# TWAIN Errata

## For Version 2.1

July 28th, 2010

# Contents

## 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**

| | |
|---|---|
| MSG_CLOSEDS : | Prepare specified Source for unloading |
| MSG_GETDEFAULT : | Get identity information of the default Source |
| MSG_GETFIRST : | Get identity information of the first available Source |
| MSG_GETNEXT : | Get identity of the next available Source |
| MSG_OPENDS : | Load and initialize the specified Source |
| MSG_SET : | Set identity information of the default Source |
| MSG_USERSELECT : | 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 x 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

| | |
|---|---|
| **Allowed Values:** | TWSX_NATIVE<br>TWSX_FILE |
| **Container for *MSG_GET:*** | TW_ENUMERATION<br>TW_ONEVALUE |

Page 10-190 (PDF page 612)

## ICAP_XFERMECH

|  |  |
|---|---|
| **Allowed Values:** | TWSX_NATIVE<br>TWSX_FILE<br>TWSX_MEMORY<br>TWSX_MEMFILE |
| **Container for *MSG_GET:*** | TW_ENUMERATION<br>TW_ONEVALUE |

Page A-27 (PDF page 653)

### 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_CLOSEDSREQ 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_DISABLEDS 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.

Page 7-105 (PDF page 253)

**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 |
| ~~DG_IMAGE~~ | ~~DAT_IMAGEFILEXFER~~ | ~~MSG_GET~~ | ~~7-128~~ |
| DG_IMAGE | DAT_IMAGENATIVEXFER | MSG_GET | 7-134 |

## Operation Triplets – Source to Application

Page 7-4 (PDF page 152)

- Add page number for MSG_CLOSEDSOK

## 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, used by DSM to send `DAT_NULL/MSG_xxx` |
| `RefCon` | An application defined reference constant. Returned as _pData in callback. |

**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
```

## MSG_RESET / MSG_GETDEFAULT / MSG_RESETALL / APPENDIX A Defaults

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_CLOSEDSOK

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

### Call

DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_CLOSEDSOK, 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_CLOSEDSOK 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_CLOSEDSOK 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_DISABLEDS
```

## DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM

Page 7-79 (PDF page 227)

### Call

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

`pParent` should be the same value used in `MSG_OPENDSM`.

### 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_OPENDSM | 0x0301 |
| MSG_CLOSEDSM | 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 `pOrigin->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` When `DAT_XFERGROUP` is set to `DG_IMAGE`, 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.

When `DAT_XFERGROUP` is set to `DG_AUDIO`, 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.

`EOJ` The application should check this field if the `CAP_JOBCONTROL` is set to other than `TWJC_NONE`. If the EOJ is not 0, the application should expect more data from the driver according to `CAP_JOBCONTROL` settings.

The source should fill this value with one of the `TWEJ_xxx` patch codes if `CAP_JOBCONTROL` is set to other than `TWJC_NONE`.

`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_HORZLINEXCOORD, TWEI_HORZLINEYCOORD, TWEI_HORZLINELENGTH, TWEI_HORZLINETHICKNESS, TWEI_VERTLINEXCOORD, 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

**Description** The X coordinate in pixels 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. If this is set, then the Application should calculate any frames with a left offset.

## Chapter 10: DG_CONTROL / DAT_CAPABILITY / MSG_GET

Page 10-10 (PDF page 432)

### Containers

**MSG_GETCURRENT & MSG_GETDEFAULT:** Acceptable containers for use on MSG_GETCURRENT and MSG_GETDEFAULT operations.

***MSG_GET*** Acceptable containers for use on MSG_GET operations.

***MSG_RESET*** Acceptable containers for use on MSG_RESET operations.

***MSG_SET*** 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

| | |
|---|---|
| **Type:** | TW_BOOL |
| **Default Value:** | None |
| **Allowed Values:** | TRUE or FALSE |

### Containers

**MSG_GETCURRENT & MSG_GETDEFAULT:** TW_ONEVALUE

**MSG_GET:** TW_ONEVALUE, // for backwards compatibility with
1.x applications only

TW_ENUMERATION // mandatory for 2.1 application

and higher

**MSG_RESET:** Not allowed

**MSG_SET:** Not allowed

**MSG_QUERYSUPPORT:** 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 (simples 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_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 | <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>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

## 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_xxx patch codes of the PENDINGXFERS data.

## Rename CAP_POWERSAVETIME to CAP_POWERDOWNTIME

Page 10-67 (PDF page 489)

| | |
|---|---|
| **Type**: | TW_INT32 |
| **Default Value**: | No Default |
| **Allowed Values**: | >= -1 |
| **Container for *MSG_GET***: | TW_ONEVALUE |
| **Container for *MSG_SET***: | TW_ONEVALUE |
| **Container for *MSG_QUERYSUPPORT***: | 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** ***MSG_GET:*** | TW_ENUMERATION, TW_ONEVALUE |
| ***Container of MSG_SET:*** | TW_ENUMERATION, TW_ONEVALUE |
| ***Container for*** MSG_QUERYSUPPORT: | 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/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 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

| | |
|---|---|
| **Type:** | `TW_UNIT16` |
| **Default Value:** | `No Default` |
| **Allowed Value:** | `TWIC_NONE,`<br>`TWIC_EMBED,`<br>`TWIC_LINK` |
| **Container for *MSG_GET:*** | `TW_ENUMERATION,`<br>`TW_ONEVALUE` |
| **Container for *MSG_SET:*** | `TW_ONEVALUE` |
| **Container for *MSG_QUERYSUPPORT:*** | `TW_ONEVALUE` |

## ICAP_ORIENTATION

Pages 10-155 (PDF page 577)

- TWOR_AUTOxxxx 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) |
| | ~~TWOR_AUTO~~ | ~~// 2.0 and higher~~ |
| | ~~TWOR_AUTOTEXT~~ | ~~// 2.0 and higher~~ |
| | ~~TWOR_AUTOPICTURE~~ | ~~// 2.0 and higher~~ |

### 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_ISOA4 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 `TWEI_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 lpDefaultChar,        // address of default for unmappable characters
LPBOOL lpUsedDefaultChar     // address of flag set when default char. used
);
int MultiByteToWideChar(
UINT CodePage,               // code page
DWORD dwFlags,               // character-type options
LPCSTR lpMultiByteStr,       // address of string to map
int cchMultiByte,            // number of characters in string
LPWSTR lpWideCharStr,        // 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)
{
#ifndef _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 MuliByte 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)
    {
    #ifndef _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"

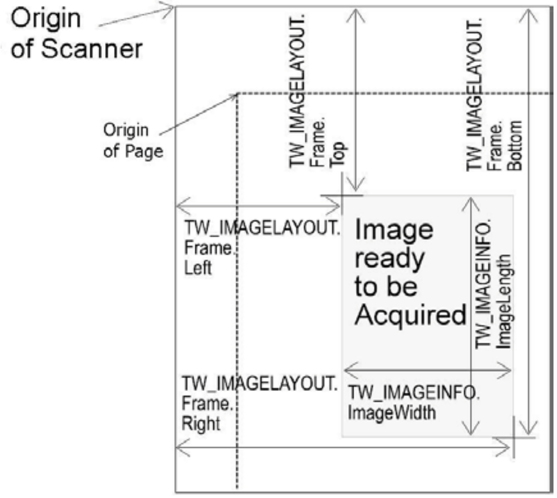

Origin of Scanner
Origin of Page
TW_IMAGELAYOUT. Frame. Top
TW_IMAGELAYOUT. Frame. Bottom
TW_IMAGELAYOUT. Frame. Left
Image ready to be Acquired
TW_IMAGEINFO. ImageLength
TW_IMAGELAYOUT. Frame. Right
TW_IMAGEINFO. ImageWidth

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_GET 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.

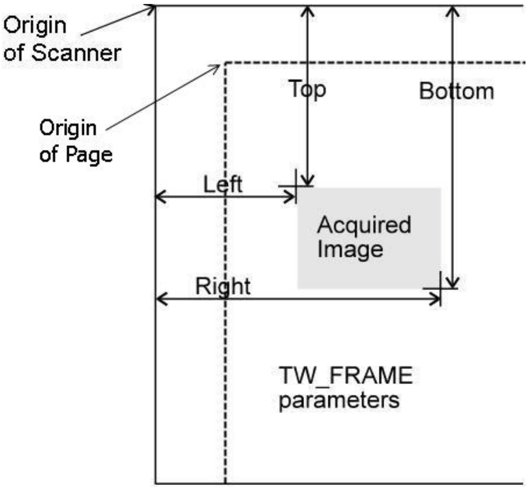
If the application has set Frame.Top and Frame.Left to a non-zero value , set the origin for the image to be acquired accordingly. If possible, the Source should consider reflecting these settings in the user interface when it is raised. For instance, if your Source presents a pre-scan image, consider 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 | ~~TW_UINT32;~~TW_HANDLE<br>On Windows - ~~low word=~~DIB handle<br>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 | ~~0x0101~~0x0100 |
| | ICAP_PIXELTYPE | ~~0x0102~~0x0101 |
| | ICAP_UNITS | ~~0x0103~~0x0102 |
| | ICAP_XFERMECH | ~~0x0104~~0x0103 |
| | ... | ... |

### Pages 8-76 (PDF page 374)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | CAP_PRINTER | ~~0x1024~~0x1026 |
| | CAP_PRINTERENABLED | ~~0x1026~~0x1027 |
| | CAP_PRINTERINDEX | ~~0x1027~~0x1028 |
| | CAP_PRINTERMODE | ~~0x1028~~0x1029 |
| | CAP_PRINTERSTRING | ~~0x1029~~0x102A |
| | CAP_PRINTERSUFFIX | ~~0x102A~~0x102B |
| | CAP_LANGUAGE | ~~0x102B~~0x102C |
| | CAP_FEEDERALIGNMENT | ~~0x102C~~0x102D |
| | CAP_FEEDERORDER | ~~0x102D~~0x102E |
| | ... | ... |

### Pages 8-78 (PDF page 376)

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

| | |
|---|---|
| ~~ICAP_BARCODESEARCHMODE~~ | ~~0x113A~~ |
| ~~ICAP_BARCODEMAXRETRIES~~ | ~~0x113B~~ |
| ~~ICAP_BARCODETIMEOUT~~ | ~~0x113C~~ |
| ~~ICAP_ZOOMFACTOR~~ | ~~0x113D~~ |
| ~~ICAP_BITDEPTHREDUCTION~~ | ~~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 |
|---|---|---|
| | ... | ... |
| | ~~CAP_UICONTROLABLE~~ CAP_UICONTROLLABLE | 0x100E |
| | ... | ... |

Pages 8-75 (PDF page 373)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | ~~CAP_DUPLEXENALBED~~ CAP_DUPLEXENABLED | 0x1013 |
| | ... | ... |

Pages 8-75 (PDF page 373)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | ~~CAP_CUSTOMEDSDATAD~~ CAP_CUSTOMDSDATA | 0x1015 |
| | ... | ... |

Pages 8-76 (PDF page 374)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | ~~CAP_DEVICEDATETIME~~ CAP_DEVICETIMEDATE | 0x101F |
| | ... | ... |

Pages 8-78 (PDF page 376)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | ~~ICAP_PATCHCODEMAXSEARCHPRIORITIE~~ICAP_PATCHCODEMAXSEARCHPRIORITIES | 0x1141 |
| | ... | ... |

Pages 8-90 (PDF page 388)

| Version | Constant | Numeric ID |
|---|---|---|
| | ... | ... |
| | TWSS_C10 | 51 |
| | ~~TWSS_USEXECUTIVE~~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 | ~~0x0701~~0x0702 |
| | ... | ... |