interface, and Source and Application writers need to pay close attention to it.

```
int WideCharToMultiByte(  
    UINT CodePage,           // code page  
    DWORD dwFlags,          // performance and mapping flags  
    LPCWSTR lpWideCharStr,  // address of wide-character string  
    int cchWideChar,        // number of characters in string  
    LPSTR lpMultiByteStr,   // address of buffer for new string  
    int cchMultiByte,       // size of buffer
```

```

LPCSTR IpDefaultChar,      // address of default for unmappable characters
LPBOOL IpUsedDefaultChar  // address of flag set when default char. used
);
int MultiByteToWideChar(
  UINT CodePage,          // code page
  DWORD dwFlags,         // character-type options
  LPCSTR IpMultiByteStr,  // address of string to map
  int cchMultiByte,       // number of characters in string
  LPWSTR IpWideCharStr,   // address of wide-character buffer
  int cchWideChar         // size of buffer
);

```

These functions are fully described in the online Microsoft Visual C++ documentation. This section does not attempt to duplicate that information, but does show how Source and Application may cooperate when using them to transmit localized data through the TWAIN interface.

### **TWAIN CAP\_LANGUAGE Code to ANSI Code-Page Table**

```

// This array maps TWAIN CAP_LANGUAGE codes to the appropriate ANSI Code-
// Page. There is no mechanism for converting to the OEM Code-Page, nor
// should one be needed, since the upper 128 bytes in the OEM pages mostly
// contain line art characters used by MS-DOS.
// Note: the index in the comment field is just an index into the array,
// it does not correspond to the TWAIN constant for a given TWLG field...
//
#define AnsiCodePageElements 88
int AnsiCodePage[AnsiCodePageElements] = {
    1252, // 0 TWLG_DANISH (TWLG_DAN)
    1252, // 1 TWLG_DUTCH (TWLG_DUT)
    1252, // 2 TWLG_ENGLISH (TWLG_ENG)
    1252, // 3 TWLG_FRENCH_CANADIAN (TWLG_FCF)
    1252, // 4 TWLG_FINNISH (TWLG_FIN)
    1252, // 5 TWLG_FRENCH (TWLG_FRN)
    1252, // 6 TWLG_GERMAN (TWLG_GER)
    1252, // 7 TWLG_ICELANDIC (TWLG_ICE)
    1252, // 8 TWLG_ITALIAN (TWLG_ITN)
    1252, // 9 TWLG_NORWEGIAN (TWLG_NOR)
    1250, // 10 TWLG_PORTUGUESE (TWLG_POR)
    1252, // 11 TWLG_SPANISH (TWLG_SPA)
    1252, // 12 TWLG_SWEDISH (TWLG_SWE)
    1252, // 13 TWLG_ENGLISH_USA (TWLG_USA)
    1252, // 14 TWLG_AFRIKAANS
    1250, // 15 TWLG_ALBANIA

```

1256, // 16 TWLG\_ARABIC  
1256, // 17 TWLG\_ARABIC\_ALGERIA  
1256, // 18 TWLG\_ARABIC\_BAHRAIN  
1256, // 19 TWLG\_ARABIC\_EGYPT  
1256, // 20 TWLG\_ARABIC\_IRAQ  
1256, // 21 TWLG\_ARABIC\_JORDAN  
1256, // 22 TWLG\_ARABIC\_KUWAIT  
1256, // 23 TWLG\_ARABIC\_LEBANON  
1256, // 24 TWLG\_ARABIC\_LIBYA  
1256, // 25 TWLG\_ARABIC\_MOROCCO  
1256, // 26 TWLG\_ARABIC\_OMAN  
1256, // 27 TWLG\_ARABIC\_QATAR  
1256, // 28 TWLG\_ARABIC\_SAUDIARABIA  
1256, // 29 TWLG\_ARABIC\_SYRIA  
1256, // 30 TWLG\_ARABIC\_TUNISIA  
1256, // 31 TWLG\_ARABIC\_UAE /\* United Arabic Emirates \*/  
1256, // 32 TWLG\_ARABIC\_YEMEN  
1252, // 33 TWLG\_BASQUE  
1251, // 34 TWLG\_BYELORUSSIAN  
1251, // 35 TWLG\_BULGARIAN  
1252, // 36 TWLG\_CATALAN  
936, // 37 TWLG\_CHINESE  
950, // 38 TWLG\_CHINESE\_HONGKONG  
936, // 39 TWLG\_CHINESE\_PRC /\* People's Republic of China \*/  
936, // 40 TWLG\_CHINESE\_SINGAPORE  
936, // 41 TWLG\_CHINESE\_SIMPLIFIED  
950, // 42 TWLG\_CHINESE\_TAIWAN  
950, // 43 TWLG\_CHINESE\_TRADITIONAL  
1250, // 44 TWLG\_CROATIA  
1250, // 45 TWLG\_CZECH  
1252, // 46 TWLG\_DUTCH\_BELGIAN  
1252, // 47 TWLG\_ENGLISH\_AUSTRALIAN  
1252, // 48 TWLG\_ENGLISH\_CANADIAN  
1252, // 49 TWLG\_ENGLISH\_IRELAND  
1252, // 50 TWLG\_ENGLISH\_NEWZEALAND  
1252, // 51 TWLG\_ENGLISH\_SOUTHAFRICA  
1252, // 52 TWLG\_ENGLISH\_UK  
1257, // 53 TWLG\_ESTONIAN  
1250, // 54 TWLG\_FAEROESE  
1256, // 55 TWLG\_FARSI  
1252, // 56 TWLG\_FRENCH\_BELGIAN  
1252, // 57 TWLG\_FRENCH\_LUXEMBOURG

```

1252, // 58 TWLG_FRENCH_SWISS
1252, // 59 TWLG_GERMAN_AUSTRIAN
1252, // 60 TWLG_GERMAN_LUXEMBOURG
1252, // 61 TWLG_GERMAN_LIECHTENSTEIN
1252, // 62 TWLG_GERMAN_SWISS
1253, // 63 TWLG_GREEK
1255, // 64 TWLG_HEBREW
1250, // 65 TWLG_HUNGARIAN
1252, // 66 TWLG_INDONESIAN
1252, // 67 TWLG_ITALIAN_SWISS
932, // 68 TWLG_JAPANESE
949, // 69 TWLG_KOREAN
1361, // 70 TWLG_KOREAN_JOHAB
1257, // 71 TWLG_LATVIAN
1257, // 72 TWLG_LITHUANIAN
1252, // 73 TWLG_NORWEGIAN_BOKMAL
1252, // 74 TWLG_NORWEGIAN_NYNORSK
1250, // 75 TWLG_POLISH
1252, // 76 TWLG_PORTUGUESE_BRAZIL
1250, // 77 TWLG_ROMANIAN
1251, // 78 TWLG_RUSSIAN
1250, // 79 TWLG_SERBIAN_LATIN
1250, // 80 TWLG_SLOVAK
1250, // 81 TWLG_SLOVENIAN
1252, // 82 TWLG_SPANISH_MEXICAN
1252, // 83 TWLG_SPANISH_MODERN
874, // 84 TWLG_THAI
1254, // 85 TWLG_TURKISH
1251, // 86 TWLG_UKRANIAN
};

```

### Sample Converting from WideChar to MultiByte

The following is a sample of converting from WideChar to MultiByte.

```

// This function converts _TCHAR* strings to MultiByte, using the
// appropriate code page. If the build is ANSI or MBCS, then no
// conversion is needed, the _tcsncpy() function is used.
// If the build is UNICODE, then the Code-Page is determined, and used to
// convert the string to MultiByte using the WideCharToMultiByte()
// function...
//
int CopyTCharToMultibyte
(char *dst,
const int sizeof_dst,

```

```

    const _TCHAR *src,
    const int twain_language_code)
{
#ifdef _UNICODE
    // MultiByte string copy...
    _tcsncpy(dst,src,sizeof_dst);
    dst[sizeof_dst-1] = 0;
    return(strlen(dst));
#else
    int cp;
    int len;
    _TCHAR cp_str[16];
    if (twain_language_code >= AnsiCodePageElements) {
        // Whoops, don't have one of those...
        return(-1);
    } else if (twain_language_code >= 0) {
        // Lookup the code page...
        cp = AnsiCodePage[twain_language_code];
    } else {
        // Get the User's code page...
        GetLocaleInfo
            (LOCALE_USER_DEFAULT,
            LOCALE_IDEFAULTANSICODEPAGE,
            cp_str,
            sizeof(cp_str));
        cp = _ttoi(cp_str);
    }
    if (IsValidCodePage(cp) == 0) {
        // That code page isn't installed on this system...
        return(-1);
    }
    len = WideCharToMultiByte(
        cp, // code page
        0, // performance and mapping flags
        src, // address of wide-character string
        -1, // number of characters in string
        dst, // address of buffer for new string
        sizeof_dst, // size of buffer (in characters)
        NULL, // address of default for unmappable characters
        NULL // address of flag set when default char. used
    );
#endif
}

```

```
}
```

## Sample Converting from MultiByte to WideChar

The following is a sample of converting from MultiByte to WideChar.

```
// This function converts multibyte strings to _TCHAR* strings, using
// the appropriate code page.
// If the build is ANSI or MBCS, then no conversion is needed, the
// _tcsncpy() function is used. If the build is UNICODE, then the
// Code-Page is determined, and used to convert the string to
// _TCHAR* using the MultiByteToWideChar() function...
//
int CopyMultibyteToTChar
    (_TCHAR *dst,
     const int sizeof_dst,
     const char *src,
     const int twain_language_code)
{
#ifdef _UNICODE
    // MultiByte string copy...
    _tcsncpy(dst,src,sizeof_dst);
    dst[sizeof_dst-1] = 0;
    return(strlen(dst));
#else
    int cp;
    int len;
    _TCHAR cp_str[16];
    if (twain_language_code >= AnsiCodePageElements) {
        // Whoops, don't have one of those...
        return(-1);
    } else if (twain_language_code >= 0) {
        // Lookup the code page...
        cp = AnsiCodePage[twain_language_code];
    } else {
        // Get the User's code page...
        GetLocaleInfo
            (LOCALE_USER_DEFAULT,
             LOCALE_IDEFAULTANSICODEPAGE,
             cp_str,
             sizeof(cp_str));
        cp = _ttoi(cp_str);
    }
    if (IsValidCodePage(cp) == 0) {
        // That code page isn't installed on this system...
```

```

        return(-1);
    }
    len = MultiByteToWideChar(
        cp, // code page
        0, // performance and mapping flags
        src, // address of wide-character string
        -1, // number of characters in string
        dst, // address of buffer for new string
        sizeof_dst/sizeof(_TCHAR) // size of buffer (in characters)
    );
    return(len);
#endif
}

```

### Sample Use of the Conversion Functions

The following are examples of UNICODE application and UNICODE source.

#### UNICODE Application

```

int sts;
int twain_language_code;
_TCHAR Author[128];
pTW_ONEVALUE pvalOneValue;
. . .
// the Application has queried the Source as to what languages it supports
//and selected TWLG_JAPANESE, storing it in twain_language_code..
. . .
// CAP_AUTHOR is queried, and a value is received..
. . .
// Convert CAP_AUTHOR string to UNICODE...
sts = CopyMultiByteToTChar
    (Author,
    sizeof(Author),
    (char*)&pvalOneValue->Item,
    twain_language_code)
if (sts < 0) {
    // Error...
. . .
}

```

#### UNICODE Source

```

. . .
int sts;
int source_language_code;
_TCHAR SourceAuthor[128];

```

```

pTW_ONEVALUE pvalOneValue;
. . .
// the Source has been told to use TWLG_JAPANESE, it stores this value
// in source_language_code ...
. . .
// CAP_AUTHOR is queried by the Application...
// The Source keeps the value in SourceAuthor...
. . .
// Convert CAP_AUTHOR string to multibyte...
    sts = CopyTCharToMultibyte
        ((char*)&pvalOneValue->Item,
        sizeof(TW_STR128),
        SourceAuthor,
        source_language_code)
if (sts < 0) {
    // Error...
    . . .
}
. . .
// The Source returns the value to the Application...

```

## The ImageData and Its Layout

Pages 4-24 to 4-25 (PDF page 98-99)

The image which is transferred from the Source to the application has several attributes. Some attributes describe the size of the image. Some describe where the image was located on the **scanner**. Still others might describe information such as resolution or number of bits per pixel. TWAIN provides means for the application to learn about these attributes. Users are often able to select and modify an image's attributes through the Source's user interface. Additionally, TWAIN provides capabilities and operations that allow the application to impact these attributes prior to acquisition and transfer.

### Getting Information About the Image That will be Transferred

Before the transfer occurs, while in State 6, the Source can provide information to the application about the actual image that it is about to transfer. Note, the information is lost once the transfer takes place so the application should save it, if needed. This information can be retrieved through two operations:

- DG\_IMAGE / DAT\_IMAGELAYOUT / MSG\_GET
- DG\_IMAGE / DAT\_IMAGEINFO / MSG\_GET

The area of an image to be acquired will always be a rectangle called a frame. There may be one or more frames located on a page. Frames can be selected by the user or designated by the application. The TW\_IMAGELAYOUT structure communicates where the image was located on the original page relative to the origin of the **scanner**. It also indicates, in its FrameNumber field, if this is the first frame, or a later frame, to be acquired from the page.

The TW\_IMAGELAYOUT structure looks like this:

```
typedef struct {
    TW_FRAME Frame;
    TW_UINT32 DocumentNumber;
    TW_UINT32 PageNumber;
    TW_UINT32 FrameNumber;
} TW_IMAGELAYOUT, FAR *pTW_IMAGELAYOUT;
```

The TW\_FRAME structure specifies the values for the Left, Right, Top, and Bottom of the frame to be acquired **based on the origin of the scanner**. Values are given in ICAP\_UNITS.

### Update image to have "Origin of Scanner"

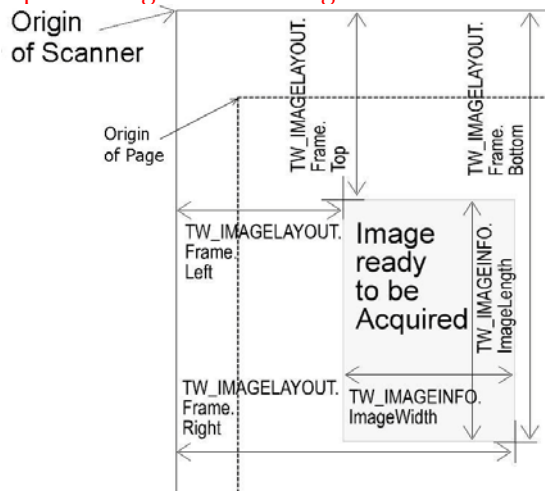


Figure 4-1. TW\_FRAME Structure

The DG\_IMAGE / DAT\_IMAGEINFO / MSG\_GET operation communicates other attributes of the image being transferred. The TW\_IMAGEINFO structure looks like this:

```
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_UINT16 Compression;
} TW_IMAGEINFO, FAR * pTW_IMAGEINFO;
```

The ImageWidth and ImageLength relate to the frame described by the TW\_IMAGELAYOUT structure after ICAP\_ROTATION is taken into account.

Pages 4-26

```
ImageWidth TW_IMAGELAYOUT.TW_FRAME.Right - TW_FRAME.Left **
ImageLength TW_IMAGELAYOUT.TW_FRAME.Bottom - TW_FRAME.Top **
```

\*\*ImageWidth and ImageLength are actually provided in pixels whereas TW\_FRAME uses ICAP\_UNITS. If ICAP\_ROTATION is 90 or -90 then ImageWidth and ImageLength are exchanged.

**Note:** Frame extents are only limited by ICAP\_PHYSICALWIDTH and ICAP\_PHYSICALHEIGHT. Setting ICAP\_SUPPORTEDSIZES does NOT imply a new extent limitation. TWSS\_xxxx sizes combined with ICAP\_ORIENTATION are simply predefined fixed frame sizes.

- If the frame is set in DAT\_IMAGELAYOUT
  - ICAP\_FRAMES shall respond to MSG\_GETCURRENT with the dimensions of the frame set in the DAT\_IMAGELAYOUT call.
  - ICAP\_SUPPORTEDSIZES shall respond to MSG\_GETCURRENT with TWSS\_NONE
- If the current frame is set from ICAP\_FRAMES
  - DAT\_IMAGELAYOUT shall respond with the dimensions of the current frame set in ICAP\_FRAMES
  - ICAP\_SUPPORTEDSIZES shall respond to MSG\_GETCURRENT with TWSS\_NONE
- If the current fixed frame is set from ICAP\_SUPPORTEDSIZES
  - DAT\_IMAGELAYOUT shall respond to MSG\_GETCURRENT with the dimensions of the fixed frame specified in ICAP\_SUPPORTEDSIZES combined with ICAP\_ORIENTATION.
  - ICAP\_FRAMES shall respond to MSG\_GETCURRENT with the dimensions of the fixed frame specified in ICAP\_SUPPORTEDSIZES combined with ICAP\_ORIENTATION.

### ICAP\_ROTATION, ICAP\_ORIENTATION Affect on ICAP\_FRAMES, DAT\_IMAGELAYOUT, DAT\_IMAGEINFO

~~There is considerable confusion when trying to resolve the affect of Rotation and Orientation on the current frames and image layout. After careful consideration of the specification it has been concluded that ICAP\_ROTATION and ICAP\_ORIENTATION shall be applied after considering ICAP\_FRAMES and DAT\_IMAGELAYOUT.~~

Obviously a change in orientation will have an effect on the output image dimensions, so these must be reflected in DAT\_IMAGEINFO during State 6. The resulting image dimensions shall be reported by the data source after considering the affect of the rotation on the current frame.

ICAP\_ORIENTATION shall be reflected in returned ICAP\_FRAMES and DAT\_IMAGELAYOUT when set using ICAP\_SUPPORTEDSIZES other than TWSS\_NONE or TWSS\_MAXSIZE.  
ICAP\_ROTATION shall only be reflected in the returned image data of DAT\_IMAGEINFO.

ICAP\_ORIENTATION and ICAP\_ROTATION are additive. The original SupportedSize is modified by ICAP\_ORIENTATION as it is downloaded to the device by the Source, and represents the orientation of the paper being scanned. ICAP\_ROTATION is then applied to the captured image to yield the final framing information that is reported to the Application in State 6 or 7. One possible reason for combining these two values is to use them to cancel each other out. For instance, some scanners with automatic document feeders may receive a performance benefit from describing an ICAP\_ORIENTATION of TWOR\_LANDSCAPE in combination with an ICAP\_ROTATION of 90 degrees. This would allow the user to feed images in a landscape orientation (which lets them feed faster), while rotating the captured images back to portrait (which is the way the user wants to view them).

---

## DG\_IMAGE / DAT\_IMAGELAYOUT / MSG\_GET

Pages 7-122 (PDF page 270)

### Description

The DAT\_IMAGELAYOUT operations control information on the physical layout of the image on the acquisition platform of the Source (e.g. the glass of a flatbed scanner, the size of a photograph, etc.).

The MSG\_GET operation describes both the size and placement of the image on the scanner. The coordinates on the image and the extents of the image are expressed in the units of measurement currently negotiated for ICAP\_UNITS (default is inches).

The outline of the image is expressed by a “frame.” The Left, Top, Right, and Bottom edges of the frame are stored in pImageLayout->Frame. These values place the frame within the scanner. All measurements are relative to the scanner’s “upper-left” corner. Define “upper-left” by how the image would appear on the computer’s screen before any rotation or other position transform is applied to the image data. This origin point will be apparent for most Sources (although folks working with satellites or radio telescopes may be at a bit of a loss).

Finally pImageLayout optionally includes information about which frame on the page, which page within a document, and which document the image belongs to. These fields were included mostly for future versions which could merge more than one type of data. A more immediate use might be for an application that needs to keep track of which frame on the page an image came from while acquiring from a Source that can supply more than one image from the same page at the same time. The information in this structure always describes the current image. To set multiple frames for any page simultaneously, reference ICAP\_FRAMES.

---

## DG\_IMAGE / DAT\_IMAGELAYOUT / MSG\_SET

Pages 7-126 (PDF page 274)

### Application

Fill in all fields of `pImageLayout`. Especially important is the `Frame` field whose values are expressed in `ICAP_UNITS`. If the application doesn't care about one or more of the other fields, be sure to set them to -1 to prevent confusion. If the application only cares about the extents of the `Frame`, and not about the origin on the page, set the `Frame.Top` and `Frame.Left` to zero. Otherwise, the application can specify the location on the `scanner` where the `Source` should begin acquiring the image, in addition to the extents of the acquired image.

### Source

Use the values in `pImageLayout` as the `Source`'s current image layout information. If you are unable to set the device exactly to the values requested in the `Frame` field, set them as closely as possible, always snapping to a value that will result in a larger frame, and return `TWRC_CHECKSTATUS` to the application.

If the application sets `Frame.Top` and `Frame.Left` to zero, then the `Source` should set the frame taking into consideration the default alignment set through `CAP_FEEDERALIGNMENT`.

If the application has set `Frame.Top` and `Frame.Left` to a non-zero value, set the origin for the image to be acquired accordingly. If possible, the `Source` should consider reflecting these settings in the user interface when it is raised. For instance, if your `Source` presents a pre-scan image, consider showing the selection region in the proper location and with the proper size suggested by the settings from this operation.

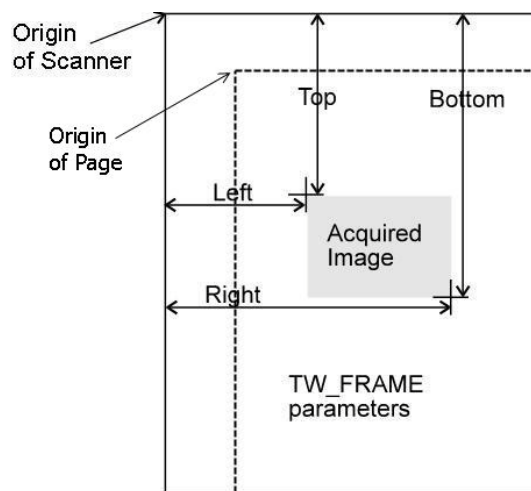
If the requested values exceed the maximum size the `Source` can acquire, set the `pImageLayout->Frame` values used within the `Source` to the largest extent possible within the axis of the offending value. Return `TWRC_FAILURE` with `TWCC_BADVALUE`.

---

## TW\_FRAME

Pages 8-33 (PDF page 331)

- Replace image with one that uses origin of scanner.



Frame Parameters

---

## TW\_IMAGELAYOUT

Pages 8-40 (PDF page 338)

### Description

Involves information about the original size of the acquired image and its position on the **scanner** relative to the **scanner's** upper-left corner. Default measurements are in inches (units of measure can be changed by negotiating the ICAP\_UNITS capability). This information may be used by the application to relate the acquired (and perhaps processed image) to the original. Further, the application can, using this structure, set the size of the image it wants acquired.

### Field Descriptions

**Frame** Defines the Left, Top, Right, and Bottom coordinates (in ICAP\_UNITS) of the rectangle enclosing the original image on the **scanner**. If the application isn't interested in setting the origin of the image, set both Top and Left to zero. The Source will fill in the actual values following the acquisition. See also TW\_FRAME.

---

---

## TWMF\_DSOWNS

Pages 7-116 (PDF page 264)

### Source

Allocates the TheMem member and sets the Flags member to have **TWFM\_DSOWNS** **TWMF\_DSOWNS**. Fills in the Length member.

It is recommended that sources obey platform specific rules about locations for profile files. When possible, it is desirable to store the profiles in the platform specific location and then to read that profile and send the data back to the location.

---

## NativeTransfer

Pages 3-30 (PDF page 62)

### pData

Points to a ~~TW\_UINT32~~ TW\_HANDLE variable. This is an exception from the typical pattern.

Pages 6-6 (PDF page 146)

---

Data Type	Used by	Associated structure or type
DAT_IMAGENATIVEXFER	DG_IMAGE	<del>TW_UINT32</del> TW_HANDLE On Windows - <del>low word</del> =DIB handle On Macintosh - PicHandle
....	...	...

---

Pages 7-134 (PDF page 282)

pHandle = A pointer to a variable of type ~~TW\_UINT32~~ TW\_HANDLE.

**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.

Pages 7-135 (PDF page 283)

### 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~~ TW\_HANDLE pointed to by pHandle.

Pages 8-63 (PDF page 361)

### DAT\_IMAGENATIVEXFER

Uses a ~~TW\_UINT32~~ TW\_HANDLE variable.

---

## Wrong Capability IDs

Pages 8-74 (PDF page 372)

---

Version	Constant	Numeric ID
	CAP_CUSTOMBASE	0x8000
	CAP_XFERCOUNT	0x0001
	ICAP_COMPRESSION	0x01010x0100
	ICAP_PIXELTYPE	0x01020x0101
	ICAP_UNITS	0x01030x0102
	ICAP_XFERMECH	0x01040x0103
	...	...

---

Pages 8-76 (PDF page 374)

---

Version	Constant	Numeric ID
	...	...
	CAP_PRINTER	0x10240x1026
	CAP_PRINTERENABLED	0x10260x1027
	CAP_PRINTERINDEX	0x10270x1028
	CAP_PRINTERMODE	0x10280x1029
	CAP_PRINTERSTRING	0x10290x102A
	CAP_PRINTERSUFFIX	0x102A0x102B
	CAP_LANGUAGE	0x102B0x102C
	CAP_FEEDERALIGNMENT	0x102C0x102D
	CAP_FEEDERORDER	0x102D0x102E
	...	...

---

Pages 8-78 (PDF page 376)

---

Version	Constant	Numeric ID
	...	...
	ICAP_BITDEPTH	0x112B
	ICAP_UNDEFINEDIMAGESIZE	0x112C
	ICAP_IMAGEDATASET	0x112D
	ICAP_EXTIMAGEINFO	0x112E
	ICAP_MINIMUMHEIGHT	0x112F
	ICAP_MINIMUMWIDTH	0x1130
	ICAP_FLIPROTATION	0x1131
	ICAP_AUTODISCARDBLANKPAGES	0x1134
	ICAP_BARCODEDETECTIONENABLED	0x1136
	ICAP_SUPPORTEDBARCODETYPES	0x1137
	ICAP_BARCODEMAXSEARCHPRIORITIES	0x1138
	ICAP_BARCODESEARCHPRIORITIES	0x1139

---

<del>ICAP_BARCODESEARCHMODE</del>	0x113A
<del>ICAP_BARCODEMAXRETRIES</del>	0x113B
<del>ICAP_BARCODETIMEOUT</del>	0x113C
<del>ICAP_ZOOMFACTOR</del>	0x113D
<del>ICAP_BITDEPTHREDUCTION</del>	0x113E
ICAP_BITDEPTHREDUCTION	0x112C
ICAP_UNDEFINEDIMAGESIZE	0x112D
ICAP_IMAGEDATASET	0x112E
ICAP_EXTIMAGEINFO	0x112F
ICAP_MINIMUMHEIGHT	0x1130
ICAP_MINIMUMWIDTH	0x1131
ICAP_AUTODISCARDBLANKPAGES	0x1134
ICAP_FLIPROTATION	0x1136
ICAP_BARCODEDETECTIONENABLED	0x1137
ICAP_SUPPORTEDBARCODETYPES	0x1138
ICAP_BARCODEMAXSEARCHPRIORITIES	0x1139
ICAP_BARCODESEARCHPRIORITIES	0x113A
ICAP_BARCODESEARCHMODE	0x113B
ICAP_BARCODEMAXRETRIES	0x113C
ICAP_BARCODETIMEOUT	0x113D
ICAP_ZOOMFACTOR	0x113E
ICAP_PATCHCODEDETECTIONENABLED	0x113F
...	...

---

---

## Wrong Names

Pages 8-75 (PDF page 373)

---

Version	Constant	Numeric ID
...	<del>CAP_UICONTROLABLE</del> -CAP_UICONTROLLABLE	...
...		0x100E
...		...

---

Pages 8-75 (PDF page 373)

---

Version	Constant	Numeric ID
...	<del>CAP_DUPLEXENABLED</del> -CAP_DUPLEXENABLED	...
...		0x1013
...		...

---

Pages 8-75 (PDF page 373)

---

Version	Constant	Numeric ID
...	<del>CAP_CUSTOMEDSDATAD</del> -CAP_CUSTOMDSDATA	...
...		0x1015
...		...

---

Pages 8-76 (PDF page 374)

---

Version	Constant	Numeric ID
...	<del>CAP_DEVICEDATETIME</del> -CAP_DEVICETIMEDATE	...
...		0x101F
...		...

---

Pages 8-78 (PDF page 376)

---

Version	Constant	Numeric ID
...	<del>ICAP_PATCHCODEMAXSEARCHPRIORITIE</del> ICAP_PATCHCODEMAX SEARCHPRIORITIES	...
...		0x1141
...		...

---

Pages 8-90 (PDF page 388)

Version	Constant	Numeric ID
	...	...
	TWSS_C10	51
	<del>TWSS_USEEXECUTIVE</del> TWSS_USSTATEMENT	52
	TWSS_BUSINESSCARD	53
	TWSS_MAXSIZE	54

---

## Missing File Format description

*Page 10-144 Update Description section with text in red :*

TWFF_DEJAVU	A file format from LizardTech.
TWFF_PDFA	A file format from Adobe PDF/A, Version 1.
TWFF_PDFA2	A file format from Adobe PDF/A, Version 2.

---

## Missing constants

Pages 8-70 (PDF page 368)

---

Version	Constant	Numeric ID
	...	...
	MSG_PASSTHRU	0x0901
	MSG_REGISTER_CALLBACK	0x0902
	MSG_RESETALL	0x0A01

---

Pages 8-68 (PDF page 366)

---

Data Groups (DG_)	Numeric ID
DG_CONTROL	0x0001L
DG_IMAGE	0x0002L
DG_AUDIO	0x0004L
DG_MASK	0xFFFFL

---

Add this section on page 8-68 (366) :

### Triplet Constants

#### Data Flags (DF\_)

**Note:** These are bits in a mask.

---

Data Flags (DF_)	Numeric ID
DF_DSM2	0x10000000L
DF_APP2	0x20000000L
DF_DS2	0x40000000L

---

---

## Wrong constants

Pages 8-70 (PDF page 368)

---

Version	Constant	Numeric ID
	...	...
	MSG_ENDXFER	0x0701
	MSG_STOPFEEDER	<del>0x0701</del> 0x0702
	...	...

---