TWAIN Specification

Version 2.4

This document was ratified
by the TWAIN Working Group
on December 11th, 2015

TWAIN
Linking Images With Applications

Acknowledgments

The TWAIN Working Group acknowledges the following individuals and their respective companies for their contributions to this document. Their hard work in defining, designing, editing, proofreading, and discussing the evolution of the document have been invaluable.

Adobe Systems Incorporated
Scott Foshee

Atalasoft
Glenn Chittenden Jr.
Adam Scarborough
Spike McLarty

Avision inc.
Leo Liao
Jun Huang

Dynamsoft Corporation
Catherine Sea

Kodak Alaris, Inc.
Mark McLaughlin

Epson
Tak Shiozaki

Fujitsu Computer Products of America
Pamela Doyle

Hewlett-Packard
Sarah Schwartz

JFL Peripheral Solutions Inc.
Jon Harju
Rebecca Holz
Mihail Mikov

PFU Limited, a Fujitsu Company
Daisuke Kutsuwada

The TWAIN Working Group
Hilary Minugh

We would also like to thank the TWAIN Working Group Technical Committee for their opinions and contributions..

The TWAIN Working Group would like to dedicate TWAIN 2.4 to the memory of Sarah Swartz who worked tirelessly to ensure the professional quality of this specification.”

Table of Contents

1 Introduction

Chapter Contents

Need for Consistency

With the introduction of scanners, digital cameras, and other image acquisition devices, users eagerly discovered the value of incorporating images into their documents and other work. However, supporting the display and manipulation of this raster data placed a high cost on application developers. They needed to create user interfaces and build in device control for the wide assortment of available image devices. Once their application was prepared to support a given device, they faced the discouraging reality that devices continue to be upgraded with new capabilities and features. Application developers found themselves continually revising their product to stay current.

Developers of both the image acquisition devices and the software applications recognized the need for a standard communication between the image devices and the applications. A standard would benefit both groups as well as the users of their products. It would allow the device vendors' products to be accessed by more applications and application vendors could access data from those devices without concern for which type of device, or particular device, provided it. TWAIN was developed because of this need for consistency and simplification.

Elements of TWAIN

TWAIN defines a standard software protocol and API (application programming interface) for communication between software applications and image acquisition devices (the source of the data).

The three key elements in TWAIN are:

- **Application software**

  An application must be modified to use TWAIN.
- **Source Manager software**

  This software manages the interactions between the application and the Source. This code is provided in the TWAIN Developer's Toolkit and should be shipped for free with each TWAIN application and Source.
- **Source software**

  This software controls the image acquisition device and is written by the device developer to comply with TWAIN specifications. Traditional device drivers are now included with the Source software and do not need to be shipped by applications.

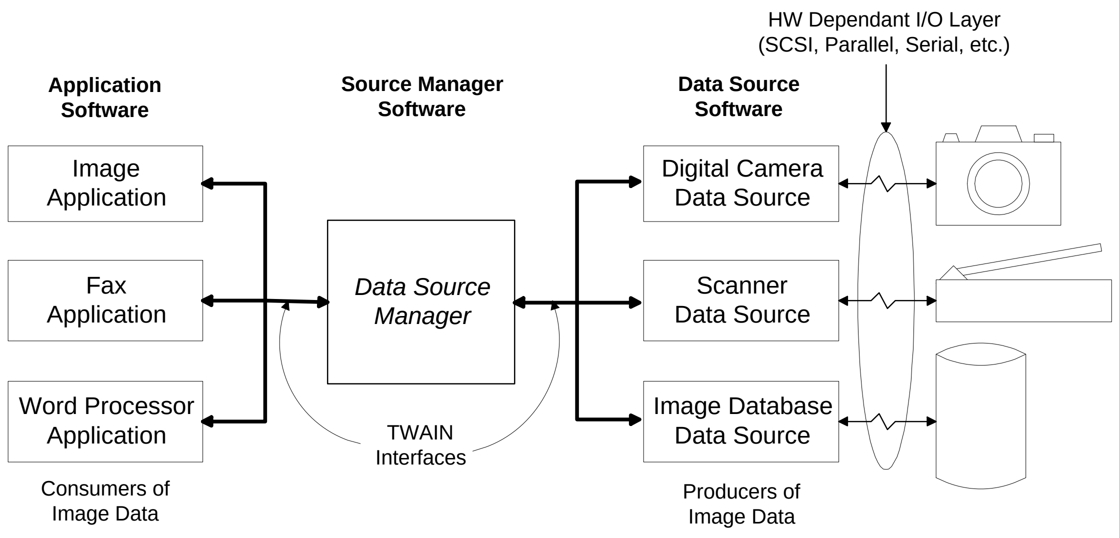


Figure 1-1 TWAIN Elements

Benefits of Using TWAIN

For the Application Developer

- Allows you to offer users of your application a simple way to incorporate images from any compatible raster device without leaving your application.
- Saves time and dollars. If you currently provide low-level device drivers for scanners, etc., you no longer need to write, support, or ship these drivers. The TWAIN-compliant image acquisition devices will provide Source software modules that eliminate the need for you to create and ship device drivers.
- Permits your application to access data from any TWAIN-compliant image peripheral simply by modifying your application code once using the high-level TWAIN application programming interface. No customization by product is necessary. TWAIN image peripherals

can include desktop scanners, hand scanners, digital cameras, frame grabbers, image databases, or any other raster image source that complies to the TWAIN protocol and API.

- Allows you to determine the features and capabilities that an image acquisition device can provide. Your application can then restrict the Source to offer only those capabilities that are compatible with your application's needs and abilities.
- Eliminates the need for your application to provide a user interface to control the image acquisition process. There is a software user interface module shipped with every TWAIN-compliant Source device to handle that process. Of course, you may provide your own user interface for acquisition, if desired.

For the Source Developer

- Increases the use and support of your product. More applications will become image consumers as a result of the ease of implementation and breadth of device integration that TWAIN provides.
- Allows you to provide a proprietary user interface for your device. This lets you present the newest features to the user without waiting for the applications to incorporate them into their interfaces.
- Saves money by reducing your implementation costs. Rather than create and support various versions of your device control software to integrate with various applications, you create just a single TWAIN-compliant Source.

For the End User

- Gives users a simple way to incorporate images into their documents. They can access the image in fewer steps because they never need to leave your application.

Note: TWAIN is supported on all versions of Microsoft Windows and Apple Mac OS X. TWAIN 2.x and higher includes support for Linux and 64-bit operating systems. Information about supporting TWAIN on 16-bit operating systems and older versions of the Apple Macintosh OS are no longer described in the current TWAIN specification. Please refer to version 1.9 of the Specification for support of older operating systems.

Creation of TWAIN

TWAIN was created by a small group of software and hardware companies in response to the need for a proposed specification for the imaging industry. The Working Group's goal was to provide an open, multi-platform solution to interconnect the needs of raster input devices with application software. The original Working Group was comprised of representatives from five companies: Aldus, Caere, Kodak Alaris, Hewlett-Packard, and Logitech. Three other companies, Adobe, Howtek, and Software Architects also contributed significantly.

The design of TWAIN began in January, 1991. Review of the original TWAIN Developer's Toolkit occurred from April, 1991 through January, 1992. The original Toolkit was reviewed by the TWAIN Coalition. The Coalition includes approximately 300 individuals representing 200 companies who continue to influence and guide the future direction of TWAIN.

The current version of TWAIN was written by members of the TWAIN Working Group including Adobe, Kodak Alaris, Inc., Fujitsu Computer Products of America, Hewlett-Packard Company, JFL Peripheral Solutions Inc., Ricoh Corporation, Xerox Corporation, and Lizardtech Corporation.

In May, 1998, an agreement was announced between Microsoft and the TWAIN Working Group which provided for the inclusion of the TWAIN Data Source Manager in Microsoft Windows 98 and Microsoft Windows NT 5.0.

During the creation of TWAIN, the following architecture objectives were adhered to:

- **Ease of Adoption** — Allow an application vendor to make their application TWAIN-compliant with a reasonable amount of development and testing effort. The basic features of TWAIN should be implemented just by making modest changes to the application. To take advantage of a more complete set of functionality and control capabilities, more development effort should be anticipated.
- **Extensibility** — The architecture must include the flexibility to embrace multiple windowing environments spanning various host platforms (Mac OS X, Microsoft Windows, Linux with KDE or Gnome, etc.) and facilitate the exchange of various data types between Source devices and destination applications. Currently, only the raster image data type is supported but suggestions for future extensions include text, facsimile, vector graphics, and others.
- **Integration** — Key elements of the TWAIN implementation "belong" in the operating system. The agreement between Microsoft and the TWAIN Working Group indicates that this integration into the operating system is beginning. TWAIN must be implemented to encourage backward compatibility (extensibility) and smooth migration into the operating system. An implementation that minimizes the use of platform-specific mechanisms will have enhanced longevity and adoptability.
- **Easy Application <-> Source Interconnect** — A straight-forward Source identification and selection mechanism will be supplied. The application will drive this mechanism through a simple API. This mechanism will also establish the data and control links between the application and Source. It will support capability and configuration communication and negotiation between the application and Source.
- **Encapsulated Human Interface** — A device-native user interface will be required in each Source. The application can optionally override this native user interface while still using the Source to control the physical device.

2

Technical Overview

Chapter Contents

The TWAIN protocol and API are easiest to understand when you see the overall picture. This chapter provides a technical overview of TWAIN.

TWAIN Architecture

The transfer of data is made possible by three software elements that work together in TWAIN: the application, the Source Manager, and the Source.

These elements use the architecture of TWAIN to communicate. The TWAIN architecture consists of four layers:

- Application
- Protocol
- Acquisition
- Device

The TWAIN software elements occupy the layers as illustrated below. Each layer is described in the sections that follow.

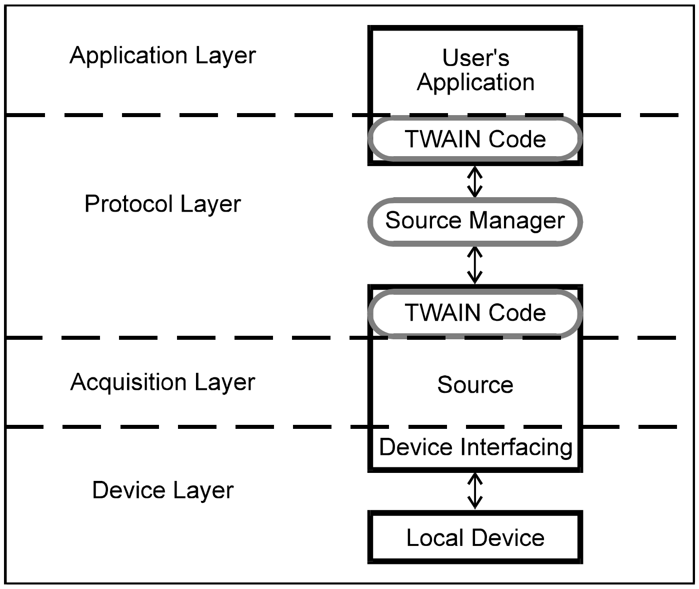


Figure 2-1 TWAIN Software Elements

Application

The user's software application executes in this layer.

TWAIN describes user interface guidelines for the application developer regarding how users access TWAIN functionality and how a particular Source is selected.

TWAIN is not concerned with how the application is implemented. TWAIN has no effect on any inter-application communication scheme that the application may use.

Protocol

The protocol is the "language" spoken and syntax used by TWAIN. It implements precise instructions and communications required for the transfer of data.

The protocol layer includes:

- The portion of application software that provides the interface between the application and TWAIN
- The TWAIN Source Manager provided by TWAIN
- The software included with the Source device to receive instructions from the Source Manager and transfer back data and Return Codes

The contents of the protocol layer are discussed in more detail in "Communication Between the Elements of TWAIN" on page 2-5.

Acquisition

Acquisition devices may be physical (like a scanner or digital camera) or logical (like an image database). The software elements written to control acquisitions are called Sources and reside primarily in this layer.

The Source transfers data for the application. It uses the format and transfer mechanism agreed upon by the Source and application.

The Source always provides a built-in user interface that controls the device(s) the Source was written to drive. An application can override this and present its own user interface for acquisition, if desired.

Device

This is the location of traditional low-level device drivers. They convert device-specific commands into hardware commands and actions specific to the particular device the driver was written to accompany. Applications that use TWAIN no longer need to ship device drivers because they are part of the Source.

TWAIN is not concerned with the device layer at all. The Source hides the device layer from the application. The Source provides the translation from TWAIN operations and interactions with the Source's user interface into the equivalent commands for the device driver that cause the device to behave as desired.

Note: The Protocol layer is the most thoroughly and rigidly defined to allow precise communications between applications and Sources. The information in this document concentrates on the Protocol and Acquisition layers.

TWAIN User Interface

When an application uses TWAIN to acquire data, the acquisition process may be visible to the application's users in the following three areas:

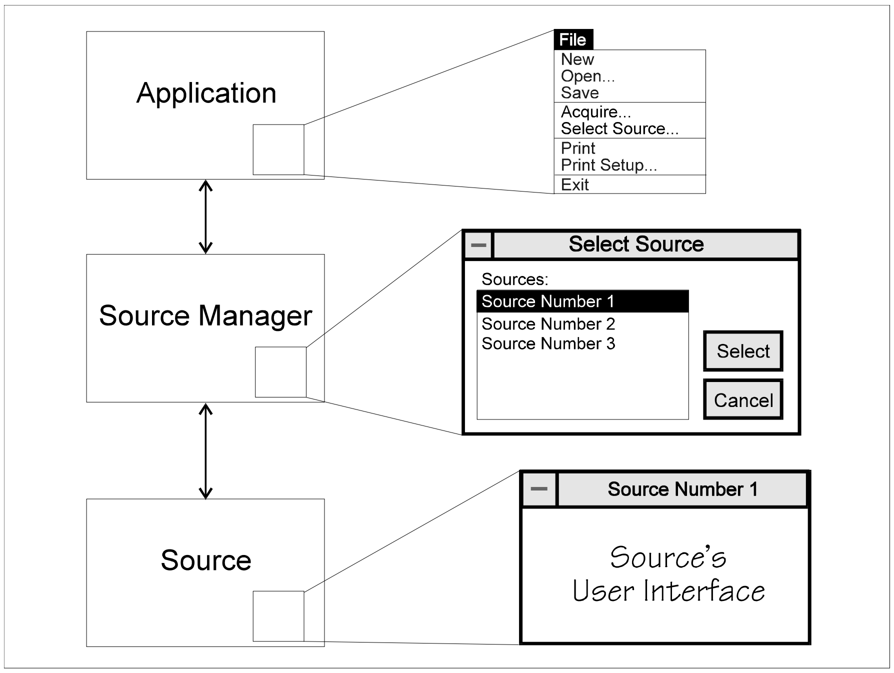


Figure 2-2 Data Acquisition Process

The Application

The user needs to select the device from which they intend to acquire the data. They also need to signal when they are ready to have the data transferred. To allow this, TWAIN strongly recommends the application developer add two options to their File menu:

- **Select Source** - to select the device
- **Acquire** - to begin the transfer process

The Source Manager

When the user chooses the Select Source option, the application requests that the Source Manager display its Select Source dialog box. This lists all available devices and allows the user to highlight and select one device. If desired, the application can write its own version of this user interface.

The Source

Every TWAIN-compliant Source provides a user interface specific to its particular device. When the application user selects the Acquire option, the **Source's User Interface** may be displayed. If desired, the application can write its own version of this interface, too.

Communication Between the Elements of TWAIN

Communication between elements of TWAIN is possible through two entry points. They are called `DSM_Entry( )` and `DS_Entry( )`. `DSM` means Data Source Manager and `DS` means Data Source.

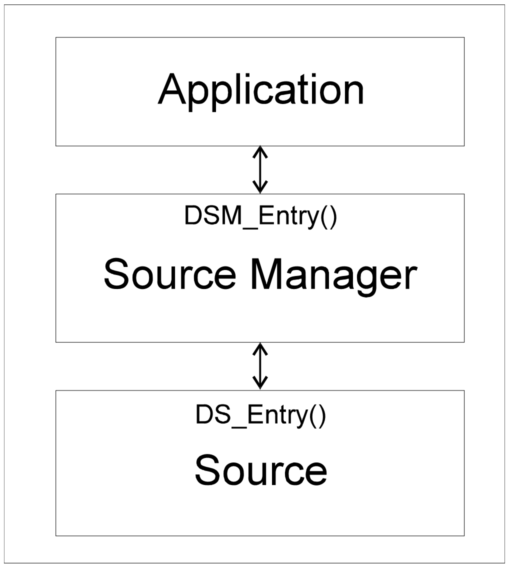


Figure 2-3 Entry Points for Communicating Between Elements

The Application

The goal of the application is to acquire data from a Source. However, applications cannot contact the Source directly. All requests for data, capability information, error information, etc. must be handled through the Source Manager.

Approximately 140 operations are defined by TWAIN. The application sends them to the Source Manager for transmission. The application specifies which element, Source Manager or Source, is the final destination for each requested operation.

The application communicates to the Source Manager through the Source Manager's only entry point, the `DSM_Entry( )` function.

The parameter list of the `DSM_Entry` function contains:

- An identifier structure providing information about the application that originated the function call
- The destination of this request (Source Manager or Source)
- A triplet that describes the requested operation. The triplet specifies:
  - Data Group for the Operation (`DG_`)
  - Data Argument Type for the Operation (`DAT_`)
  - Message for the Operation (`MSG_`)

- (These are described more in the section called Using Operation Triplets located later in this chapter.)
- A pointer field to allow the transfer of data

The function call returns a value (the Return Code) indicating the success or failure of the operation.

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

Note: Data type definitions are covered in Chapter 8, "Data Types and Data Structures", and in the file called `TWAIN.H` which can be downloaded from the TWAIN Working Group web site http://www.twain.org.)

The Source Manager

The Source Manager provides the communication path between the application and the Source, supports the user's selection of a Source, and loads the Source for access by the application. Communications from application to Source Manager arrive in the `DSM_Entry( )` entry point.

- **If the destination in the DSM_Entry call is the Source Manager**

  The Source Manager processes the operation itself.

- **If the destination in the DSM_Entry call is the Source**

  The Source Manager translates the parameter list of information, removes the destination parameter and calls the appropriate Source. To reach the Source, the Source Manager calls the Source's `DS_Entry( )` function. TWAIN requires each Source to have this entry point.

Written in C code form, the `DS_Entry` function call looks like this:

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

In addition, the Source Manager can initiate three operations that were not originated by the application. These operation triplets exist just for Source Manager to Source communications and are executed by the Source Manager while it is displaying its Select Source dialog box. The operations are used to identify the available Sources and to open or close Sources.

The implementation of the Source Manager differs between the supported systems:

On Windows

- The Source Manager is a Dynamic Link Library (TWAINDSM.DLL).
- The Source Manager can manage simultaneous sessions between an application and many Sources.

On Macintosh

- The Source Manager is a Mach-O framework (TWAIN.framework, TWAINDSM.framework).

On Linux

- The Source Manager is a shared library (/usr/local/lib/libtwaindsm.so).
- The Source Manager can manage simultaneous sessions between an application and many Sources.

The Source

The Source receives operations either from the application, via the Source Manager, or directly from the Source Manager. It processes the request and returns the appropriate Return Code (the codes are prefixed with `TWRC_`) indicating the results of the operation to the Source Manager. If the originator of the operation was the application, then the Return Code is passed back to the application as the return value of its `DSM_Entry( )` function call. If the operation was unsuccessful, a Condition Code (the codes are prefixed with `TWCC_`) containing more specific information is set by the Source. Although the Condition Code is set, it is not automatically passed back. The application must invoke an operation to inquire about the contents of the Condition Code.

The implementation of the Source is the same as the implementation of the Source Manager:

On Windows

- The Source is a Dynamic Link Library (DLL) with a .ds extension.

On Macintosh

- The Source is implemented as a bundle (preferably Mach-O) with a .ds extension.

On Linux

- The Source is a shared library (.so) with a .ds extension.

Communication Flowing from Source to Application

The majority of operation requests are initiated by the application and flow to the Source Manager and Source. The Source, via the Source Manager, is able to pass back data and Return Codes.

However, there are four times when the Source needs to interrupt the application and request that an action occur. These situations are:

- **Notify the application that a data transfer is ready to occur.** The time required for a Source to prepare data for a transfer will vary. Rather than have the application wait for the preparation to be complete, the Source just notifies it when everything is ready. The `MSG_XFERREADY` notice is used for this purpose.

- **Request that the Source's user interface be disabled.** This notification should be sent by the Source to the application when the user clicks on the "Close" button of the Source's user interface. The `MSG_CLOSEDSREQ` notice is used for this purpose.
- **Notify the application that the OK button has been pressed, accepting the changes the user has made.** This is only used if the Source is opened with `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY`. The `MSG_CLOSEDSOK` notice is used for this purpose.
- **A Device Event has occurred.** This notification is sent by the Source to the Application when a specific event has occurred, but only if the Application gave the Source prior instructions to pass along such events. The `MSG_DEVICEEVENT` notice is used for this purpose.

These notices are presented to the application in its event (or message) loop. The process used for these notifications is covered more fully in Chapter 12, "Operating System Dependencies", in the discussion of the application's event loop.

Identifying TWAIN 2.0 Elements

It is not sufficient to test the `TW_IDENTITY`.ProtocolMajor field to determine if an Application, a Data Source Manager or a Source is TWAIN 2.0 compliant. Check the `TW_IDENTITY.SupportedGroups` field for the Application or the Source, and look for the following:

- `DF_APP2`, indicating that the Application is 2.0 compliant
- `DF_DSM2`, indicating that the Data Source Manager is 2.0 compliant
- `DF_DS2`, indicating that the Data Source is 2.0 compliant

Applications

All TWAIN 2.0 compliant Applications must report `DF_APP2` in their `TW_IDENTITY.SupportedGroups` field.

All TWAIN 2.0 compliant Applications must test for the DF_DSM2 flag in the `TW_IDENTITY.SupportedGroups` field, after a call to `DG_CONTROL / DAT_PARENT / MSG_OPENDSM`. If this flag is not found, then follow the legacy behavior for 1.x Applications, using the memory management functions detailed in the TWAIN Specification.

If the flag is found, then the Application must call `DG_CONTROL / DAT_ENTRYPOINT / MSG_GET` in State 3, before performing any other operation, to obtain pointers to the memory management functions.

Sources

All TWAIN 2.0 compliant Sources must report DF_DS2 in their `TW_IDENTITY.SupportedGroups` field.

All TWAIN 2.0 compliant Sources must be prepared to receive the `DG_CONTROL / DAT_ENTRYPOINT / MSG_SET` call in State 3, before `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS` is called. If this operation is not called, then follow the legacy behavior for 1.x Sources, using the memory management functions detailed in the TWAIN Specification, and locating the Data Source Manager as indicated.

If the operation is called then the Source must use the pointers to the memory management functions, and must use the supplied entry point to access DSM_Entry.

Using DAT_CALLBACK for Messages from the Source to the Application

Applications

TWAIN Applications running on Linux, Apple Macintosh OS X or Windows must use `DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK` to register to receive asynchronous notifications for events like `MSG_XFERREADY`.

TWAIN Applications using older versions of the Data Source Manager (no `DF_DSM2` flag detected) must use legacy behavior. Please refer to Chapter 12, "Operating System Dependencies" for more information.

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

Sources

TWAIN Sources that detect the presence of the `DF_DSM2` flag inside of `TW_IDENTITY.SupportedGroups` must use `DG_CONTROL / DAT_NULL` with the appropriate message to return events like `MSG_XFERREADY`.

TWAIN Sources using older versions of the Data Source Manager (no `DF_DSM2` flag detected) must use legacy behavior. Please refer to Chapter 12, "Operating System Dependencies" for more information.

Installation of the Data Source Manager

TWAIN Applications and Sources should install the latest version of the Data Source Manager. Please check the TWAIN website http://www.twain.org to see if your Operating System or distro is represented, and if not, please consider making a submission to the TWAIN Working Group.

Refer to Chapter 12, "Operating System Dependencies".

Memory Management in TWAIN 2.0 and Higher

TWAIN requires Applications and Sources to manage each other's memory. The chief problem is guaranteeing agreement on the API's to use.

TWAIN 2.0 introduces four new functions that are obtained from the Source Manager through `DAT_ENTRYPOINT`.

```
TW_HANDLE TW_CALLINGSTYLE DSM_MemAllocate (TW_UINT32)
void TW_CALLINGSTYLE DSM_MemFree (TW_HANDLE)
TW_MEMREF TW_CALLINGSTYLE DSM_MemLock (TW_HANDLE)
void TW_CALLINGSTYLE DSM_MemUnlock (TW_HANDLE)
```

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

If `DAT_ENTRYPOINT` is not obtained from the Source Manager then Applications and Sources must use the legacy calls. Refer to Chapter 12, "Operating System Dependencies".

Also see DSMInterface.cpp sample source here: http://twain-samples.svn.sourceforge.net/

Using Operation Triplets

The `DSM_Entry( )` and `DS_Entry( )` functions are used to communicate operations. An operation is an action that the application or Source Manager invokes. Typically, but not always, it involves using data or modifying data that is indicated by the last parameter (`pData`) in the function call.

Requests for actions occur in one of these ways:

From	To	Using this function
The application	The Source Manager	`DSM_Entry` with the `pDest` parameter set to `NULL`
The application	The Source (via the Source Manager)	`DSM_Entry` with the `pDest` parameter set to point to a valid structure that identifies the Source
The Source Manager	The Source	`DS_Entry`

The desired action is defined by an operation triplet passed as three parameters in the function call. Each triplet uniquely, and without ambiguity, specifies a particular action. No operation is specified by more than a single triplet. The three parameters that make up the triplet are Data Group, Data Argument Type, and Message ID. Each parameter conveys specific information.

Data Group (DG_xxxx)

Operations are divided into large categories by the Data Group identifier. The following are the currently defined Data Groups in TWAIN:

- **CONTROL** (The identifier is `DG_CONTROL`.): These operations involve control of the TWAIN session. An example where `DG_CONTROL` is used as the Data Group identifier is the operation to open the Source Manager.
- **IMAGE** (The identifier is `DG_IMAGE`.): These operations work with image data. An example where `DG_IMAGE` is used as a Data Group is an operation that requests the transfer of image data.
- **AUDIO** (The identifier is `DG_AUDIO`): These operations work with audio data (supported by some digital cameras). An example where `DG_AUDIO` is used as a Data Group is an operation that requests the transfer of audio data.

Data Argument Type (DAT_xxxx)

This parameter of the triplet identifies the type of data that is being passed or operated upon. The argument type may reference a data structure or a variable. There are many data argument types. One example is `DAT_IDENTITY`.

The `DAT_IDENTITY` type is used to identify a TWAIN element such as a Source. Data is typically passed or modified through the `pData` parameter of the `DSM_Entry` and `DSM_Entry`. In this case, the `pData` parameter would point to a data structure of type `TW_IDENTITY`. The data argument type begins with `DAT_xxxx` and the associated data structure begins with `TW_xxxx` and duplicates the second part of the name. This pattern is followed consistently for most data argument types and their data structures. Any exceptions are noted on the reference pages in Chapter 7, "Operation Triplets" and Chapter 8, "Data Types and Data Structures".

Message ID (MSG_xxxx)

This parameter identifies the action that the application or Source Manager wishes to have taken. There are many different messages such as `MSG_GET` or `MSG_SET`. They all begin with the prefix of `MSG_`.

Examples of Operation Triplets

- The triplet the application sends to the Source Manager to open the Source Manager module is:

  `DG_CONTROL / DAT_PARENT / MSG_OPENDSM`

- The triplet that the application sends to instruct the Source Manager to display its Select Source dialog box and thus allow the user to select which Source they plan to obtain data from is:

  `DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT`

- The triplet the application sends to transfer data from the Source into a file is:

  `DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET`

The State-Based Protocol

The application, Source Manager, and Source must communicate to manage the acquisition of data. It is logical that this process must occur in a particular sequence. For example, the application cannot successfully request the transfer of data from a Source before the Source Manager is loaded and prepared to communicate the request.

To ensure the sequence is executed correctly, the TWAIN protocol defines seven states that exist in TWAIN sessions. A session is the period while an application is connected to a particular Source via the Source Manager. The period while the application is connected to the Source Manager is another unique session. At a given point in a session, the TWAIN elements of Source Manager and Source each occupy a particular state. Transitions to a new state are caused by operations requested by the application or Source. Transitions can be in the forward or backward direction. Most transitions are single-state transitions. For example, an operation moves the Source Manager from State 1 to State 2 not from State 1 to State 3. (There are situations where a two-state transition may occur. They are discussed in Chapter 3, "Application Implementation".)

When viewing the state-based protocol, it is helpful to remember:

States 1, 2, and 3

- Are occupied only by the Source Manager.

- The Source Manager never occupies a state greater than State 3.

States 4, 5, 6, and 7

- Are occupied exclusively by Sources.
- A Source never has a state less than 4 if it is open. If it is closed, it has no state.
- If an application uses multiple Sources, each connection is a separate session and each open Source "resides" in its own state without regard for what state the other Sources are in.

The State Transition Diagram looks like this:

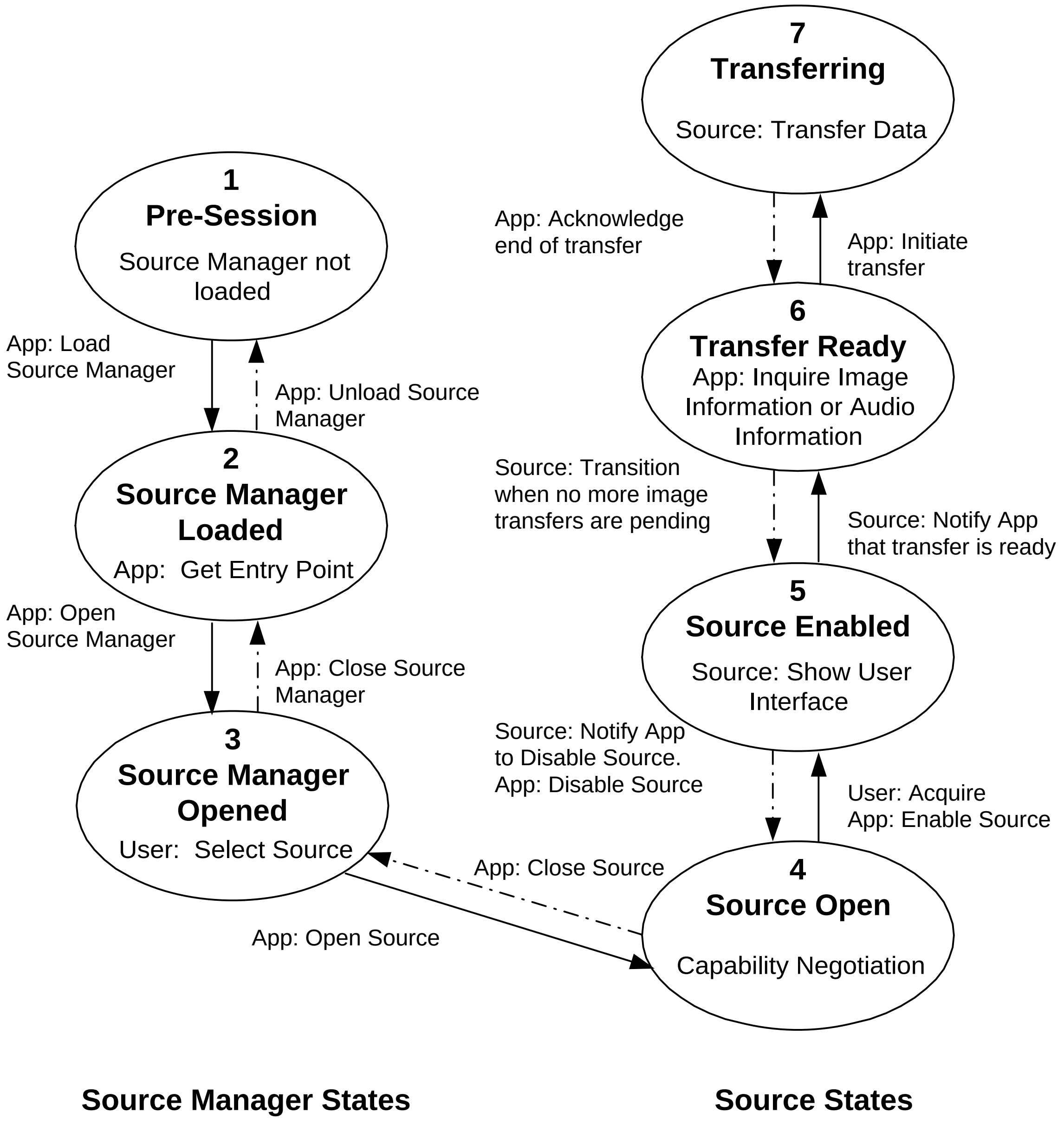

7
Transferring
Source: Transfer Data
1
Pre-Session
Source Manager not loaded
App: Acknowledge end of transfer
App: Initiate transfer
App: Load Source Manager
App: Unload Source Manager
6
Transfer Ready
App: Inquire Image Information or Audio Information
2
Source Manager Loaded
App: Get Entry Point
Source: Transition when no more image transfers are pending
Source: Notify App that transfer is ready
App: Open Source Manager
App: Close Source Manager
5
Source Enabled
Source: Show User Interface
3
Source Manager Opened
User: Select Source
Source: Notify App to Disable Source. App: Disable Source
User: Acquire App: Enable Source
App: Close Source
4
Source Open
Capability Negotiation
App: Open Source
Source Manager States
Source States

Figure 2-4 State Transition Diagram

The Description of the States

The following sections describe the states.

State 1 — Pre-Session

The Source Manager *resides* in State 1 before the application establishes a session with it.

At this point, the Source Manager code has been installed on the disk but typically is not loaded into memory yet.

The only case where the Source Manager could already be loaded and running is under Windows because the implementation is a DLL (hence, the same instance of the Source Manager can be shared by multiple applications). If that situation exists, the Source Manager will be in State 2 or 3 with the application that loaded it.

State 2 — Source Manager Loaded

The Source Manager now is loaded into memory. It is not open yet.

At this time, the Source Manager is prepared to accept other operation triplets from the application.

State 3 — Source Manager Open

The Source Manager is open and ready to manage Sources.

The Source Manager is now prepared to provide lists of Sources, to open Sources, and to close Sources.

The Source Manager will remain in State 3 for the remainder of the session until it is closed. The Source Manager refuses to be closed while the application has any Sources open.

State 4 — Source Open

The Source has been loaded and opened by the Source Manager in response to an operation from the application. It is ready to receive operations.

The Source should have verified that sufficient resources (i.e. memory, device is available, etc.) exist for it to run.

The application can inquire about the Source's capabilities (i.e. levels of resolution, support of color or black and white images, automatic document feeder available, etc.). The application can also set those capabilities to its desired settings. For example, it may restrict a Source capable of providing color images to transferring black and white only.

Note: Inquiry about a capability can occur while the Source is in States 4, 5, 6, or 7. But, an application can set a capability only in State 4 unless special permission is negotiated between the application and Source.

State 5 — Source Enabled

The Source has been enabled by an operation from the application via the Source Manager and is ready for user-enabled transfers.

If the application has allowed the Source to display its user interface, the Source will do that when it enters State 5.

State 6 — Transfer is Ready

The Source is ready to transfer one or more data items (images) to the application.

The transition from State 5 to 6 is triggered by the Source notifying the application that the transfer is ready.

Before initiating the transfer, the application must inquire information about the image (resolution, image size, etc.). If the Source supports audio, then before transferring the image, the Application must transfer all the audio snippets that are associated with the image.

It is possible for more than one image to be transferred in succession. This topic is covered thoroughly in Chapter 4, "Advanced Application Implementation".

State 7 — Transferring

The Source is transferring the image to the application.

The transfer mechanism being used was negotiated during State 4.

The transfer will either complete successfully or terminate prematurely. The Source sends the appropriate Return Code indicating the outcome.

Once the Source indicates that the transfer is complete, the application must acknowledge the end of the transfer.

A TWAIN 2.0 compliant Application tests its `TW_IDENTITY` .SupportedGroups for `DF_DSM2` after a call to `DG_CONTROL/DAT_PARENT/MSG_OPENDSM` and if found it issues a call to `DG_CONTROL` / `DAT_ENTRYPOINT` / `MSG_GET`.

A TWAIN 2.0 compliant Source is sent `DG_CONTROL` / `DAT_ENTRYPOINT` / `MSG_SET`; it tests the Application’s `TW_IDENTITY.SupportedGroups` for `DF_DSM2` and `DF_APP2`.

Capabilities

One of TWAIN’s benefits is it allows applications to easily interact with a variety of acquisition devices. Devices can provide image or audio data. For instance,

- Some devices have automatic document feeders.
- Some devices are not limited to one image but can transfer multiple images.
- Some devices support color images.
- Some devices offer a variety of halftone patterns.
- Some devices support a range of resolutions while others may offer different choices.
- Some devices allow the recording of audio data associated with an image.

Developers of applications need to be aware of a Source’s capabilities and may influence the capabilities that the Source offers to the application’s users. To do this, the application can perform **capability negotiation**. The application generally follows this process:

- **Determine** if the selected Source supports a particular capability.
- **Inquire** about the Current Value for this capability. Also, inquire about the capability’s Default Value and the set of Available Values that are supported by the Source for that capability.
- **Request** that the Source set the Current Value to the application’s desired value. The Current Value will be displayed as the current selection in the Source’s user interface.

- **Limit**, if needed, the Source's Available Values to a subset of what would normally be offered. For instance, if the application wants only black and white data, it can restrict the Source to transmit only that. If a limitation effects the Source's user interface, the Source should modify the interface to reflect those changes. For example, it may gray out options that are not available because of the application's restrictions.
- **Verify** that the new values have been accepted by the Source.

TWAIN capabilities are divided into three groups:

- **`CAP_xxxx`:** Capabilities whose names begin with CAP are capabilities that could apply to any general Source. Such capabilities include use of automatic document feeders, identification of the creator of the data, etc.
- **`ICAP_xxxx`:** Capabilities whose names begin with ICAP are capabilities that apply to image devices. The "I" stands for image. (When TWAIN is expanded to support other data transfer such as text or fax data, there will be TCAPs and FCAPs in a similar style.)
- **`ACAP_xxxx`:** Capabilities whose names begin with ACAP are capabilities that apply to devices that support audio. The "A" stands for audio.

Capability Containers

Capabilities exist in many varieties but all have a Default Value, Current Value, and may have other values available that can be supported if selected. To help categorize the supported values into clear structures, TWAIN defines four types of **containers** for capabilities.

Name of the Data Structure for the Container	Type of Contents
`TW_ONEVALUE`	A single value whose current and default values are coincident. The range of available values for this type of capability is simply this single value. For example, a capability that indicates the presence of a document feeder could be of this type.
`TW_ARRAY`	An array of values that describes the current logical item. The available values may be a larger array of values. For example, a list of the names, such as the supported capabilities list returned by the `CAP_SUPPORTEDCAPS` capability, would use this type of container.
`TW_RANGE`	Many capabilities allow users to select their current value from a range of regularly spaced values. The capability can specify the minimum and maximum acceptable values and the incremental step size between values. For example, resolution might be supported from 100 to 600 in steps of 50 (100, 150, 200, ..., 550, 600).

Name of the Data Structure for the Container	Type of Contents
`TW_ENUMERATION`	This is the most general type because it defines a list of values from which the Current Value can be chosen. The values do not progress uniformly through a range and there is not a consistent step size between the values. For example, if a Source's resolution options did not occur in even step sizes then an enumeration would be used (for example, 150, 400, and 600).

In general, most capabilities can have more than one of these containers applied to them depending on how the particular Source implements the capability. The data structure for each of these containers is defined in Chapter 8, "Data Types and Data Structures". A complete table with all defined capabilities is located in Chapter 10, "Capabilities". A few of the capabilities must be supported by the application and Source. The remainder of the capabilities are optional.

Capability Negotiation and Container Types

It is very important for Application and Data Source developers to note that Container types are dictated by the Data Source in all cases where a value is queried. Also the allowable container types of each capability are clearly defined in Chapter 10, "Capabilities", of the TWAIN Specification. The only time it is appropriate for the calling Application to specify a container type is during the `MSG_SET` operation. At that time, the Application must also consider the allowable containers and types for the particular capability.

It is recommended that an Application use the containers for `MSG_SET` that it received in `MSG_GET`.

Capability Containers and String Values

The only containers that can possibly hold a string are the following:

```
TW_ENUMERATION
TW_ARRAY
TW_ONEVALUE
```

It is not possible or useful to use this type in a `TW_RANGE.` In fact, there is no case where a capability has been defined in Chapter 10, "Capabilities", of the TWAIN Specification where a `TW_RANGE` is allowed for a `TW_STRxxxx` type of value.

There are four types of TWAIN strings defined for developer use:

```
TW_STR32
TW_STR64
TW_STR128
TW_STR256
```

As of version 1.7, only the following capabilities accept strings:

```
CAP_AUTHOR, TW_ONEVALUE, TW_STR128
```

```
CAP_CAPTION, TW_ONEVALUE, TW_STR255
CAP_TIMEDATE, TW_ONEVALUE, TW_STR32
ICAP_HALFTONES, TW_ONEVALUE/TW_ENUMERATION/TW_ARRAY, TW_STR32
```

The definition of the various container types could be confusing. For example, the definition of a TW_ONEVALUE is as follows:

```
/* TWON_ONEVALUE. Container for one value. */
typedef struct {
   TW_UINT16  ItemType;
   TW_UINT32  Item;
} TW_ONEVALUE, FAR * pTW_ONEVALUE;
```

At first glance, it is tempting to try placing the string into this container by assigning "Item" to be a pointer. This is not at all consistent with the implementation of other structures in the specification and introduces a host of problems concerning management of the memory occupied by the string. (See TW_IDENTITY for consistent TWAIN string use)

The correct and consistent method of holding a string in a TWAIN container is to ensure the string is embedded in the container itself. Either a new structure is defined within the developers code, or the added size is considered when allocating the container.

The following examples are designed to demonstrate possible methods of using TWAIN Strings in Containers. These examples are suitable for demonstration only, and require refinement to be put to real use.

Example 1:

TW_ONEVALUE structure defined for holding a TW_STR32 value

```
/* TWON_ONEVALUESTR32. Container for one value holding TW_STR32. */
typedef struct {
   TW_UINT16  ItemType;
   TW_STR32  Item;
} TW_ONEVALUESTR32, FAR * pTW_ONEVALUESTR32;
```

Note: Pay attention to two-byte structure packing when defining custom container structures.

This clearly demonstrates where the memory is allocated and where the string resides. The data source does not have to be concerned with how the string is managed locally, and the application does not have to be concerned with managing the string memory or contents.

Example 2:

TW_ONEVALUE structure allocated and filled with consideration of holding a TW_STR32 value (Windows Example)

```
HGLOBAL AllocateAndFillOneValueStr32( const pTW_STR32 pInString )
{
   DWORD dwContainerSize = 0l;
   HGLOBAL hContainer = NULL;
   pTW_ONEVALUE pOneValue = NULL;
   pTW_STR32 pString = NULL;
   assert(pInString);

   // Note: This calculation will yield a size approximately one
   // pointer larger than that required for this container
   // (sizeof(TW_UINT32)).  For simplicity the size difference
   // is negligible.  The first TW_STR32 item shall be located
   // immediately after the pEnum->DefaultIndex member.

   dwContainerSize = sizeof(TW_ONEVALUE) + sizeof(TW_STR32);
   hContainer = GlobalAlloc( GPTR, dwContainerSize );

   if(hContainer)
   {
      pOneValue = (pTW_ONEVALUE)GlobalLock(hContainer);
      if(pOneValue)
      {
          pOneValue->ItemType = TWTY_STR32;
          pString = (pTW_STR32)&pOneValue->Item;

          memcpy(pString, pInString, sizeof(TW_STR32));
          GlobalUnlock(hContainer);
          pOneValue = NULL;
          pString = NULL;
      }
   }
   return hContainer;
}
```

Example 3:

TW_ENUMERATION structure allocated with consideration of holding TW_STR32 values (Windows Example)

```
HGLOBAL AllocateEnumerationStr32( TW_UINT32 unNumItems )
{
   DWORD dwContainerSize = 0l;
   HGLOBAL hContainer = NULL;
   pTW_ENUMERATION pEnum = NULL;

   // Note: This calculation will yield a size approximately
   // one pointer larger than that required for this container
   // (sizeof(pTW_UINT8)).  For simplicity the size difference is
   // negligible.  The first TW_STR32 item shall be located
   // immediately after the pEnum->DefaultIndex member.
   dwContainerSize = sizeof(TW_ENUMERATION) + ( sizeof(TW_STR32) *
   unNumItems);
   hContainer = GlobalAlloc( GPTR, dwContainerSize );
   if(hContainer)
   {
        pEnum = (pTW_ENUMERATION) GlobalLock(hContainer);
        if(pEnum)
        {
           pEnum->ItemType = TWTY_STR32;
           pEnum->NumItems = unNumItems;
             GlobalUnlock(hContainer);
           pEnum = NULL;
        }
   }
   return hContainer;
}
```

Example 4

Indexing a string from an Enumeration Container

```
pTW_STR128 IndexStr128FromEnumeration( pTW_ENUMERATION pEnum, TW_UINT32
unIndex)
{
      BYTE *pBegin = (BYTE *)&pEnum->ItemList[0];
      assert(pEnum->NumItems > unIndex);
      assert(pEnum->ItemType == TWTY_STR128);
      pBegin += (unIndex * sizeof(TW_STR128));
      return (pTW_STR128)pBegin;
}
```

Modes Available for Data Transfer

There are three different modes that can be used to transfer data from the Source to the application: native, disk file, and buffered memory.

Note: At this time, TWAIN support for audio only allows native and disk file transfers.)

Native

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

The format of the data is platform-specific:

- Windows: DIB (Device-Independent Bitmap)
- Macintosh: A TIFF image file in memory if both application and data source are version 2.4 or later. A PICT image in memory if either the application or the data source is TWAIN 2.3 and earlier.
- Linux: A TIFF image file in memory

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

Disk File

A Source is not required to support this transfer mode but it is recommended.

The application creates the file to be used in the transfer and ensures that it is accessible by the Source for reading and writing.

A capability exists that allows the application to determine which file formats the Source supports. The application can then specify the file format and file name to be used in the transfer.

The disk file mode is ideal when transferring large images that might encounter memory limitations with Native mode. Disk File mode is simpler to implement than the buffered mode discussed next. However, Disk File mode is a bit slower than Buffered Memory mode and the application must be able to manage the file after creation.

Buffered Memory

Every Source must support this transfer mode.

The transfer occurs through memory using one or more buffers. Memory for the buffers are allocated and deallocated by the application.

The data is transferred as an unformatted bitmap. The application must use information available during the transfer (`TW_IMAGEINFO` and `TW_IMAGEMEMXFER`) to learn about each individual buffer and be able to correctly interpret the bitmap.

If using the Native or Disk File transfer modes, the transfer is completed in one action. With the Buffered Memory mode, the application may need to loop repeatedly to obtain more than one buffer of data.

Buffered Memory transfer offers the greatest flexibility, both in data capture and control. However, it is the least simple to implement.

3

Application Implementation

Chapter Contents

This chapter provides the basic information needed to implement TWAIN at a minimum level.

Advanced topics are discussed in Chapter 4, "Advanced Application Implementation". They include how to take advantage of Sources that offer automatic feeding of multiple images.

For OS specific requirements refer to Chapter 12, "Operating System Dependencies".

Levels of TWAIN Implementation

Application developers can choose to implement TWAIN features in their application along a range of levels.

- **At the minimum level:** The application does not have to take advantage of capability negotiation or transfer mode selection. Using TWAIN defaults, it can just acquire a single image in the Native mode.
- **At a greater level:** The application can negotiate with the Source for desired capabilities or image characteristics and specify the transfer arrangement. This gives the application more control over the type of image it receives. To do this, developers should follow the instructions provided in this chapter and use information from Chapter 4, "Advanced Application Implementation", as well.
- **At the highest level:** An application may choose to negotiate capabilities, select transfer mode, and create/present its own user interfaces instead of using the built-in ones provided

with the Source Manager and Source. Again, refer to this chapter and Chapter 4, "Advanced Application Implementation".

Installation of the Source Manager Software

The TWAIN Source Manager is an Open Source project maintained and owned by the TWAIN Working Group (TWG). Binaries are built and distributed by the TWG for a few Operating Systems. Applications are responsible for distributing and installing the most recent release of the Source manager software available from twain.org.

For OS specific requirements refer to Chapter 12, "Operating System Dependencies".

Changes Needed to Prepare for a TWAIN Session

The following areas of the application must be changed before a TWAIN session can begin. The application developer must:

- Alter the application's user interface to add **Select Source** and **Acquire** menu choices.
- Include the file called `TWAIN.H` in your application.
- Alter the application's event loop.

Alter the Application's User Interface to Add Select Source and Acquire Options

As mentioned in the Chapter 2, "Technical Overview", the application should include two menu items in its File menu: **Select Source...** and **Acquire...**. It is strongly recommended that you use these phrases since this consistency will benefit all users.

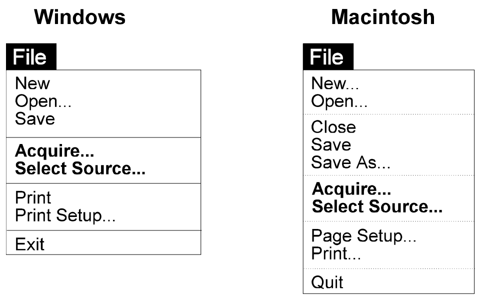


Figure 3-1 User Interface for Selecting a Source and Acquiring Options

Note the following:

When this is selected:	The application does this:
Select Source...	The application requests that the Source Manager's Select Source Dialog Box appear (or it may display its own version). After the user selects the Source they want to use, control returns to the application.
Acquire...	The application requests that the Source display its user interface. (Again, the application can create its own version of a user interface or display no user interface.)

Detailed information on the operations used by the application to successfully acquire data is provided later in this chapter in "Controlling a TWAIN Session from Your Application" on page 3-9.

Include the TWAIN.H File in Your Application

The `TWAIN.H` file that is shipped with this TWAIN Developer's Toolkit contains all of the critical definitions needed for writing a TWAIN-compliant application or Source. Be sure to include it in your application's code and print out a copy to refer to while reading this chapter.

The `TWAIN.H` file contains:

Category	Prefix for each item
Data Groups	`DG_`
Data Argument Types	`DAT_`
Messages	`MSG_`
Capabilities	`CAP_`, `ICAP_`, or `ACAP_`
Return Codes	`TWRC_`
Condition Codes	`TWCC_`
Type Definitions	`TW_`
Structure Definitions	`TW_`
Entry points	These are `DSM_Entry` and `DS_Entry`

In addition, there are many constants defined in `TWAIN.H` which are not listed here.

Alter the Application's Event Loop

The application passes the request for all actions to the Source Manager via the DSM_Entry function call, which contains an operation triplet describing the requested action. In code form, the DSM_Entry function looks like this:

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

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

The DG, DAT, and MSG parameters contain the operation triplet. The parameters must follow these rules:

pOrigin

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

pDest

Set to NULL if the operation's final destination is the Source Manager.

Otherwise, set to point to a valid TW_IDENTITY structure for an open Source.

DG_xxxx

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

DAT_xxxx

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

MSG_xxxx

Message specifies the action to be taken.

pData

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

Operation Triplets - Application to Source Manager

The following operation triplets 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

DG_CONTROL / DAT_PARENT

`MSG_CLOSEDSM` :	Prepare Source Manager for unloading
`MSG_OPENDSM` :	Initialize the Source Manager

DG_CONTROL / DAT_STATUS

`MSG_GET` :	Return Source Manager’s current Condition Code

Operation Triplets - Application to Source

The next group of operations are sent to a specific Source by the application. These operations are still passed via the Source Manager using the `DSM_Entry` call. The first set of triplets use the `DG_CONTROL` identification for their data group. These are operations that could be performed on any kind of TWAIN device. The second set of triplets use the `DG_IMAGE` identification for their data group which indicates these operations are specific to image data. Details about each operation are available in reference format in Chapter 7, "Operation Triplets".

Control Operations from Application to Source

DG_CONTROL / DAT_CAPABILITY

`MSG_GET`	Return a Capability’s available value(s) including the current and default values
`MSG_GETCURRENT`	Get a Capability’s current value
`MSG_GETDEFAULT`	Get a Capability’s preferred default value (Source specific)
`MSG_RESET`	Change a Capability’s current value to its TWAIN-defined default
`MSG_SET`	Change a Capability’s current value only (TWAIN 2.2 and higher)
`MSG_SETCONSTRAINT`	Change a Capability’s current, default, and available value(s) (Same functionality as `MSG_SET` prior to TWAIN 2.2)

DG_CONTROL / DAT_DEVICEEVENT

`MSG_GET`:	Get an event from the Source (issue this call only in response to a `DG_CONTROL` / `DAT_NULL` / `MSG_DEVICEEVENT` from the Source)

DG_CONTROL / DAT_EVENT

`MSG_PROCESSEVENT`	Pass an event to the Source from the application

DG_CONTROL / DAT_FILESYSTEM

`MSG_AUTOMATICCAPTUREDIRECTORY`	Select directory to receive automatically captured images
`MSG_CHANGEDIRECTORY`	Change the current domain, host, directory, or device.
`MSG_COPY`	Copy files
`MSG_CREATEDIRECTORY`	Create a directory
`MSG_DELETE`	Delete a file or directory

MSG_FORMATMEDIA	Format a storage device
MSG_GETCLOSE	Close a context created by a call to MSG_GETFILEFIRST
MSG_GETFIRSTFILE	Get the first file in a directory
MSG_GETINFO	Get information about the current file context
MSG_RENAME	Rename a file

DG_CONTROL / DAT_PASSTHRU / MSG_PASSTHRU

MSG_PASSTHRU	Special command for the use by Source vendors when writing diagnostic Applications

DG_CONTROL / DAT_PENDINGXFERS

MSG_ENDXFER	Application acknowledges or requests the end of data transfer
MSG_GET	Return the number of transfers the Source is ready to supply
MSG_RESET	Set the number of pending transfers to zero
MSG_STOPFEEDER	Stop ADF without ending session

DG_CONTROL / DAT_SETUPFILEXFER

MSG_GET	Return info about the file that the Source will write the acquired data into
MSG_GETDEFAULT	Return the default file transfer information
MSG_RESET	Reset current file information to default values
MSG_SET	Set file transfer information for next file transfer

DG_CONTROL / DAT_SETUPMEMXFER

MSG_GET	Return Source's preferred, minimum, and maximum transfer buffer sizes

DG_CONTROL / DAT_STATUS

MSG_GET	Return the current Condition Code from specified Source

DG_CONTROL / DAT_USERINTERFACE

MSG_DISABLEDS	Cause Source's user interface to be taken down
MSG_ENABLEDS	Cause Source to prepare to display its user interface

DG_CONTROL / DAT_XFERGROUP

MSG_GET	Return the Data Group (currently DG_IMAGE or a custom data group) for the upcoming transfer

There are additional DG_CONTROL operations for communications between the Source Manager and the Source. They are discussed in Chapter 5, "Source Implementation".

Image Operations from Application to Source

DG_IMAGE / DAT_CIECOLOR

MSG_GET	Return the CIE XYZ information for the current transfer

DG_IMAGE / DAT_GRAYRESPONSE

`MSG_RESET`	Reinstate identity response curve for grayscale data
`MSG_SET`	Source uses specified response curve on grayscale data

DG_IMAGE / DAT_IMAGEFILEXFER

`MSG_GET`	Initiate image acquisition using the Disk File transfer mode

DG_IMAGE / DAT_IMAGEINFO

`MSG_GET`	Return information that describes the image for the next transfer

DG_IMAGE / DAT_IMAGELAYOUT

`MSG_GET`	Describe physical layout / position of "original" image
`MSG_GETDEFAULT`	Default information on the layout of the image
`MSG_RESET`	Set layout information for the next transfer to defaults
`MSG_SET`	Set layout for the next image transfer

DG_IMAGE / DAT_IMAGEMEMXFER

`MSG_GET`	Initiate image acquisition using the Buffered Memory transfer mode

DG_IMAGE / DAT_IMAGEMEMFILEXFER

`MSG_GET`	Initiate image acquisition using the Buffered Memory transfer mode, but transferring the same data one would save to a file

DG_IMAGE / DAT_IMAGENATIVEXFER

`MSG_GET`	Initiate image acquisition using the Native transfer mode

DG_IMAGE / DAT_JPEGCOMPRESSION

`MSG_GET`	Return JPEG compression parameters for current transfer
`MSG_GETDEFAULT`	Return default JPEG compression parameters
`MSG_RESET`	Use Source's default JPEG parameters on JPEG transfers
`MSG_SET`	Use specified JPEG parameters on JPEG transfers

DG_IMAGE / DAT_PALETTE8

`MSG_GET`	Return palette information for current transfer
`MSG_GETDEFAULT`	Return Source's default palette information for current pixel type
`MSG_RESET`	Use Source's default palette for transfer of this pixel type
`MSG_SET`	Use specified palette for transfers of this pixel type

DG_IMAGE / DAT_RGBRESPONSE

`MSG_RESET`	Use Source's default (identity) RGB response curve
`MSG_SET`	Use specified response curve for RGB transfers

DG_AUDIO / DAT_AUDIOFILEXFER

`MSG_GET`	Transfers audio data in file mode

DG_AUDIO / DAT_AUDIOINFO

`MSG_GET`	Gets information about the current transfer

DG_AUDIO / DAT_AUDIONATIVEXFER

`MSG_GET`	Transfers audio data in native mode

DSM_Entry Parameters

The parameters for the `DG_xxxx`, `DAT_xxxx`, and `MSG_xxxx` fields are determined by the operation triplet. The other parameters are filled as follows:

- `pOrigin`

  Refers to a copy of the application's `TW_IDENTITY` structure.
- `pDest`

  If the operation's destination is the Source Manager: Always holds a value of `NULL`. This indicates to the Source Manager that the operation is to be consumed by it not passed on to a Source.

  If the operation's destination is a Source: This parameter references a copy of the Source's `TW_IDENTITY` structure that is maintained by the application. The application received this structure in response to the `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS` operation sent from the application to the Source Manager. This is discussed more in the next section ("Controlling a TWAIN Session from Your Application" - State 3 to 4).
- `pData`

  Always references a structure or variable corresponding to the TWAIN type specified by the `DAT_xxxx` parameter. Typically, but not always, the data argument type name corresponds to a `TW_xxxx` data structure name. For example, the `DAT_IDENTITY` argument type uses the corresponding `TW_IDENTITY` data structure. All data structures can be seen in the file called `TWAIN.H`. The application is responsible for allocating and deallocating the structure or element and assuring that `pData` correctly references it.

  Note that there are two cases when the Source, rather than the application, allocates a structure that is used during an operation.

  - One occurs during `DG_CONTROL / DAT_CAPABILITY / MSG_GET`, `MSG_GETCURRENT`, `MSG_GETDEFAULT`, and `MSG_RESET` operations. The application still allocates *`pData` but the Source allocates a structure referenced by *`pData` called a "container structure".
  - The other occurs during the `DG_IMAGE / DAT_JPEGCOMPRESSION` operations. The topic of data compression is covered in Chapter 4, "Advanced Application Implementation".

  In all cases, the application still deallocates all structures.

Application Callback Function

The following TWAIN triplet is used, by the application, to register a function to receive callback messages from the Source:

- `DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK`

Note that the older event loop method still works on Windows, but it is recommended to use Callback. For the older event loop method refer to the TWAIN 1.9 Specification for implementation. Applications will register the callback after opening the DS using the `DG_CONTROL/ DAT_CALLBACK/ MSG_REGISTER_CALLBACK` triplet.

The callback function takes the form:

```
TW_UINT16 TWAIN_callback(pTW_IDENTITY pOrigin,
   pTW_IDENTITY pDest,
   TW_UINT32 DG,
   TW_UINT16 DAT,
   TW_UINT16 MSG,
   TW_MEMREF pData)
{
    // The message should not be processed here.
    // A flag is set so the Message can be processed in the same
     thread that Enabled the Data Source.
     m_Message = MSG;
    return TWRC_SUCCESS;  // or failure etc
}
```

An application registers the callback function in the following fashion:

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

The application passes the request for the action to the Source Manager via the `DSM_Entry` function call which contains an operation triplet describing the requested action.

Controlling a TWAIN Session from Your Application

In addition to the preparations discussed at the beginning of this chapter, the application must be modified to actually initiate and control a TWAIN session.

The session consists of the seven states of the TWAIN protocol as introduced in the Technical Overview. However, the application is not forced to move the session from State 1 to State 7 without stopping. For example, some applications may choose to pause in State 3 and move among the higher states (4 - 7) to repeatedly open and close Sources when acquisitions are requested by the user. Another example of session flexibility occurs when an application transfers multiple images during a session. The application will repeatedly move the session from State 6 to State 7 then back to State 6 and forward to State 7 again to transfer the next image.

For the sake of simplicity, this chapter illustrates moving the session from State 1 to State 7 and then backing it out all the way from State 7 to State 1. The diagram on the next page shows the operation triplets that are used to transition the session from one state to the next. Detailed information about each state and its associated transitions follow. The topics include:

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

- State 2 to 3 - Open the Source Manager
- State 3 - Select the Source
- State 3 to 4 - Open the Source
- State 4 - Negotiate Capabilities with the Source
- State 4 to 5 - Request the Acquisition of Data from the Source
- State 5 to 6 - Recognize that the Data Transfer is Ready
- State 6 to 7 - Start and Perform the Transfer
- State 7 to 6 to 5 - Conclude the Transfer
- State 5 to 1 - Disconnect the TWAIN Session

Note: Sources and Applications that support the `DAT_FILESYSTEM` operation may negotiate and select different device contexts immediately after the opening of a Source. For example, an Application may choose to browse through the stored images on a digital camera, rather than treat it as a real-time capture device.

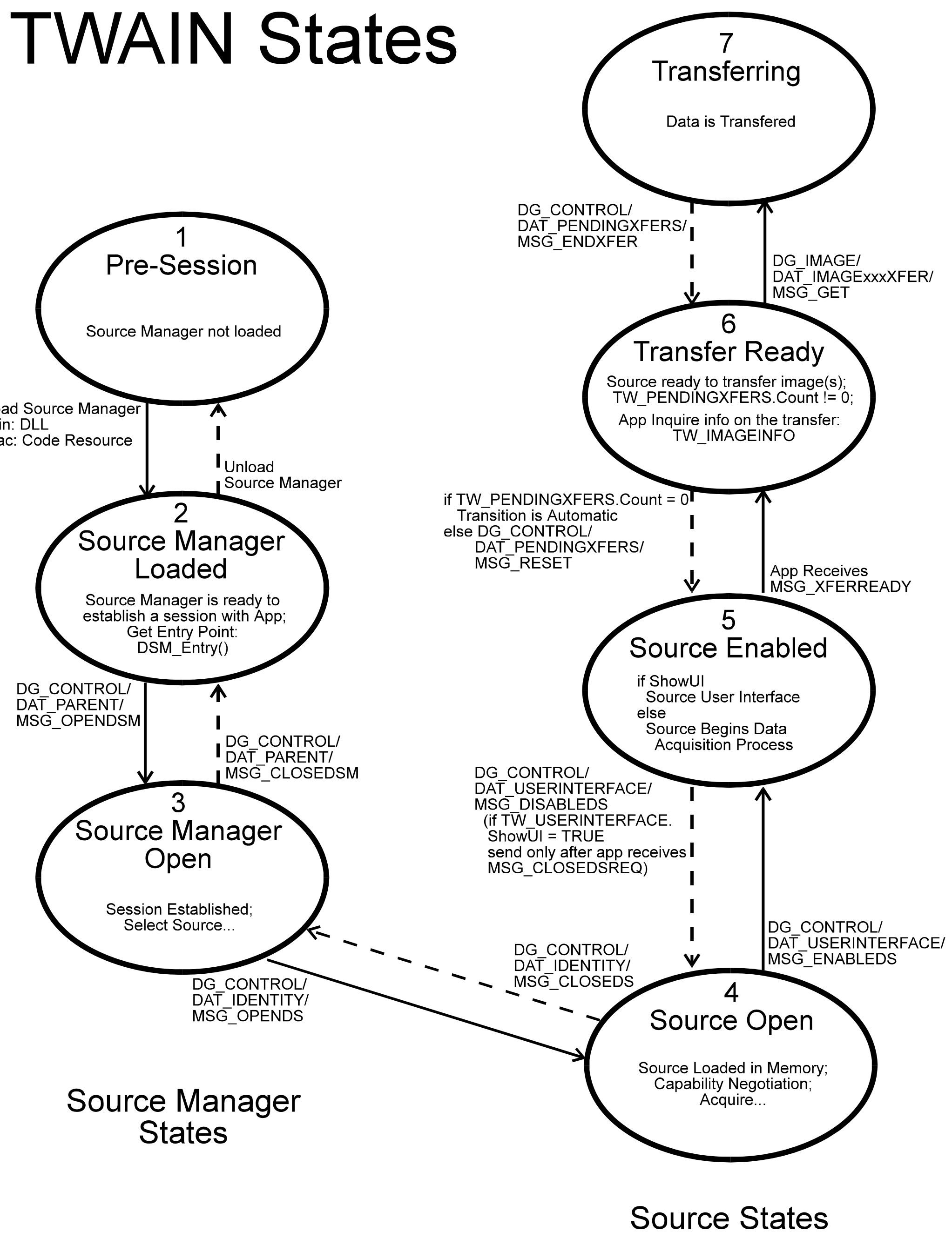


Figure 3-2 TWAIN States

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

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

Operations Used:

No TWAIN operations are used for this transition. Instead it is an OS specific operation, please refer to the Operating System chapter.

State 2 to 3 - Open the Source Manager

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

One Operation is Used:

`DG_CONTROL` / `DAT_PARENT` / `MSG_OPENDSM`

pOrigin

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

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

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

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

```
typedef struct {
   TW_UINT32     Id; /* Unique number assigned by DSM for
                     identification*/
   TW_VERSION    Version;
   TW_UINT16     ProtocolMajor;
   TW_UINT16     ProtocolMinor;
   TW_UINT32     SupportedGroups
   TW_STR32      Manufacturer;
   TW_STR32      ProductFamily;
   TW_STR32      ProductName;
} TW_IDENTITY, FAR *pTW_IDENTITY;
```

pDest

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

`pData`

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

- **On Windows:** `pData` points to the window handle (`hWnd`) that will act as the Source's "parent". The Source Manager will maintain a copy of this window handle for posting messages back to the application.
- **On Macintosh:** `pData` should be a `NULL` value.
- **On Linux:** `pData` should be a `NULL` value.

How to Initialize the TW_IDENTITY Structure

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

```
TW_IDENTITY    AppID;          // App's identity structure
  AppID.Id = 0;               // Initialize to 0 (Source Manager
                                  // will assign real value)
  AppID.Version.MajorNum = 3;   //Your app's version number
  AppID.Version.MinorNum = 5;
  AppID.Version.Language = TWLG_ENGLISH_USA;
  AppID.Version.Country  = TWCY_USA;
  lstrcpy (AppID.Version.Info, "Your App's Version String");
  AppID.ProtocolMajor = 2;     //Use yours not the one from twain.h
  AppID.ProtocolMinor = 2;     //Use yours not the one from twain.h
  AppID.SupportedGroups = DF_APP2 | DG_IMAGE | DG_CONTROL;
  lstrcpy (AppID.Manufacturer, "App's Manufacturer");
  lstrcpy (AppID.ProductFamily, "App's Product Family");
  lstrcpy (AppID.ProductName, "Specific App Product Name");
```

On Windows: Using DSM_Entry to open the Source Manager

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

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

On Macintosh: Using DSM_Entry to open the Source Manager

```
rc = DSM_Entry(&AppID,

     NULL,

     DG_CONTROL,

     DAT_PARENT,

     MSG_OPENDSM,

     NULL);
```

On Linux: Using DSM_Entry to open the Source Manager

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

where AppID is the `TW_IDENTITY` structure that the application set up to identify.

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

State 3 - Select the Source

The Source Manager has just been opened and is now available to assist your application in the selection of the desired Source.

`DG_CONTROL / DAT_PARENT / MSG_OPENDSM`. If it finds `DF_DSM2` then the Application must issue the `DG_CONTROL / DAT_ENTRYPOINT / MSG_GET` call before it opens the Source. This takes the form:

`DG_CONTROL / DAT_ENTRYPOINT / MSG_GET`

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Set to `NULL`.

pData

Points to a structure of type `TW_ENTRYPOINT`

The Source Manager returns pointers to functions that the Application must use when managing memory that is either freed or allocated by the Source.

One Operation is Used:

`DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT`

pOrigin

Points to the application's `TW_IDENTITY` structure. The desired data type should be specified by the application. This was done when you initialized the `SupportedGroups` field in your application's `TW_IDENTITY` structure.

This causes the Source Manager to make available for selection by the user only those Sources that can provide the requested data type(s). All other Sources are grayed out. (Note, if more than one data type were available, for example image and text, and the application wanted to accept both types of data, it would do a bit-wise OR of the types' constants and place the results into the `SupportedGroups` field.)

pDest

Set to `NULL`.

pData

Points to a structure of type `TW_IDENTITY`. The application must allocate this structure prior to making the call to `DSM_Entry`. Once the structure is allocated, the application must:

- Set the `Id` field to zero.
- Set the `ProductName` field to the null string ("`\0`"). (If the application wants a specific Source to be highlighted in the Select Source dialog box, other than the system default, it can enter the ProductName of that Source into the ProductName field instead of null. The system default Source and other available Sources can be determined by using the `DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT`, `MSG_GETFIRST` and `MSG_GETNEXT` operations.)

Additional fields of the structure will be filled in by the Source Manager during this operation to identify the selected Source. Make sure the application keeps a copy of this updated structure after completing this call. You will use it to identify the Source from now on.

The most common approach for selecting the Source is to use the Source Manager's Select Source dialog box. This is typically displayed when the user clicks on your Select Source option. To do this:

1. The application sends a `DG_CONTROL` / `DAT_IDENTITY` / `MSG_USERSELECT` operation to the Source Manager to have it display its dialog box. The dialog displays a list of all Sources that are installed on the system that can provide data of the type specified by the application. It highlights the Source that is the system default unless the application requests otherwise.

2. The user selects a Source or presses the **Cancel** button. If no devices are available, the Select Source Dialog's **Select/OK** button will be grayed out and the user will have no choice but to select Cancel.

3. The application must check the Return Code of `DSM_Entry` to determine the user's action.

   a: **If** `TWRC_SUCCESS`: Their selected Source is listed in the `TW_IDENTITY` structure pointed to by the `pData` parameter and is now the default Source.

   b: **If** `TWRC_CANCEL`: The user either clicked **Cancel** intentionally or had no other choice because no devices were listed. Do not attempt to open a Source.

   c: **If** `TWRC_FAILURE`: Use the `DG_CONTROL` / `DAT_STATUS` / `MSG_GET` operation (sent to the Source Manager) to determine the cause. The most likely cause is a lack of sufficient memory.

As an alternative to using the Source Manager's Select Source dialog, the application can devise its own method for selecting a Source. For example, it could create and display its own user interface or simply select a Source without offering the user a choice. This alternative is discussed in Chapter 4, "Advanced Application Implementation".

State 3 to 4 - Open the Source

The Source Manager is open and able to help your application open a Source.

One Operation is Used:

`DG_CONTROL` / `DAT_IDENTITY` / `MSG_OPENDS`

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Set to `NULL`.

pData

Points to a structure of type `TW_IDENTITY`.

Typically, this points to the application's copy of the Source's `TW_IDENTITY` structure filled in during the `MSG_USERSELECT` operation previously.

However, if the application wishes to have the Source Manager simply open the default Source, it can do this by setting the `TW_IDENTITY`.ProductName field to "`\0`" (null string) and the `TW_IDENTITY`.Id field to zero.

During the `MSG_OPENDS` operation, the Source Manager assigns a unique identifier to the Source and records it in the `TW_IDENTITY`.Id field. Copy the resulting `TW_IDENTITY` structure. Once the Source is opened, the application will point to this resulting structure via the `pDest` parameter on every call that the application makes to `DSM_Entry` where the desired destination is this Source.

Note: The user is not required to take advantage of the Select Source option. They may click on the Acquire option without having selected a Source. In that case, your application should open the default Source. **The default source is either the last one used by the user or the last one installed.**

State 4 - Negotiate Capabilities with the Source

At this point, the application has a structure identifying the open Source. Operations can now be directed from the application to that Source. To receive a single image from the Source, only one capability, `CAP_XFERCOUNT`, must be negotiated now. All other capability negotiation is optional.

Note: When the application detects `DF_DSM2` in its `TW_IDENTITY.SupportedGroups`, then the Application must use the `DSM_MemAllocate`, `DSM_MemFree`, `DSM_MemLock` and `DSM_MemUnlock` functions it got from `DG_CONTROL` / `DAT_ENTRYPOINT` / `MSG_GET` to manage any memory it uses with the Source.

Two Operations are Used:

`DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GET`

`DG_CONTROL` / `DAT_CAPABILITY` / `MSG_SET`

The parameters for each of the operations, in addition to the triplet, are these:

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Points to the desired Source's `TW_IDENTITY` structure. The Source Manager will receive the `DSM_Entry` call, recognize that the destination is a Source rather than itself, and pass the operation along to the Source via the `DS_Entry` function.

`pData`

Points to a structure of type `TW_CAPABILITY`.

The definition of `TW_CAPABILITY` is:

```
typedef struct {
  TW_UINT16   Cap;      /* ID of capability to get or set */
  TW_UINT16   ConType;   /* TWON_ONEVALUE, TWON_RANGE,     */
                       /* TWON_ENUMERATION or TWON_ARRAY */
  TW_HANDLE  hContainer; /* Handle to container of type    */
                       /* ConType                       */
} TW_CAPABILITY, FAR *pTW_CAPABILITY;
```

The Source allocates the container structure pointed to by the `hContainer` field when called by the `MSG_GET` operation. The application allocates it when calling with the `MSG_SET` operation. Regardless of who allocated it, the application deallocates the structure either when the operation is complete or when the application no longer needs to maintain the information.

Each operation serves a special purpose:

MSG_GET

Since Sources are not required to support all capabilities, this operation can be used to determine if a particular TWAIN-defined capability is supported by a Source. The application needs to set the Cap field of the `TW_CAPABILITY` structure to the identifier representing the capability of interest. The constants identifying each capability are listed in the `TWAIN.H` file.

If the capability is supported and the operation is successful, it returns the Current, Default, and Available values. These values reflect previous `MSG_SET` operations on the capability which may have altered them from the TWAIN default values for the capability.

This operation may fail due to several causes. If the capability is not supported by the Source, the Return Code will be `TWRC_FAILURE` and the condition code will be one of the following:

`TWCC_CAPUNSUPPORTED`	Capability not supported by Source
`TWCC_CAPBADOPERATION`	Operation not supported by capability
`TWCC_CAPSEQERROR`	Capability has dependency on other capability

Applications should be prepared to receive the condition code `TWCC_BADCAP` from Sources written prior to TWAIN 1.7, which maps to any of the three situations mentioned above.

MSG_SET

Changes the Current Value(s) of the specified capability to those requested by the application.

If the Return Code indicates `TWRC_FAILURE`, check the Condition Code. A code of `TWCC_BADVALUE` can mean:

- The application sent an invalid value for this Source's range.
- The Source does not allow the setting of this capability.
- The Source doesn't allow the type of container used by the application to set this capability.

Capability negotiation gives the application developer power to guide the Source and control the images they receive from the Source. The negotiation typically occurs during State 4. The following material illustrates only one very basic capability and container structure. Refer to Chapter 4, "Advanced Application Implementation" for a more extensive discussion of capabilities including information on how to delay the negotiation of some capabilities beyond State 4.

Note: It is important here to once again remind application writers to always check the return code from any negotiated capabilities transactions.

MSG_SETCONSTRAINT

Changes the Current Value(s) of the specified capability to those requested by the application, and constrains the allowable contents of an array, enumeration or range container.

If the Return Code indicates `TWRC_FAILURE`, check the Condition Code. A code of `TWCC_BADVALUE` can mean:

- The application sent an invalid value for this Source's container.
- The Source doesn't allow the type of container used by the application to set this capability.

Capability negotiation gives the application developer power to guide the Source and control the images they receive from the Source. The negotiation typically occurs during State 4. The following material illustrates only one very basic capability and container structure. Refer to Chapter 4, "Advanced Application Implementation" for a more extensive discussion of capabilities including information on how to delay the negotiation of some capabilities beyond State 4.

Note: It is important here to once again remind application writers to always check the return code from any negotiated capability transactions.

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

The capability that specifies how many images an application can receive during a TWAIN session is `CAP_XFERCOUNT`. All Sources must support this capability. Possible values for `CAP_XFERCOUNT` are:

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

The default value allows multiple images to be transferred. The code example online illustrates the setting of a capability and specifically shows how to limit the number of images to one.

See `set_CapabilityOneValue` function for live code example in TwainApp.cpp at http://twain-samples.svn.sourceforge.net

Other Capabilities

Image Type

The application should be aware of the Source's `ICAP_PIXELTYPE` and `ICAP_BITDEPTH`. If your application cannot accept all of the Source's Available Values, capability negotiation should be done. (Refer to Chapter 4, "Advanced Application Implementation".)

Transfer Mode

The default transfer mode is Native. That means the Source will access the largest block of memory available and use it to transfer the entire image to the application at once. If the available memory is not large enough for the transfer, then the Source should fail the transfer. The application does not need to do anything to select this transfer mode. If the application wishes to specify a different transfer mode, Disk File or Buffered Memory, further capability negotiation is required. (Refer to Chapter 4, "Advanced Application Implementation".)

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

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

One Operation is Used:

DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Points to the Source's `TW_IDENTITY` structure.

`pData`

Points to a structure of type `TW_USERINTERFACE`.

The definition of `TW_USERINTERFACE` is:

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

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

The Application will set the `ModalUI` field to `TRUE` if it wants the Source to run modal, and `FALSE` if it wants the Source to run modeless. Please note that to successfully run modal, it may be necessary for the application to disable inputs to its windows while the Source's GUI is running.

- **On Windows** - It is not recommended to set this field to `TRUE`. The Source may ignore this value and use `FALSE` if it is version 2.1 or lower. If both Source and the Application are 2.2 or higher, then the Source must return `TWRC_CHECKSTATUS` if it does not support requested value.
- **On Macintosh** - It is recommended to use this field.
- **On Linux** - This field is not used.

The application sets the `hParent` field differently depending on the platform on which the application runs.

- **On Windows** - The application should place a handle to the Window that is acting as the Source's parent.
- **On Macintosh** - The application sets this field to `NULL`.
- **On Linux** - The application sets this field to `NULL`.

In response to the user choosing the application's Acquire menu option, the application sends this operation to the Source to enable it. The application typically requests that the Source display the Source's user interface to assist the user in acquiring data. If the Source is told to display its user interface, it will display it when it receives the operation triplet. Modal and Modeless interfaces are discussed in Chapter 4, "Advanced Application Implementation" and Chapter 5, "Source Implementation". Sources **must** check the `ShowUI` field and return an error if they cannot support the specified mode. In other words it is unacceptable for a source to ignore a `ShowUI = FALSE` request and still activate its user interface. The application may develop its own user interface instead of using the Source's. This is discussed in Advanced Application Implementation.

Note: Once the Source is enabled via the `DG_CONTROL / DAT_USERINTERFACE/ MSG_ENABLEDS` operation, all events that enter the application's main event loop must be immediately forwarded to the Source. The explanation for this is given in Chapter 12, "Operating System Dependencies" when modifying the event loop in preparation for a TWAIN session.

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

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

No Operations (from the application) are Used:

This transition is not triggered by the application sending an operation. The Source causes the transition.

- **On Windows** - while the application has the Source enabled, the application is forwarding all events in its event loop to the Source by using the `DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT` operation.

Refer to Chapter 12, "Operating System Dependencies" for this.

The application will receive one of these `MSG_XFERREADY`, `MSG_CLOSEDSREQ`, or `MSG_CLOSEDSOK` messages in its callback function. When the Application receives `MSG_XFERREADY` it will transit from `State 5` to `State 6`.

For legacy methods, please refer to version 1.9 of the Specification.

State 6 to 7 - Start and Perform the Transfer

The Source indicated it is ready to transfer data. It is waiting for the application to inquire about the image details, initiate the actual transfer, and, hence, transition the session from State 6 to 7.

Two Operations are Used:

The application may want to inquire about the image data that it will be receiving. The `DG_IMAGE / DAT_IMAGEINFO / MSG_GET` operation allows this. Other operations, such as `DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`, provide additional information. This information can be used to determine if the application actually wants to initiate the transfer.

`DG_IMAGE / DAT_IMAGEINFO / MSG_GET`

pOrigin
Points to the application's `TW_IDENTITY` structure.

pDest
Points to the Source's `TW_IDENTITY` structure.

pData

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

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

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

The application may want to inquire about the image data that it will be receiving. The `DG_IMAGE / DAT_IMAGEINFO / MSG_GET` operation allows this. Other operations, such as `DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`, provide additional information. This information can be used to determine if the application actually wants to initiate the transfer.

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

If the initiation (`DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`) fails, the session does not transition to State 7 but remains in State 6.

DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Points to the Source's `TW_IDENTITY` structure.

pData

Points to an OS specific native Image format returned by the Data Source. For more information see Chapter 12, "Operating System Dependencies".

- **On Windows:** The Source will set `pData` to point to a device-independent bitmap (DIB) that it allocates.
- **On Macintosh:** The Source will set `pData` to point to a TIFF that it allocates if both the application and the data source are TWAIN 2.4 and later: The Source will set `pData` to point to a `PicHandle` that it allocates if either the application or the data source is TWAIN 2.3 and earlier.

- **On Linux:** The Source will set `pData` to point to a TIFF that it allocates.

The application is responsible for de-allocating the memory block holding the Native-format image.

The function `initiateTransfer_Native` illustrates how to get information about the image that will be transferred, and how to actually perform a native transfer.

Refer to TwainApp.cpp at http://twain-samples.svn.sourceforge.net.

State 7 to 6 to 5 - Conclude the Transfer

While the transfer occurs, the session is in State 7. When the Source indicates via the Return Code that the transfer is done (`TWRC_XFERDONE`) or canceled (`TWRC_CANCEL`), the application needs to transition the session backwards.

One Operation is Used:

`DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER`

pOrigin
Points to the application's `TW_IDENTITY` structure.

pDest
Points to the Source's `TW_IDENTITY` structure.

pData
Points to a structure of type `TW_PENDINGXFERS`.

The `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation is sent by the application to the Source at the end of every transfer, successful or canceled, to indicate the application has received all the data it expected.

After this operation returns, the application should examine the `pData->Count` field to determine if there are more images waiting to be transferred. The value of `pData->Count` indicates the following:

Value	Description
`pData->Count = 0`	If zero, the Source will "automatically" transition back to State 5 without the application needing to take any additional action. Application writers please make special note of this instance of an implied source transition. The application should return to its main event loop and await notification from the Source (either `MSG_XFERREADY` or `MSG_CLOSEDSREQ`).

Value	Description
`pData->Count = -1` or `pData->Count > 0`	The Source has more transfers available and is waiting in State 6. If the value is -1, that means the Source has another image available but it is unsure of how many more will be available. This might occur if the Source was controlling a device equipped with a document feeder and some unknown number of documents were stacked in that feeder. If the number of images is known, the Count will be a value greater than 0. Either way, the Source will remain in State 6 ready for the application to initiate another transfer. The Source will *NOT* send another `MSG_XFERREADY` to trigger this. The application should proceed as if it just received a `MSG_XFERREADY`.

If more images were pending and your application does not wish to transfer all of them, you can discard one or all pending images by doing the following:

- **To discard just the next pending image**, use the `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation. Then, check the Count field again to determine if there are additional images pending.
- **To discard all pending images**, use the `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_RESET` operation. Following successful execution of this operation, the session will be in State 5.

The function `DoAbortXfer` illustrates how to stop a transfer in TwainApp.cpp on http://twain-samples.svn.sourceforge.net.

State 5 to 1 - Disconnect the TWAIN Session

Once the application has acquired all desired data from the Source, the application can disconnect the TWAIN session. To do this, the application transitions the session backwards.

In the last section, the Source transitioned to State 5 when there were no more images to transfer (`TW_PENDINGXFERS.Count = 0`) or the application called the `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_RESET` operation to purge all remaining transfers. To back out the remainder of the session:

Three Operations (plus some platform-dependent code) are Used:

To move from State 5 to State 4

`DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_DISABLEDS`

pOrigin

Points to the application's `TW_IDENTITY` structure.

pDest

Points to the Source's `TW_IDENTITY` structure.

pData

Points to a structure of type `TW_USERINTERFACE`.

The definition of `TW_USERINTERFACE` is:

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

Its contents are not used.

Note the following:

- **If the Source's User Interface was displayed:** This operation causes the Source's user interface, if displayed during the transition from State 4 to 5, to be lowered. This operation is sent by the application in response to a MSG_CLOSEDSREQ from the Source. This request from the Source appears in the TWMessage field of the TW_EVENT structure. It is sent back from the DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT operation used by the application to send events to the application.
- **If the application did not have the Source's User Interface displayed:** The application invokes this command when all transfers have been completed. In addition, the application could invoke this operation to transition back to State 4 if it wanted to modify one or more of the capability settings before acquiring more data.

To move from State 4 to State 3

DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS

pOrigin

Points to the application's TW_IDENTITY structure.

pDest

Should reference a NULL value (indicates destination is Source Manager)

pData

Points to a structure of type TW_IDENTITY

This is the same TW_IDENTITY structure that you have used throughout the session to direct operation triplets to this Source.

When this operation is completed, the Source is closed. (In a more complicated scenario, if the application had more than one Source open, it must close them all before closing the Source Manager. Once all Sources are closed and the application does not plan to initiate any other TWAIN session with another Source, the Source Manager should be closed by the application.)

To move from State 3 to State 2

DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM

pOrigin

Points to the application's TW_IDENTITY structure.

pDest

Should reference a NULL value (indicates destination is Source Manager)

pData

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

On Windows: `pData` points to the window handle (`hWnd`) that acted as the Source's "parent".

On Macintosh: `pData` should be a `NULL` value.

On Linux: `pData` should be a `NULL` value.

To move from State 2 to State 1

Once the Source Manager has been closed, the application must unload it from memory before continuing.

See Chapter 12, "Operating System Dependencies" for more information.

TWAIN Session Review

Applications have flexibility regarding which state they leave their TWAIN sessions in between TWAIN commands (such as Select Source and Acquire).

For example:

- An application might load the Source Manager on start-up and unload it on exit. Or, it might load the Source Manager only when it is needed (as indicated by Select Source and Acquire).
- An application might open a Source and leave it in State 4 between acquires.

The following is the simplest view of application's TWAIN flow. All TWAIN actions are initiated by a TWAIN command, either user-initiated (Select Source and Acquire) or notification from the Source (`MSG_XFERREADY` and `MSG_CLOSEDSREQ`).

Application Receives	State	Application Action
Select Source...	1 -> 2	Load Source Manager
	2 -> 3	`DG_CONTROL / DAT_PARENT / MSG_OPENDSM`
		`DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT`
	3 -> 2	`DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM`
	2 -> 1	Unload Source Manager
Acquire...	1 -> 2	Load Source Manager
	2 -> 3	`DG_CONTROL / DAT_PARENT / MSG_OPENDSM`
	3 -> 4	`DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`
		Capability Negotiation
	4 -> 5	`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

Application Receives	State	Application Action
MSG_XFERREADY	6	For each pending transfer:
		DG_IMAGE / DAT_IMAGEINFO / MSG_GET
		DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
		DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
	6 -> 7	DG_IMAGE / DAT_IMAGExxxxXFER / MSG_GET
	7 -> 6	DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
	6 -> 5	Automatic transition to State 5 if TW_PENDINGXFERS.Count equals 0.
MSG_CLOSEDSREQ	5 -> 4	DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS
	4 -> 3	
	3 -> 2	DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS
	2 -> 1	DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM
		Unload the Source Manager

Error Handling

Your application must be robust enough to recognize and handle error conditions that may occur during a TWAIN session. Every TWAIN operation triplet has a defined set of Return Codes and Conditions Codes that it may generate. These codes are listed on the reference pages for each triplet located in Chapter 7, "Operation Triplets". Be sure to check the Return Code following every call to the DSM_Entry function. If it is TWRC_FAILURE, make sure your code checks the Condition Code and handles the error condition appropriately.

The following code segment illustrates the basic operations for doing this:

```
TW_STATUS    twStatus;
if (rc == TWRC_FAILURE)
      //check Condition Code
      rc = (*pDSM_Entry) (&AppID,
            &SourceID,
            DG_CONTROL,
            DAT_STATUS,
            MSG_GET,
           (TW_MEMREF)&twStatus);
      switch (twStatus.ConditionCode)
            //handle each possible Condition Code for the operation
```

Common Types of Error Conditions

Sequence Errors

The TWAIN protocol allows the invoking of specific operations only while the TWAIN session is in a particular state or states. The valid states for each operation are listed on the operation’s reference pages inChapter 7, "Operation Triplets". If an operation is called from an inappropriate state, the call will return an error, TWRC_FAILURE, and set the Condition Code to TWCC_SEQERROR. Although this error should not occur if both the application and Source are behaving correctly, it is possible for the session to get out of sync.

If this error occurs, correct it by assuming the Source believes it is in State 7. The application should invoke the correct operations to back up from State 7 to State 6 and so on down the states until an operation succeeds. Then, the application can continue or terminate the session.

The following pseudo code illustrates this:

```
if (TWCC_SEQERROR)
   //  Assume State 7, start backing out from State 7 until
   //  the Condition Code != TWCC_SEQERROR
   State 7 to 6    DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
   State 6 to 5    DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET
   State 5 to 4    DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS
   State 4 to 3    DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS
```

Low Memory Errors

Another common type of error condition occurs when insufficient memory is available to perform a requested operation. The most likely times for this to occur are:

- When a Source is being opened
- When a Source is being enabled
- During a Native image transfer

Your application must check the Return Code and Condition Code (TWRC_FAILURE / TWCC_LOWMEMORY) to recognize this. Your application may be able to free up sufficient memory to continue or it must quit.

State Transition Operation Triplet Errors

Many operations normally cause state transitions. If one of these operations fails, for example, returns TWRC_FAILURE, do not make the state transition. The application must check the Return Code following every operation and update the current state only if the operation succeeds.

An implied state transition during DG_CONTROL/DAT_PENDINGXFERS/ MSG_ENDXFER deserves special note here. If the Count field of the TW_PENDINGXFERS structure is zero then the source will automatically transition back to State 5. Application writers should be aware of this condition and react accordingly.

Error Handling and State Transitions

It is possible that during execution of any triplet that the data source will fail unexpectedly. It is very important that applications pay attention to the TWAIN State of the data source at the time of failure. A hanging or deadlock condition will occur if the application fails to recover from error conditions with the proper state transitions. Most error handling is fairly obvious, however the following items have been mishandled in the past.

Failing Transition to State 5

A data source may fail a call to DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS unexpectedly. It is important to note that if an application requests the User Interface be suppressed, and the data source returns a code of TWRC_CHECKSTATUS, this means only that User Interface suppression was not possible. The transition to State 5 still occurred. If the application does not like this condition, then it may call MSG_DISABLEDS to close the data source without further user interaction. A return code of TWRC_FAILURE indicates that the transition to State 5 has not occurred.

Failure During State 6 or 7

It is important to be aware that when an error occurs during image transfer, a state transition to State 5 is not implicit. A call to DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET or MSG_ENDXFER is required for a state transition back to State 5. If an applications calls MSG_DISABLEDS immediately after such a failure without first making the required calls to DAT_PENDINGXFERS, the resulting behavior of the data source will not be predictable. The data source should fail any call to MSG_DISABLEDS outside of State 5.

Best Practices for TWAIN Compliant Applications

The following items are covered in this section:

- Handling Status Returns
- States 1, 2, 3: Finding and Opening a Data Source
- States 4, 5: Capability Negotiation
- States 6, 7: Transferring Data
- Stepping Back Down the States

Handling Status Returns

TWAIN supports a small number of status return codes and condition codes. If an operation returns TWRC_FAILURE, then the application must immediately issue the DG_CONTROL / DAT_STATUS / MSG_GET operation to collect the condition code.

The following tables describe the meaning for each return code and condition code, and explains the action that an application should take in response.

Return Code	Meaning / Action
TWRC_CANCEL	Intended for use with the DAT_IMAGE*XFER operations. Operation has been canceled.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.

Return Code	Meaning / Action
TWRC_CHECKSTATUS	Intended for use with DAT_CAPABILITY and DAT_IMAGELAYOUT. Operation failed to completely perform the desired operation. For example, setting ICAP_BRIGHTNESS to 3 when its range is -1000 to 1000 with a step of 200. The data source may opt to set the value to 0 and return this status.
	The application should confirm its last setting, if it depends on getting the exact value it requested.
TWRC_DATANOTAVAILABLE	Intended for use with DAT_EXTIMAGEINFO. There is no data available for the requested TWEI_ item.
	Scanning may continue. The decision to continue with scanning is at the discretion of the application, depending on which field reported this status.
TWRC_DSEVENT	Intended for use with DAT_EVENT. The data source processed the event.
	The application must not take any further action on this message.
TWRC_ENDOFLIST	Intended for use with DAT_IDENTITY and DAT_FILESYSTEM.
	There are no more items to enumerate in this list. If a call is needed to close the list, it must be called next.
TWRC_FAILURE	May be returned by any operation. An error has occurred.
	The application must call DAT_STATUS, and refer to the condition code for more information.
TWRC_INFONOTSUPPORTED	Intended for use with DAT_EXTIMAGEINFO. The requested TWEI_ data is either not supported by this data source, or is not supported for this particular image.
	Scanning may continue. The decision to continue with scanning is at the discretion of the application, depending on which field reported this status.
TWRC_NOTDSEVENT	Intended for use with DAT_EVENT. The data source did not process the event.
	The application passes the message to its own dialogs.
TWRC_SUCCESS	Operation was successful.
	The application continues as normal.
TWRC_XFERDONE	Intended for use with the DAT_IMAGE*XFER operations. The image has been fully transferred.
	The application must be in state 7. It should call DAT_IMAGEINFO or DAT_EXTIMAGEINFO, if it needs to collect metadata for this image.

Condition Code	Meaning / Action
TWCC_BADCAP	Intended for use with DAT_CAPABILITY. Returned by pre-1.7 data sources to indicate that the capability is not supported, that the value was bad, or that the desired value could not be set at this time.
	The application should use the MSG_GET call for the operation to find out the constraints on the current values, if any, and to confirm the current value.
TWCC_BADDEST	May be returned by any operation (save for the DAT_PARENT operations). The TW_IDENTITY for the destination (the data source) does not match any items opened by MSG_OPENDS.
	The application may have a corrupt TW_IDENTITY, or it may have already closed the data source associated with the values in the TW_IDENTITY structure. It should return to state 3 if it wants to attempt to reopen the data source.
TWCC_BADPROTOCOL	May be returned by any operation. The requested DG_* / DAT_* / MSG_* is not supported by the data source.
	The application cannot perform this operation; any further action is at its discretion.
TWCC_BADVALUE	May be returned by any operation. The capability or operation has rejected the requested setting. Unless otherwise indicated in the Specification the original setting remains unchanged.
	The application should use the MSG_GET call for the operation to find out the constraints on the current values, if any, and to confirm the current value.
TWCC_BUMMER	May be returned by any operation. The data source is in a critical state.
	The application must save any important information and exit as soon as possible.
TWCC_CAPBADOPERATION	Intended for use with DAT_CAPABILITY. The capability does not support the requested operation.
	The application must use DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT to determine what operations a capability supports.
TWCC_CAPSEQERROR	Intended for use with DAT_CAPABILITY. The capability being MSG_SET or MSG_RESET cannot be modified due to a setting for a related capability. For instance, this may be returned by ICAP_CITTKFACTOR if ICAP_COMPRESSION is set to any value other than TWCP_GROUP32D.
	The application must set values in the correct order.

Condition Code	Meaning / Action
TWCC_CAPUNSUPPORTED	Intended for use with DAT_CAPABILITY. The capability is not supported.
	The application cannot negotiate this capability.
TWCC_CHECKDEVICEONLINE	May be returned for any operation in state 4 or higher, except ones that reduce state (DAT_PENDINGXFERS / MSG_ENDXER, DAT_PENDINGXFERS / MSG_RESET, DAT_USERINTERFACE / MSG_DISABLEDS, DAT_IDENTITY / MSG_CLOSEDS, DAT_PARENT / MSG_CLOSEDSM).
	When received the application uses CAP_DEVICEONLINE to determine when the device is available.
TWCC_DAMAGEDCORNER	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_DENIED	Intended for DAT_IMAGEFILEXFER and DAT_FILESYSTEM, the specified file or directory cannot be modified or deleted.
	If for DAT_IMAGEFILEXFER, then select a different filename and try again. If for DAT_FILESYSTEM, then alert the user.
TWCC_DOCTOODARK	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_DOCTOOLIGHT	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_FILEEXISTS	Intended for DAT_FILESYSTEM. The specified file or directory already exists.
	Pick a different file name and try again.
TWCC_FILENOTFOUND	Intended for DAT_IMAGEFILEXFER and DAT_FILESYSTEM. The specified file or directory cannot be found.
	If received during scanning the application may select a new directory path and try again, otherwise alert the user.
TWCC_FILEWRITEERROR	Intended for DAT_IMAGEFILEXFER and DAT_FILESYSTEM, the specified file or directory could not be written, usually indicating a disk full condition, though it may also indicate a file or directory that the user has no permission to write.
	If received during scanning the application may free resources and try again, otherwise alert the user.

Condition Code	Meaning / Action
TWCC_FOCUSERROR	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_INTERLOCK	Intended for use with the DAT_IMAGE*XFER operations.
	The application takes action to return to state 5 (the GUI is up) or state 4 (no GUI).
TWCC_LOWMEMORY	May be returned for any operation except ones that reduce state (DAT_PENDINGXFERS / MSG_ENDXER, DAT_PENDINGXFERS / MSG_RESET, DAT_USERINTERFACE / MSG_DISABLEDS, DAT_IDENTITY / MSG_CLOSEDS, DAT_PARENT / MSG_CLOSEDSM).
	When received the application may free resources and try again.
TWCC_MAXCONNECTIONS	Intended for use with DAT_IDENTITY / MSG_OPENDS. The data source cannot support any more connections to this device.
	Try again later.
TWCC_NODS	Intended for use with DAT_IDENTITY / MSG_OPENDS. The device is not online.
	Try again later.
TWCC_NOMEDIA	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_NOTEMPTY	Intended for use with DAT_FILESYSTEM. Directory is in use, and cannot be deleted.
	Delete the contents of the directory first, then delete the directory.
TWCC_OPERATIONERROR	The operation failed, but the user has already been informed by the data source.
	If CAP_INDICATORS is TRUE or TW_USERINTERFACE.ShowUI was set to TRUE, then the application should not issue its own message to the user. If these values are FALSE (meaning that no user interface is showing) then the application should alert the user and treat the condition as a TWCC_BADPROTOCOL. If the current state is 5, 6 or 7 return back to state 4 as soon as possible.
TWCC_PAPERDOUBLEFEED	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.
TWCC_PAPERJAM	Intended for use with the DAT_IMAGE*XFER operations.
	Call DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as one normally does after a successful transfer.

Condition Code	Meaning / Action
TWCC_SUCCESS	Operation was successful. This value should only be paired with TWRC_SUCCESS.
	If it is paired with another return code, like TWRC_FAILURE, treat it as TWCC_BADPROTOCOL. If it happens during state 5, 6 or 7, then return to state 4 as soon as possible.

States 1, 2, 3: Finding and Opening a Data Source

Registering as a TWAIN 2.x+ Application

The application loads the TWAIN Data Source Manager. When this is done it constructs a TW_IDENTITY structure, which includes the following flag in the TW_IDENTITY.SupportedGroups field: DF_APP2. It then issues the DG_CONTROL / DAT_PARENT / MSG_OPENDSM command with this TW_IDENTITY structure.

Confirming that the DSM is 2.x

The application examines the TW_IDENTITY.SupportedGroups field. If it contains the DF_DSM2 flag, then the DSM supports the TWAIN 2.x interface.

Issuing DAT_ENTRYPOINT

If the DF_DSM2 flag is detected, then the application issues the DG_CONTROL / DAT_ENTRYPOINT / MSG_GET call to retrieve function pointers for the memory allocation routines. The application must use these routines for any handles that it sends to the data source.

Selecting a Data Source

If the application wishes to use the default data source, it can issue the DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT command. This is preferred to calling DG_CONTROL / DAT_IDENTITY / MSG_OPENDS with an empty structure.

If the application wishes to get the list of available data sources it uses DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST and DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT, retaining the TW_IDENTITY of the data source it wants to use. This structure must not be modified in any way.

Use of DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT is discouraged because it is not localized for many languages, and because it is not available on systems other than Windows.

States 4, 5: Capability Negotiation

Overview

An application may negotiate settings with a data source in one of these ways:

- through the data source's built-in user interface
- using snapshots of the data source's previous settings
- through TWAIN's programmatic interface

In all cases the application is responsible for negotiating capabilities relating to data transfers. These capabilities come with defaults which must serve as the startup value for any data source (refer to the chapter on Capabilities to find the default values). These values will not appear on any data source's user interface, and they will not be affected by any data source's `DAT_CUSTOMDSDAT`:

```
CAP_SHEETCOUNT
CAP_XFERCOUNT
ICAP_XFERMECH
ICAP_UNITS
```

All other settings may be negotiated using one of the techniques described above.

The Data Source's User Interface

When calling `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `TW_USERINTERFACE.ShowUI` set to `TRUE`, an application may not make assumptions about what settings the user may change. Any programmatic changes (other than the items mentioned above) may be overridden by the user interface.

Use of `TW_USERINTERFACE.ModalUI` set to `TRUE` is discouraged. An application should take responsibility for disabling its interface if it wants modal behavior.

Using Snapshots

The application raises the data source's user interface using `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY`. If `MSG_CLOSEDSREQ` is received, then no action is taken (other than calling `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`).

If `MSG_CLOSEDSOK` is received, then immediately after calling `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS` the application calls `DG_CONTROL / DG_CUSTOMDSDATA / MSG_GET`.

The data returned by a data source in the `TW_CUSTOMDSDATA` structure is opaque; an application must not examine or alter this data in any way.

To restore settings, the application calls `DG_CONTROL / DG_CUSTOMDSDATA / MSG_SET` with the data from the previous `MSG_GET` operation.

This method requires a small amount of coding, but it allows the application to control all aspects of the data source, including custom features. It can be combined with the programmatic interface, using `DAT_CUSTOMDSDATA` to select most settings and the programmatic interface to make a smaller set of adjustments.

Programmatic Interface

Programmatic is the most advanced method of controlling a data source. It takes place in state 4. The application uses a functional approach; features are discovered through the TWAIN interface, not by hardcoded settings or relying on version numbers.

Available functions are determined through the following capabilities and operations (details on these items are covered later in the Specification):

```
CAP_SUPPORTEDCAPS
```

```
CAP_SUPPORTEDDATS
CAP_SUPPORTEDEXTIMAGEINFO
CAP_CUSTOMINTERFACEGUID
DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT
```

As a matter of good defensive programming an application should make no assumptions about the available capabilities, not even ones that are mandatory.

Assumptions should not be made about the value of capabilities when a data source is opened. Use the `DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL` operation or `DG_CONTROL / DAT_CUSTOMDSDATA / MSG_SET` to make sure that negotiation is starting from a known state.

Applications should follow the instructions in the Capability Ordering section to navigate the dependencies that capabilities have on one another. If a change is made out of order, then all capabilities dependent on that setting must be renegotiated.

Assumptions should not be made about the container types returned by a data source. For instance, a `DG_CONTROL / DAT_CAPABILITY / MSG_GET` for `ICAP_XRESOLUTION` may return `TW_ONEVALUE` if only one resolution is supported, `TW_ENUMERATION` if a small set of discontinuous resolutions is supported, or `TW_RANGE`.

Each of the containers has a field named `.Item`, `.ItemList` or value fields, which receives the new setting. These fields are variable, so a cast is needed. For example:

```
*((TW_UINT16*)&ptwonevalue->Item) = TWSX_MEMORY;
((TW_UINT16*)&ptwarray->ItemList)[2] = TWFT_RED;
((TW_FRAME*)&ptwenumeration->ItemList)[0] = twframeValue;
*((TW_FIX32*)&ptwrange->CurrentValue) = twfix32Value;
```

Strings in TWAIN are zero padded, not zero terminated. An application should not assume that the string will end with ASCII 0. Use memcpy to move the data to a string, and make sure to properly terminate it. For Mac OS X the first byte is a prefix indicating the valid number of characters in the string.

One safe method of setting any current value is to take the following steps:

- call `DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT` on the desired capability
- determine the container type from the `TW_CAPABILITY.ConType` field
- for `TWON_ONEVALUE`, do the following:
  - lock the container using the `DAT_ENTRYPOINT.DSM_MemLock` function
  - determine the item type from the container's `.ItemType` field
  - update the `.Item` field with the desired value
  - unlock the container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
- for `TWON_ARRAY`, do the following:
  - lock the container using the `DAT_ENTRYPOINT.DSM_MemLock` function
  - determine the item type from the container's `.ItemType` field

    - create a new container (with sufficient room for all the elements) using the `DAT_ENTRYPOINT.DSM_MemAllocate` function
    - lock the new container using the `DAT_ENTRYPOINT.DSM_MemLock` function
    - set the `.ItemType` field to the one reported by `MSG_GETCURRENT`
    - set the `.NumItems` field to the number of desired elements
    - set the `.ItemList field` with the desired values
    - unlock the new container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
    - unlock the original container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
    - free the original container using the `DAT_ENTRYPOINT.DSM_MemFree` function
- call `MSG_SET` with the updated container
- free the container using the `DAT_ENTRYPOINT.DSM_MemFree` function
- respond to the status returned by `MSG_SET`

If setting constraints, then do the following:

- call `DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT` to confirm that the capability has `TWQC_SETCONSTRAINT`
- call `DG_CONTROL / DAT_CAPABILITY / MSG_GET` on the desired capability
- determine the container type from the `TW_CAPABILITY.ConType` field
- for `TWON_ONEVALUE`, do the following:
    - lock the container using the `DAT_ENTRYPOINT.DSM_MemLock` function
    - determine the item type from the container's `.ItemType` field
    - update the `.Item` field with the desired value
    - unlock the container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
- for `TWON_ARRAY`, do the following:
    - lock the container using the `DAT_ENTRYPOINT.DSM_MemLock` function
    - determine the item type from the container's `.ItemType` field
    - create a new container (with sufficient room for all the elements) using the `DAT_ENTRYPOINT.DSM_MemAllocate` function
    - lock the new container using the `DAT_ENTRYPOINT.DSM_MemLock` function
    - set the `.ItemType` field to the one reported by `MSG_GET`
    - set the `.NumItems` field to the number of desired elements
    - set the `.ItemList` field with the desired values
    - unlock the new container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
    - unlock the original container using the `DAT_ENTRYPOINT.DSM_MemUnlock` function
    - free the original container using the `DAT_ENTRYPOINT.DSM_MemFree` function
- for `TWON_ENUMERATION`, do the following:
    - lock the container using the `DAT_ENTRYPOINT.DSM_MemLock` function

  - determine the item type from the container's .ItemType field
  - create a new container (with sufficient room for all the elements) using the DAT_ENTRYPOINT.DSM_MemAllocate function
  - lock the new container using the DAT_ENTRYPOINT.DSM_MemLock function
  - set the .ItemType field to the one reported by MSG_GETCURRENT
  - set the .NumItems field to the number of desired elements
  - set the .ItemList field with the desired values
  - set the .CurrentIndex field with the 0-based index of the .ItemList value that represents the current value
  - set the .DefaultIndex field to 0 (this value will be ignored by the data source)
  - unlock the new container using the DAT_ENTRYPOINT.DSM_MemUnlock function
  - unlock the original container using the DAT_ENTRYPOINT.DSM_MemUnlock function
  - free the original container using the DAT_ENTRYPOINT.DSM_MemFree function
- call MSG_SETCONSTRAINT with the updated container
- free the container using the DAT_ENTRYPOINT.DSM_MemFree function
- respond to the status returned by MSG_SETCONSTRAINT

The Graphical User Interface

This section assumes the application sets TW_USERINTERFACE.ShowUI to TRUE.

The application must not negotiate any values using DG_CONTROL / DAT_CAPABILITY / MSG_SET or MSG_RESET or MSG_RESETALL while in state 5.

Using MSG_ENABLEDS

The DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS operation raises a data source GUI that contains a Scan and a Cancel button. The Scan button may result in the receipt of a DG_CONTROL / DAT_NULL / MSG_XFERREADY message from the data source to the application, at which point the application must move to state 6 and begin image transfers.

The Cancel button causes the receipt of a DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ message from the data source to the application, at which point the application must issue the appropriate operations to the data source to take it from its current state (which may be 5, 6 or 7) to state 4.

Using MSG_ENABLEDSUIONLY

The DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY operation raises a data source GUI that contains an OK and a Cancel button. The OK button causes the receipt of a DG_CONTROL / DAT_NULL / MSG_CLOSEDSOK message from the data source to the application, at which point the application must issue DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS to the data source. The application must immediately take action on the OK request, for instance, calling DG_CONTROL / DAT_CUSTOMDSDATA / MSG_GET.

The Cancel button causes the receipt of a DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ message from the data source to the application, at which point the application must issue

DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS to the data source. Any changes made by the GUI will be discarded, but the application should consider issuing DG_CONTROL / DAT_CAPABILTY / MSG_RESETALL or DG_CONTROL / DAT_CUSTOMDSDATA / MSG_SET to make sure the data source is in a known state.

States 6, 7: Transferring Data

When the DG_CONTROL / DAT_NULL / MSG_XFERREADY message is received by the application, it moves to state 6 and begins transferring images. There are four transfer methods, as specified by ICAP_XFERMECH:

TWSX_NATIVE, which uses DAT_IMAGENATIVEXFER

TWSX_MEMORY, which uses DAT_IMAGEMEMXFER

TWSX_FILE, which uses DAT_IMAGEFILEXFER

TWSX_MEMFILE, which uses DAT_IMAGEMEMFILEXFER

Each method has advantages and disadvantages.

Using DAT_IMAGENATIVEXFER

Native transfers are the default and must be supported by all data sources. Being ‘native’ to the operating system they vary, with Bitmaps used on Windows and TIFF used on Mac OS X and Linux. Since they include meta-data describing the image no other call is required to view the image, and saving the image to disk is easy.

The chief drawback to native transfers is their size. Bitmaps cannot be compressed, and even TIFF files must be kept entirely in physical memory during the transfer. Some formats, like Bitmap may require additional image processing, such as changing the packing order for color data, or the location of the image origin, or realignment of each raster line.

Using DAT_IMAGEMEMXFER

Memory transfers must be supported by all data sources. They allow for efficient use of physical memory, since they transfer data using stripes or tiles. They support compressed images.

Memory transfers may not include any meta-data about the image, requiring a call to DG_IMAGE / DAT_IMAGEINFO / MSG_GET or DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET.

Using DAT_IMAGEFILEXFER

File transfers are optional for data sources. They are supported if the data source accepts a value of TWSX_FILE for ICAP_XFERMECH. They allow for efficient use of physical memory, since they transfer data using the disk drive. They support compressed images. Since they include meta-data describing the image no other call is required to view the image.

Being optional means that file transfer may not be an option for a given data source. There is also no guarantee that the data source supports the image file format needed by the application.

Using DAT_IMAGEMEMFILEXFER

Memory File transfers are optional for data sources. They are supported if the data source accepts a value of TWSX_MEMFILE for ICAP_XFERMECH. They allow for efficient use of physical memory, since they transfer data using stripes or tiles. They support compressed images. Since they include meta-data describing the image no other call is required to view the image.

Being optional means that memory file transfer may not be an option for a given data source. There is also no guarantee that the data source supports the image file format needed by the application.

The Image Transfer Loop

When the application receives `DG_CONTROL / DAT_NULL / MSG_XFERREADY` it goes to state 6 and transfers the first image.

`DAT_IMAGENATIVEXFER` and `DAT_IMAGEFILEXFER` only require one call to transfer the complete image. `DAT_IMAGEMEMXFER` and `DAT_IMAGEMEMFILEXFER` may require multiple calls returning `TWRC_SUCCESS` to indicate when there is more data to transfer for the current image. All of the calls return `TWRC_XFERDONE` when the image is completely transferred. Any other status is an error.

When `TWRC_XFERDONE` is received the application may call `DG_IMAGE / DAT_IMAGEINFO / MSG_GET` or `DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET` to get information about the image. Calling `DAT_IMAGEINFO` before `TWRC_XFERDONE` is received may result in an error or data that does not correspond exactly to the transferred image.

After either a successful transfer or an error the application calls `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`. It determines if there are more images to transfer by examining the value of `TW_PENDINGXFERS.Count`.

If there are more images the state goes to 6. If `TW_PENDINGXFERS.Count` is equal to zero then the state skips 6 and goes to 5.

The application has the option to discard an image by calling `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` without first transferring the data. It also has the option to gracefully exit the scanning state with `DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER`, or it can immediately abort scanning using `DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET`.

Stepping Back Down the States

The application and the data source both track a current state from 1 to 7 (with the most time spent in states 4 to 7). If they get out of sync, then the data source returns `TWRC_FAILURE / TWCC_SEQERROR` for an operation being called in the wrong state.

When this happens the application must take measures to resynchronize itself with the data source. The easiest way to go about this is to use the following call sequence, stopping at the desired state.

`DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` → state 7 to 6

`DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET` → state 6 to 5

`DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS` → state 5 to 4

`DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS` → state 4 to 3

Ignore the status returns from the calls prior to the one yielding the desired state. For instance, if a call during scanning returns `TWCC_SEQERROR` and the desire is to return to state 5, then use the following commands.

`DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` → state 7 to 6

DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET → state 6 to 5

Being sure to confirm that DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET returned success, the return status from DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER may be ignored.

Legacy Issues

Single Value Capabilities

Data Sources

On MSG_GET always use the preferred container (TW_ARRAY, TW_ENUMERATION or TW_RANGE), even if there is only one value available. Do not use TW_ONEVALUE, unless indicated by the Specification. For TW_RANGE the minimum and maximum values should be the same, and the step value should be zero.

Applications

Be prepared to accept any container returned by the data source.

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.

Also, owing to a bug in an old version of the sample driver, some Data Sources may report all of their possible bit depth values, instead of those that apply just to the current ICAP_PIXELTYPE value. For instance, with a setting of TWPT_RGB, ICAP_BITDEPTH may report allowed values of 1, 8 and 24, when only 24 is really permitted.

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.

CAP_ENDORSER vs CAP_PRINTERINDEX

Technically, endorsers differ from printers. Printers are typically used to mark physical sheets so that it's easier to correlate images with physical documents. Endorsers are used to confirm that a given sheet of paper has passed through the scanner, usually with some kind of non-ink stamp.

True endorsers are rare, and have been used interchangeably with printers. TWAIN applications and data sources should treat them as identical.

Data Sources

Deprecate the use of `CAP_ENDORSER` in favor of `CAP_PRINTER`, which offers more options. If there's a history of using `CAP_ENDORSER`, map it to `CAP_PRINTERINDEX`.

Applications

Check for `CAP_PRINTERINDEX`, and use it when it's available. Be prepared to check for `CAP_ENDORSER` with pre-TWAIN 2.3 data sources.

CAP_FEEDERLOADED

Data Sources

`CAP_FEEDERLOADED` reports the current state of the feeder regardless of the setting of any other capability (even if `CAP_FEEDERENABLED` is set to `FALSE`).

Applications

If `CAP_FEEDERLOADED` returns `TWRC_FAILURE` / `TWCC_CAPSEQERROR` on `MSG_GET`, make sure that `CAP_FEEDERENBLED` is set to `TRUE` before trying again. `CAP_FEEDERLOADED` should only be tested in State 4. Use `TW_PENDINGXFERS`. Count returned by `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` to determine if more images are pending for transfer.

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.

ICAP_XFERMECH

Data Sources

Applications are supposed to alert a data source to the transfer mechanism they'll be using in states 6 and 7 by setting `ICAP_XFERMECH`. However, not all applications do this. So, when possible, a data source should tolerate this, and return the image data using whatever `DAT_IMAGE*XFER` call the application selects.

4

Advanced Application Implementation

Chapter Contents

Using TWAIN to acquire a raster image from a device is relatively simple to implement as demonstrated in Chapter 3, "Application Implementation". However, TWAIN also allows application developers to go beyond the simple acquisition of a single image in Native (DIB, TIFF or PICT) format. These more advanced topics are discussed in this chapter.

Capabilities

Capabilities, and the power of an application to negotiate capabilities with the Source, give control to TWAIN-compliant applications. In Chapter 12, "Operating System Dependencies", you will see the negotiation of one capability, `CAP_XFERCOUNT`. This capability is negotiated during State 4 as is always the case unless delayed negotiation is agreed to by both the application and Source. In fact, there is much more to know about capabilities.

Capability Values

Several values are used to define each capability. As seen in Chapter 10, "Capabilities", TWAIN defines a Default Value and a set of Allowed Values for each of the capabilities. The application is not able to modify the Default Value. However, it is able to limit the values offered to a user to a subset of the Allowed Values and to select the capability's Current Value.

Default Value

When a Source is opened, the Current Values for each of its capabilities are set to the TWAIN Default Values listed in Chapter 10, "Capabilities". If no default is defined by TWAIN, the Source will select a value for its default. An application can return a capability to its TWAIN-defined default by issuing a `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_RESET` operation.

Although TWAIN defines defaults for many of the capabilities, a Source may have a different value that it would prefer to use as its default because it would be more efficient. For example, the Source may normally use a 0 bit in a black and white image to indicate white. However, the default for `ICAP_PIXELFLAVOR` is `TWPF_CHOCOLATE` which states that a 0 represents black. Although the TWAIN default is `TWPF_CHOCOLATE`, the Source's preferred default would be `TWPF_VANILLA`. When the application issues a `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GETDEFAULT` operation, the Source returns information about its preferred defaults. The Source and application may be able to negotiate a more efficient transfer based on this information.

Note that this does not imply that the TWAIN defaults should be completely disregarded. When trying to resolve the conflict between the "preferred" value of a particular data source capability and the TWAIN-specified default, it should be considered that the problem is similar to storing and restoring image attributes from session to session. It is reasonable to assume that a data source will want to store the current values for some capabilities to be restored as the current values in a future session. It is then also reasonable to expect that these restored values will be reflected as the current settings for the appropriate capabilities. While storing settings is only really useful for image attributes (the data source would not store the value of `ICAP_PIXELFLAVOR`, but it might store the current `ICAP_RESOLUTION`), it should be stated that preferred values of a data source are to be treated in the same manner.

At the time of loading the data source, all current values for the appropriate capabilities would be set to values that have either been restored from a previous session, or those that are "preferred" by the data source. This current value will remain until it has been explicitly changed by the calling application, or that application issues a `MSG_RESET`.

These are best illustrated using examples, since not all capabilities are suitable for preferred values, and most are not suitable to be stored and restored across multiple scanning sessions.

Example 1:
Scan Parameters are stored in one session and restored in another

1. User configures the data source User Interface with the following parameters: 4x6 inch image in 24-bit at 200 DPI X and Y resolution
2. User selects "Scan" and data source signals application to transfer.
3. Application acquires the image successfully.
4. Application disables the data source.
5. Application inquires during State 4 the current values of Frame, Pixel Type, Bit Depth, and Resolution.
6. Data source reports to each inquiry the current values that were set by the user: 4x6 inch image in 24-bit at 200 DPI X and Y resolution.
7. Application closes the data source.
8. During close procedure, the data source stores the current Frame, Pixel Type, Bit Depth and Resolution.
9. Application opens data source.

10. During open procedure, the data source restores current Frame, Pixel Type, Bit Depth and Resolution.
11. Application inquires during State 4 the current values of Frame, Pixel Type, Bit Depth, and Resolution.
12. Data source reports to each inquiry the current values that were restored from previous session: 4x6 inch image in 24-bit at 200 DPI X and Y resolution in one session.

Example 2:
Data Source represents the preferred Pixel Flavor without compromising TWAIN Defined Default value

1. Application opens data source for the first time
2. Application inquires during State 4 about the Default Pixel Flavor
3. Data source reports that the default pixel flavor is `TWPF_CHOCOLATE.` (See Chapter 10, "Capabilities".)
4. Application inquires during State 4 about the current pixel flavor.
5. Data source reports that the current pixel flavor is `TWPF_VANILLA` (because this device returns data in that gender natively).
6. Application issues reset to current pixel flavor.
7. During reset operation, data source changes current value to `TWPF_CHOCOLATE` and prepares to invert data during transfer to accommodate the calling application request.

There is a condition where this logic falls apart. If the data source wants to return a `TW_ENUMERATION` to a `MSG_GET` request for a constrained capability, there is a chance that the Default value imposed by the TWAIN Specification (Chapter 10, "Capabilities") will not exist within the constrained set of values. In this case, the application should consider the default value to be undefined. Common sense should dictate that the data source provide some default that is reasonable within the currently available set of values for safety (a bad index in a `TW_ENUMERATION` could be a disaster). When the default value is actually used (during `MSG_RESET`) the constraints shall be lifted, and the original default value will once again exist and be defined. (See next section on Constrained Capabilities about `MSG_RESET`) This is only a problem with a `TW_ENUMERATION` container, since it contains an index to the default.

Current Value

The application may request to set the Current Value of a capability. If the Source's user interface is displayed, the Current Value should be reflected (perhaps by highlighting). If the application sets the Current Value, it will be used for the acquire and transfer unless the user or an automatic Source process changes it. The application can determine if changes were made by checking the Current Value during State 6.

To determine just the capability's Current Value, use `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GETCURRENT`. To determine both the Current Value and the Available Values, use the `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GET` operation. For example, you could do a `MSG_GET` on `ICAP_PIXELTYPE` and the Source might return a `TW_ENUMERATION` container containing `TWPT_BW`, `TWPT_GRAY`, and `TWPT_RGB` as Available Values.

To set the Current Value:

Use `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_SET` and one of the following containers:

- `TWON_ONEVALUE`: Place the desired value in `TW_ONEVALUE.Item`.
- `TWON_ARRAY`: Place only the desired items in `TW_ARRAY.ItemList`.

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

It is also possible to set Current Values using the `TW_ENUMERATION` and `TW_RANGE` containers. See the Available Values information for details.

Available Values

To limit the settings the Source can use during the acquire and transfer process, the application may be able to restrict the Available Values. The Source should not use a value outside these values. These restrictions should be reflected in the Source's user interface so unavailable values are not offered to the user.

For example, if the `MSG_GET` operation on `ICAP_PIXELTYPE` indicates the Source supports `TWPT_BW`, `TWPT_GRAY`, and `TWPT_RGB` images and the application only wants black and white images, it can request to limit the Available Values to black and white.

To limit the Available Values:

Use `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_SETCONSTRAINT` and one of the following containers:

- `TWON_ENUMERATION`: Place only the desired values in the `TW_ENUMERATION.ItemList` field. The Current Value can also be set at this time by setting the `CurrentIndex` to point to the desired value in the `ItemList`.
- `TWON_RANGE`: Place only the desired values in the `TW_RANGE` fields. The current value can also be set by setting the `CurrentValue` field.

Note: `TW_ONEVALUE` containers cannot be used to limit the Available Values.

Capability Negotiation

The negotiation process consists of three basic parts:

1. The application determines which capabilities a Source supports
2. The application sets the supported capabilities as desired
3. The application verifies that the settings were accepted by the Source

Negotiation (Part 1)
Application Determines Which Capabilities the Source Supports

Step 1

Application allocates a `TW_CAPABILITY` structure and fills its fields as follows:

- `Cap` = the `CAP_`, `ICAP_` or `ACAP_` name for the capability it is interested in
- `ConType` = `TWON_DONTCARE16`
- `hContainer` = `NULL`

Step 2

Application uses the TW_CAPABILITY structure in a DG_CONTROL / DAT_CAPABILITY / MSG_GET operation.

Step 3

The Source examines the Cap field to see if it supports the capability. If it does, it creates information for the application. In either case, it sets its Return Code appropriately.

Step 4

Application examines the Return Code, and maybe the Condition Code, from the operation.

If TWRC_SUCCESS then the Source does support the capability and

- The ConType field was filled by the Source with a container identifier (TWON_ARRAY, TWON_ENUMERATION, TWON_ONEVALUE, or TWON_RANGE)
- The Source allocated a container structure of ConType and referenced the hContainer field to this structure. It then filled the container with values describing the capability's Current Value, Default Value, and Available Values.

Based on the type of container and its contents (whose type is indicated by its ItemType field), the application can read the values. The application must deallocate the container.

If TWRC_FAILURE and TWCC_CAPUNSUPPORTED

- Source does not support this capability

The application can repeat this process for every capability it wants to learn about. If the application really only wants to get the Current Value for a capability, it can use the MSG_GETCURRENT operation instead. In that case, the ConType will just be TWON_ONEVALUE or TWON_ARRAY but not TWON_RANGE or TWON_ENUMERATION.

Note: The capability, CAP_SUPPORTEDCAPS, returns a list of capabilities that a Source supports. But it doesn't indicate whether the supported capabilities can be negotiated, If the Source does not support the CAP_SUPPORTEDCAPS capabilities, it returns TWRC_FAILURE / TWCC_CAPUNSUPPORTED.

Negotiation (Part 2)
The Application Sets the Supported Capability as Desired

Step 1

Application allocates a TW_CAPABILITY structure and fills its fields as follows:

- Cap = the CAP_, ICAP_, or ACAP_ name for the capability it is interested in
- ConType = TWON_ARRAY, TWON_ENUMERATION, TWON_ONEVALUE or TWON_RANGE (Refer to Chapter 10, "Capabilities" to see each capability and what type(s) of container may be used to set a particular capability.)
- hContainer = The application must allocate a structure of type ConType and reference this field to it. (See the next step.)

Step 2

Application allocates a structure of type ConType and fills it. Based on values received from the Source during the MSG_GET, it can specify the desired Current Value and Available Values that it wants the Source to use. The application should not attempt to set the Source's Default Value, just put an appropriate constant in that field (ex. TWON_DONTCARE32).

Note: The application is responsible for deallocating the container structure when the operation is finished.

Step 3

Send the request to the Source using DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT.

Negotiation (Part 3)
The Application MUST Verify the Result of Their Request

Step 1

Even if a Source supports a particular capability, it is not required to support the setting of that capability. The application must examine the Return Code from the MSG_SET request to see what took place.

If TWRC_SUCCESS then the Source set the capability as requested.

If TWRC_CHECKSTATUS then

- The Source could not use one or more of your exact values. For instance, you asked for a value of 310 but it could only accept 100, 200, 300, or 400. Your request was within its legitimate range so it rounded it to its closest valid setting.

Use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation to determine the current and available settings at this time. This is the only way to determine if the Source's choice was acceptable to your application.

If TWRC_FAILURE / TWCC_BADVALUE then

- Either the Source is not granting your request to set or restrict the value.
- Or, your requested values were not within its range of legitimate values. It may have attempted to set the value to its closest available value.

Use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation to determine the current and available settings at this time. This is the only way to determine if your application can continue without your requested values.

You can repeat the setting and verifying processes for every capability of interest to your application. Remember, your application must deallocate all container structures.

The Most Common Capabilities

TWAIN defines over 150 capabilities. Although the number may seem overwhelming, it is easier to handle if you recognize that some of the capabilities are more commonly used. Here are some of these capabilities:

Basic Capabilities

Units

The ICAP_UNITS capability determines the unit of measure which will be used by the Source. The default is inches but centimeters, pixels, etc. are allowed. This capability's value is used when measuring several other values in capabilities and data structures including:

ICAP_PHYSICALHEIGHT,

ICAP_PHYSICALWIDTH,

ICAP_XNATIVERESOLUTION,

ICAP_YNATIVERESOLUTION,

ICAP_XRESOLUTION,

ICAP_YRESOLUTION,

TW_FRAME,

TW_IMAGEINFO.XResolution,

TW_IMAGEINFO.YResolution

Sense of the Pixel

The ICAP_PIXELFLAVOR specifies how a bit of data should be interpreted when transferred from Source to application. The default is TWPF_CHOCOLATE which means a 0 indicates black (or the darkest color). The alternative, TWPF_VANILLA, means a 0 indicates white (or the lightest color).

Resolution

The image resolution is reported in the TW_IMAGEINFO structure. To inquire or set the Source's resolution, use ICAP_XRESOLUTION and ICAP_YRESOLUTION.

Refer also to ICAP_XNATIVERESOLUTION and ICAP_YNATIVERESOLUTION.

Image Type Capabilities

Types of Pixel

The application should negotiate ICAP_PIXELTYPE and ICAP_BITDEPTH unless it can handle all pixel types at all bit depths. The allowed pixel types are: TWPT_BW, TWPT_GRAY, TWPT_RGB, TWPT_PALETTE, TWPT_CMY, TWPT_CMYK, TWPT_YUV, TWPT_YUVK, TWPT_CIEXYZ, and TWPT_INFRARED.

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.

Parameters for Acquiring the Image

Exposure

Several capabilities can influence this. They include ICAP_BRIGHTNESS, ICAP_CONTRAST, ICAP_SHADOW, ICAP_HIGHLIGHT, ICAP_GAMMA, and ICAP_AUTOBRIGHT.

Scaling

To instruct a Source to scale an image before transfer, refer to ICAP_XSCALING and ICAP_YSCALING.

Rotation

To instruct a Source to rotate the image before transfer, refer to ICAP_ROTATION and ICAP_ORIENTATION.

Constrained Capabilities and Message Responses

There is some confusion about how the data source should respond to various capability queries when the application has imposed constraints upon the supported values. The following guidelines should help clarify the situation.

MSG_RESET

It is known that this call resets the current value of the requested capability to the default. It must also be stated that this call will also reset any application imposed constraints upon the requested capability.

MSG_GETCURRENT, and MSG_GETDEFAULT

It is intuitive to assume that this message should not be supported by capabilities that have no Current or Default value. However, the specification says otherwise in Chapter 10, "Capabilities" (a good example is `CAP_SUPPORTEDCAPS`). In this case, it makes sense to simply respond to these messages in the same manner as `MSG_GET`.

It can also be assumed that it is more intuitive for a data source to respond to this capability with a `TW_ONEVALUE` container in all cases that a `TW_ONEVALUE` container is allowed.

MSG_GET

If an application has constrained the current capability, then the data source response to this message should reflect those constraints. Otherwise, this should respond with all the values that the data source supports. Of course, the number of values that can be placed in the response are restricted by the allowed containers for the particular current capability outlined in Chapter 10, "Capabilities".

MSG_SET (applies if either the application or the driver is TWAIN 2.1 or less)

As indicated in Chapter 7, "Operation Triplets", description of this capability triplet:

> “Current Values are set when the container is a `TW_ONEVALUE` or `TW_ARRAY`. Available and Current Values are set when the container is a `TW_ENUMERATION` or `TW_RANGE`.”

To further clarify this operation, it should be stated that when an application imposes a constraint, the data source must consider the set of supported values and the set of requested constraints. The resulting set of values shall contain only the values that are shared by those supported and those requested.

A condition may arise after constraints are imposed, where the default value is no longer within the set of supported values. When using a `TW_ENUMERATION`, the reported default index should be changed by the data source to something that falls within the new constrained set. This is simply a precaution to ensure it is a valid index. In this case, the Default index in a `TW_ENUMERATION` loses meaning and should be ignored by applications, since `MSG_RESET` shall cause the constraints to be eliminated.

MSG_SET (applies if both the application and the driver is TWAIN 2.2 or more)

When both the application and the driver are TWAIN 2.2 or higher `MSG_SET` only changes the current value, it has no effect on the available values. This applies regardless of the container type used. In other words, `TW_ENUMERATION` and `TW_RANGE` can be used to set the current value using `MSG_SET`. In the case of `TW_ENUMERATION` only the ItemType, CurrentIndex and ItemList fields are used to set the current value. In the case of `TW_RANGE` only the ItemType and CurrentValue fields are used.

MSG_SETCONSTRAINT (applies if both the application and the driver is TWAIN 2.2 or more)

As noted in Chapter 7, "Operation Triplets":

> “Current Values are set when the container is a TW_ONEVALUE or TW_ARRAY. Available and Current Values are set when the container is a TW_ENUMERATION or TW_RANGE.”

To further clarify this operation, it should be stated that when an application imposes a constraint, the data source must consider the set of supported values and the set of requested constraints. The resulting set of values shall contain only the values that are shared by those supported and those requested.

A condition may arise after constraints are imposed, where the default value is no longer within the set of supported values. When using a TW_ENUMERATION, the reported default index should be changed by the data source to something that falls within the new constrained set. This is simply a precaution to ensure it is a valid index. In this case, the Default index in a TW_ENUMERATION loses meaning and should be ignored by applications, since MSG_RESET shall cause the constraints to be eliminated.

Capability Containers in Code Form

Capability information is passed between application and Source by using data structures called containers: TW_ARRAY, TW_ENUMERATION, TW_ONEVALUE, and TW_RANGE. The actions needed to create (pack) and read (unpack) containers are illustrated here in the following code segments. Containers are flexible in that they can be defined to contain one of many types of data. Only one ItemType (TWTY_xxxx) is illustrated per Container (TWON_xxxx) here. Refer to the toolkit disk for complete packing and unpacking utilities that you can use with containers.

Reading (unpacking) a Container from a MSG_GET Operation

```
//---------------------------------------------------
//Example of DG_CONTROL / DAT_CAPABILITY / MSG_GET
//---------------------------------------------------
TW_CAPABILITY   twCapability;
TW_INT16         rc;
//Setup TW_CAPABILITY Structure
   twCapability.Cap = Cap;     //Fill in capability of interest
   twCapability.ConType = TWON_DONTCARE16;
   twCapability.hContainer = NULL;
//Send the Triplet
   rc = (*pDSM_Entry)(&AppID,
                &SourceID,
                DG_CONTROL,
                DAT_CAPABILITY,
                MSG_GET,
               (TW_MEMREF)&twCapability);
```

```
//Check return code
   if (rc == TWRC_SUCCESS)
{
//Switch on Container Type
       switch (twCapability.ConType)
       {
//-----ENUMERATION
          case TWON_ENUMERATION:
          {
          pTW_ENUMERATION   pvalEnum;
          TW_UINT16         valueU16;
          TW_UINT16         index;
          pvalEnum =
(pTW_ENUMERATION)GlobalLock(twCapability.hContainer);
          NumItems = pvalEnum->NumItems;
          CurrentIndex = pvalEnum->CurrentIndex;
          DefaultIndex = pvalEnum->DefaultIndex;
          for (index = 0; index < pvalEnum->NumItems; index++)
          {
              if (pvalEnum->ItemType == TWTY_UINT16)
              {
                  valueU16 = ((TW_UINT16)(pvalEnum->ItemList[index*2]));
                  //Store Item Value
              }
          else if (pvalOneValue->ItemType == TWTY_BOOL)
             {
                valueBool = ((TW_BOOL*)&pvalEnum->ItemList)[index];
                //Store Item Value
             }

          }
          GlobalUnlock(twCapability.hContainer);
          }
          break;
//-----ONEVALUE
          case TWON_ONEVALUE:
```

```
          {
          pTW_ONEVALUE      pvalOneValue;
          TW_BOOL         valueBool;
         pvalOneValue =
(pTW_ONEVALUE)GlobalLock(twCapability.hContainer);
         if (pvalOneValue->ItemType == TWTY_BOOL)
         {
               valueBool = (TW_BOOL)pvalOneValue->Item;
               //Store Item Value
         }
         GlobalUnlock(twCapability.hContainer);
         }
         break;
//-----RANGE
          case TWON_RANGE:
          {
          pTW_RANGE         pvalRange;
          pTW_FIX32         pTWFix32;
          float            valueF32;
          TW_UINT16         index;
            pvalRange = (pTW_RANGE)GlobalLock(twCapability.hContainer);
            if ((TW_UINT16)pvalRange->ItemType == TWTY_FIX32)
            {
               pTWFix32 = &(pvalRange->MinValue);
               valueF32 = FIX32ToFloat(*pTWFix32);
               //Store Item Value
               pTWFix32 = &(pvalRange->MaxValue);
               valueF32 = FIX32ToFloat(*pTWFix32);
               //Store Item Value
               pTWFix32 = &(pvalRange->StepSize);
               valueF32 = FIX32ToFloat(*pTWFix32);
               //Store Item Value
            }
            GlobalUnlock(twCapability.hContainer);
         }
         break;
```

```
//-----ARRAY
          case TWON_ARRAY:
          {
          pTW_ARRAY         pvalArray;
          TW_UINT16         valueU16;
          TW_UINT16         index;
            pvalArray = (pTW_ARRAY)GlobalLock(twCapability.hContainer);
            for (index = 0; index < pvalArray->NumItems; index++)
            {
               if (pvalArray->ItemType == TWTY_UINT16)
               {
                 valueU16 = ((TW_UINT16)(pvalArray->ItemList[index*2]));
                  //Store Item Value
               }
            }
            GlobalUnlock(twCapability.hContainer);
         }
         break;
      }   //End Switch Statement
     GlobalFree(twCapability.hContainer);
   } else {
      //Capability MSG_GET Failed check Condition Code
   }
/**********************************************************
* Fix32ToFloat
* Convert a FIX32 value into a floating point value.
**********************************************************/
float FIX32ToFloat (TW_FIX32    fix32)
{
   float    floater;
   floater = (float)fix32.Whole + (float)fix32.Frac / 65536.0;
   return floater;
}
```

Creating (packing) a Container for a MSG_SET Operation

```
//---------------------------------------------------
//Example of DG_CONTROL / DAT_CAPABILITY / MSG_SET
```

```
//------------------------------------------------
TW_CAPABILITY   twCapability;
TW_INT16         rc;
TW_UINT32         NumberOfItems;
   twCapability.Cap = Cap;      //Insert Capability of Interest
   twCapability.ConType = Container;
      //Use TWON_ONEVALUE or TWON_ARRAY to set current value
      //Use TWON_ENUMERATION or TWON_RANGE to limit available values
   switch (twCapability.ConType)
   {
//-----ENUMERATION
      case TWON_ENUMERATION:
      {
      pTW_ENUMERATION   pvalEnum;
         //The number of Items in the ItemList
         NumberOfItems = 2;

       //Allocate memory for the container and additional ItemList
       // entries
       twCapability.hContainer = GlobalAlloc(GHND,
          (sizeof(TW_ENUMERATION) + sizeof(TW_UINT16) *
(NumberOfItems)));
       pvalEnum = (pTW_ENUMERATION)GlobalLock(twCapability.hContainer);
       pvalEnum->NumItems = 2      //Number of Items in ItemList
       pvalEnum->ItemType = TWTY_UINT16;
       ((TW_UINT16)(pvalEnum->ItemList[0])) = 1;
       ((TW_UINT16)(pvalEnum->ItemList[1])) = 2;
       GlobalUnlock(twCapability.hContainer);
    }
    break;
//-----ONEVALUE
      case TWON_ONEVALUE:
      {
      pTW_ONEVALUE      pvalOneValue;
         twCapability.hContainer = GlobalAlloc(GHND,
sizeof(TW_ONEVALUE));
```

```
         pvalOneValue =
(pTW_ONEVALUE)GlobalLock(twCapability.hContainer);
         (TW_UINT16)pvalOneValue->ItemType = TWTY_UINT16;
         (TW_UINT16)pvalOneValue->Item = 1;
         GlobalUnlock(twCapability.hContainer);
      }
      break;
//-----RANGE
      case TWON_RANGE:
      {
      pTW_RANGE         pvalRange;
      TW_FIX32         TWFix32;
      float            valueF32;
         twCapability.hContainer = GlobalAlloc(GHND, sizeof(TW_RANGE));
         pvalRange = (pTW_RANGE)GlobalLock(twCapability.hContainer);
         (TW_UINT16)pvalRange->ItemType = TWTY_FIX32;
         valueF32 = 100;
         TWFix32 = FloatToFIX32 (valueF32);
         pvalRange->MinValue = *((pTW_INT32) &TWFix32);
         valueF32 = 200;
         TWFix32 = FloatToFIX32 (valueF32);
         pvalRange->MaxValue = *((pTW_INT32) &TWFix32);
         GlobalUnlock(twCapability.hContainer);
      }
      break;
//-----ARRAY
      case TWON_ARRAY:
      {
      pTW_ARRAY         pvalArray;
         //The number of Items in the ItemList
         NumberOfItems = 2;
       //Allocate memory for the container and additional ItemList
entries
       twCapability.hContainer = GlobalAlloc(GHND,
         (sizeof(TW_ARRAY) + sizeof(TW_UINT16) * (NumberOfItems)));
         pvalArray = (pTW_ARRAY)GlobalLock(twCapability.hContainer);
```

```
            (TW_UINT16)pvalArray->ItemType = TWTY_UINT16;
            (TW_UINT16)pvalArray->NumItems = 2;
            ((TW_UINT16)(pvalArray->ItemList[0])) = 1;
            ((TW_UINT16)(pvalArray->ItemList[1])) = 2;
         GlobalUnlock(twCapability.hContainer);
      }
      break;
   }
//-----MSG_SET
   rc = (*pDSM_Entry)(&AppID,
                 &SourceID,
                 DG_CONTROL,
                 DAT_CAPABILITY,
                 MSG_SET,
                 (TW_MEMREF)&twCapability);
   GlobalFree(twCapability.hContainer);
   switch (rc)
   {
      case TWRC_SUCCESS:
         //Capability's Current or Available value was set as specified
      case TWRC_CHECKSTATUS:
        //The Source matched the specified value(s) as closely as
possible
         //Do a MSG_GET to determine the settings made
      case TWRC_FAILURE:
         //Check the Condition Code for more information
   }
/***********************************************************
* FloatToFix32
* Convert a floating point value into a FIX32.
***********************************************************/
TW_FIX32 FloatToFix32 (float floater)
{
   TW_FIX32 Fix32_value;
   TW_INT32 value = (TW_INT32) (floater * 65536.0 + 0.5);
   Fix32_value.Whole = value >> 16;
   Fix32_value.Frac = value & 0x0000ffffL;
```

```
        return (Fix32_value);
    }
```

Delayed Negotiation - Negotiating Capabilities After State 4

Applications may inquire about a Source's capability values at any time during the session with the Source. However, as a rule, applications can only request to **set** a capability during State 4. The rationale behind this restriction is tied to the display of the Source's user interface when the Source is enabled. Many Sources will modify the contents of their user interface in response to some of the application's requested settings. These user interface modifications prevent the user from selecting choices that do not meet the application's requested values. The Source's user interface is never displayed in State 4 so changes can be made without the user's awareness. However, the interface may be displayed in States 5 through 7.

Some capabilities have no impact on the Source's user interface and the application may really want to set them later than State 4. To allow delayed negotiation, the application must request, during State 4, that a particular capability be able to be set later (during States 5, 6 or 7). The Source may agree to this request or deny it. The request is negotiated by the application with the Source by using the `DG_CONTROL / DAT_CAPABILITY` operations on the `CAP_EXTENDEDCAPS` capability.

On the `CAP_EXTENDEDCAPS` capability, the `DG_CONTROL / DAT_CAPABILITY` operations:

`MSG_GET`

Indicates the capabilities the Source is willing to negotiate in States 5, 6 or 7.

`MSG_SET`

Specifies which capabilities the application wishes to negotiate in States 5, 6 or 7. For TWAIN 2.3 or later data sources, this value will already to be set to the values allowed by the data source, the list never starts empty.

`MSG_GETCURRENT`

Provides an array of the capabilities the Source allows to be negotiated in States 5, 6 and 7. For TWAIN 2.3 or later data sources, this value will already to be set to the values allowed by the data source, the list never starts empty.

As with any other capability, if the Source does not support negotiating `CAP_EXTENDEDCAPS`, it will return the Return Code `TWRC_FAILURE` with the Condition Code `TWCC_CAPUNSUPPORTED`.

If an application attempts to set a capability in State 5, 6 or 7 and the Source has not previously agreed to this arrangement, the operation will fail with a Return Code of `TWRC_FAILURE` and a Condition Code of `TWCC_SEQERROR`.

If an application does not use the Source's user interface but presents its own, the application controls the state of the Source explicitly. If the application wants to set the value of any capability, it returns the Source to State 4 and does so. Therefore, an application using its own user interface will probably not need to use `CAP_EXTENDEDCAPS`.

Options for Transferring Data

As discussed previously, there are three modes defined by TWAIN for transferring data:

- Native
- Disk File
- Buffered Memory

A Source is required to support Native and Buffered Memory transfers.

Native Mode Transfer

The use of Native mode, the default mode, for transferring data was covered in Chapter 3, "Application Implementation". There is one potential limitation that can occur in a Native mode transfer. That is, there may not be an adequately large block of RAM available to hold the image. This situation will not be discovered until the transfer is attempted when the application issues the `DG_IMAGE` / `DAT_IMAGENATIVEXFER` / `MSG_GET` operation.

When the lack of memory appears, the Source may respond in one of several ways. It can:

- Simply fail the operation.
- Clip the image to make it fit in the available RAM - The Source should notify the user that the clipping operation is taking place due to limited RAM. The clipping should maintain both the aspect ratio of the selected image and the origin (upper-left).
- Interact with the user to allow them to resize the image or cancel the capture.

The Return Code / Condition Code returned from the `DG_IMAGE` / `DAT_IMAGENATIVEXFER` / `MSG_GET` operation may indicate one of these actions occurred.

If the Return Code is TWRC_XFERDONE:

This indicates the transfer was completed and the session is in State 7. However, it does not guarantee that the Source did not clip the image to make it fit. Even if the application issued a `DG_IMAGE` / `DAT_IMAGEINFO` / `MSG_GET` operation prior to the transfer to determine the image size, it cannot assume that the ImageWidth and ImageLength values returned from that operation really apply to the image that was ultimately transferred. If the dimensions of the image are important to the application, the application should always check the actual transferred image size after the transfer is completed. To do this:

1. Execute a `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation to move the session from State 7 to State 6 (or 5).
2. Determine the actual size of the image that was transferred by reading the header of the actual image data transferred.
   See Chapter 12, "Operating System Dependencies" for more information.

If the Return Code is TWRC_CANCEL:

The acquisition was canceled by the user. The session is in State 7. Execute a `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation to move the session from State 7 to State 6 (or 5).

If the Return Code is TWRC_FAILURE:

Check the Condition Code to determine the cause of the failure. The session is in State 6. No memory was allocated for the DIB, TIFF or PICT. The image is still pending. If lack of memory was the cause, you can try to free additional memory or discard the pending image by executing DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER.

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;
```

The application could use the MSG_GETDEFAULT operation to determine the default file format and filename (TWAIN.TMP or TWAIN.AUD in the current directory). If acceptable, the application could just use that file. However, most applications prefer to set their own values for filename and format. The MSG_SET operation allows this. It is done during State 6. To set your own filename and format, do the following:

1. Allocate the required TW_SETUPFILEXFER structure. Then, fill in the appropriate fields:

   a. FileName – the desired filename. On Windows, be sure to include the complete path name.
   b. Format – the constant for the desired, and supported, format (TWFF_xxxx). If you set it to an unsupported format, the operation returns TWRC_FAILURE / TWCC_BADVALUE and the Source resets itself to write data to the default file.
   c. VRefNum – On Macintosh, write the file's volume reference number. On Windows, fill in the field with a TWON_DONTCARE16.
2. Invoke the DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET as appropriate.

Execute the Transfer into the File

After the application receives the MSG_XFERREADY notice from the Source and has issued the DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET.

Use the following operation: DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET

This operation does not have an associated data structure but just uses NULL for the pData parameter in the DSM_Entry call.

- If the application has not specified a filename (during the setup) - the Source will use either its default file or the last file information it was given.
- If the file specified by the application does not exist - the Source should create it.
- If the file exists but already has data in it - the Source should overwrite the existing data. Notice, if you are transferring multiple files and using the same file name each time, you will overwrite the data unless you copy it to a different filename between transfers.

Note: The application cannot abort a Disk File transfer once initiated. However, the Source's user interface may allow the user to cancel the transfer.

Following execution, be sure to check the Return Code:

TWRC_XFERDONE: File was written successfully. The application needs to invoke the DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER to transition the session back to State 6 (or 5) as was illustrated in Chapter 3, "Application Implementation".

TWRC_CANCEL: The user canceled the transfer. The contents of the file are undefined. Invoke DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER to transition the session back to State 6 (or 5) as was illustrated in Chapter 3, "Application Implementation".

TWRC_FAILURE
The Source remained in State 6.
The contents of the file are undefined.
The image is still pending. To discard it, use DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER.

Check the Condition Code to determine the cause of the failures. The alternatives are:

TWCC_BADDEST = Operation aimed at invalid Source

TWCC_OPERATIONERROR = Either the file existed but could not be accessed or a system error occurred during the writing

TWCC_SEQERROR = Operation invoked in invalid state (i.e. not 6)

Buffered Memory Mode Transfer

Set Capability Values for the Buffered Memory Mode, if Desired

Data is typically transferred in uncompressed format. However, if you are interested in knowing if the Source can transfer compressed data when using the buffered memory mode, perform a DG_CONTROL / DAT_CAPABILITY / MSG_GET on the ICAP_COMPRESSION. The values will include TWCP_NONE (the default) and perhaps others such as TWCP_PACKBITS, TWCP_JPEG ,etc. (See the list in the Constants section of Chapter 8, "Data Types and Data Structures".) More information on compression is available later in this chapter in the section called Transfer of Compressed Data.

Set up the Transfer

During State 4:
Set the ICAP_XFERMECH to TWSX_MEMORY by using the DG_CONTROL / DAT_CAPABILITY / MSG_SET operation.

During States 4, 5, or 6:
The DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation is used by the application to determine what buffer sizes the Source wants to use during the transfer. The Source might have more accurate information in State 6.

The data structure used in the DSM_Entry call is a TW_SETUPMEMXFER structure:

```
typedef struct {
  TW_UINT32   MinBufSize  /* Minimum buffer size in bytes   */
  TW_UINT32   MaxBufSize  /* Maximum buffer size in bytes   */
  TW_UINT32   Preferred   /* Preferred buffer size in bytes */
    } TW_SETUPMEMXFER, FAR *pTW_SETUPMEMXFER;
```

The Source will fill in the appropriate values for its device.

Buffers Used for Uncompressed Strip Transfers

- The application is responsible for allocating and deallocating all memory used during the buffered memory transfer.
- For optimal performance, create buffers of the Preferred size.
- In all cases, the size of the allocated buffers must be within the limits of MinBufSize to MaxBufSize. If outside of these limits, the Source will fail the transfer operation with a Return Code of TWRC_FAILURE / TWCC_BADVALUE.
- If using more than one buffer, all buffers must be the same size.
- Raster lines must be double-word aligned and padded with zeros is recommended .

Execute the Transfer Using Buffers

After the application receives the MSG_XFERREADY notice from the Source and has issued the DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation:

- Allocate one or more buffers of the same size. The best size is the one indicated by the TW_SETUPMEMXFER.Preferred field. If that is impossible, be certain the buffer size is between MinBufSize and MaxBufSize.
- Allocate the TW_IMAGEMEMXFER structure. It will be used in the DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET operation.

The TW_IMAGEMEMXFER structure looks like this:

```
typedef struct {
  TW_UINT16  Compression;
  TW_UINT32  BytesPerRow;
  TW_UINT32  Columns;
  TW_UINT32  Rows;
  TW_UINT32  XOffset;
  TW_UINT32  YOffset;
  TW_UINT32  BytesWritten;
  TW_MEMORY  Memory;
} TW_IMAGEMEMXFER, FAR *pTW_IMAGEMEMXFER;
```

Fill in the TW_IMAGEMEMXFER's first field with TWON_DONTCARE16 and the following six fields with TWON_DONTCARE32.

The TW_MEMORY structure embedded in there looks like this:

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

Fill in the TW_MEMORY structure as follows:

Memory.Flags
Place TWMF_APPOWNS bit-wise ORed with TWMF_POINTER or TWMF_HANDLE

Memory.Length
The size of the buffer in bytes

Memory.TheMem
A handle or pointer to the memory buffer allocated above (depending on which one was specified in the Flags field).

Following each buffer transfer, the Source will have filled in all the fields except Memory which it uses as a reference to the memory block for the data.

The flow of the transfer of buffers is as follows:

Step 1

Buffered Memory transfers provide no embedded header information. Therefore, the application must determine the image attributes. After receiving the MSG_XFERREADY, i.e. while in State 6, the application issues the DG_IMAGE / DAT_IMAGEINFO / MSG_GET and DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET operations to learn about the image's bitmap characteristics and the size and location of the original image on the original page (before scaling or other processing). If additional information is desired, use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation.

Step 2

The application issues DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET.

Step 3

The application checks the Return Code.

- If TWRC_SUCCESS:

  Examine the TW_IMAGEMEMXFER structure for information about the buffer. If you plan to reuse the buffer, copy the data to another location.

  Loop back to Step 2 to get another buffer. Be sure to reinitialize the information in the TW_IMAGEMEMXFER structure (including the Memory fields), if necessary. Issue another DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET operation.

- If TWRC_XFERDONE:

  This is how the Source indicates it just transferred the last buffer successfully. Examine the TW_IMAGEMEMXFER structure for information about the buffer. Perhaps, copy the data to another location, as desired, then go to Step 4.

- If TWRC_CANCEL:

  The user aborted the transfer. The application must send a DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER as described in Chapter 3, "Application Implementation" to move from State 7 to State 6 (or 5).

- If TWRC_FAILURE:

  Examine the Condition Code to determine the cause and handle it.
  If the failure occurred during the transfer of the first buffer, the session is in State 6. If the failure occurred on a subsequent buffer, the session is in State 7.

  The contents of the buffer are invalid and the transfer of the buffer is still pending. To abort it, use DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER.

Step 4

Once the TWRC_XFERDONE has been returned, the application must send the DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER to conclude the transfer. This was described in Chapter 3, "Application Implementation" in the section called State 7 to 6 to 5 - Conclude the Transfer.

Note: The majority of Sources divide the image data into strips when using buffered transfers. A strip is a horizontal band starting at the leftmost side of the image and spanning the entire width but covering just a portion of the image length. The application can verify that strips are being used if the information returned from the Source in the TW_IMAGEMEMXFER structure's XOffset field is zero and the Columns field is equal to the value in the TW_IMAGEINFO structure's ImageWidth field.

An alternative to strips is the use of tiles although they are used by very few Sources. Refer to the TW_IMAGEMEMXFER information in Chapter 8, "Data Types and Data Structures" for an illustration of tiles.

Buffered Memory Mode Transfer With File Format

This operation works very much like Buffered Memory Mode, but the data transferred from the Source to the Application conforms to the image file format specified by a previous call to DG_IMAGE / DAT_SETUPFILEXFER / MSG_GET. There is no requirement for the data to be transferred as complete image lines or for any kind of padding, the data is assumed to be self-contained and self-describing.

The ImageData and Its Layout

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.

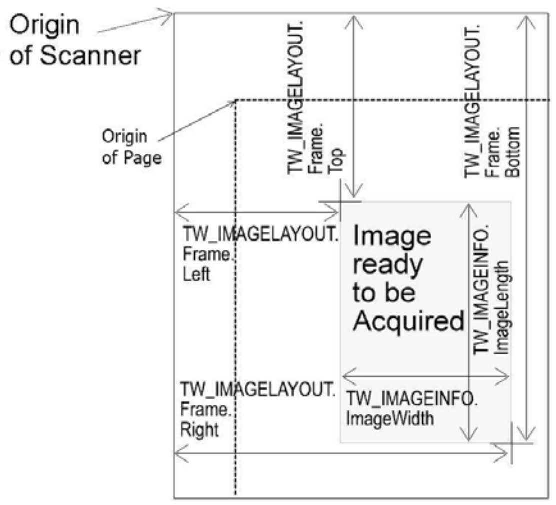

Origin of Scanner
Origin of Page
TW_IMAGELAYOUT. Frame. Top
TW_IMAGELAYOUT. Frame. Bottom
TW_IMAGELAYOUT. Frame. Left
Image ready to be Acquired
TW_IMAGEINFO. ImageLength
TW_IMAGELAYOUT. Frame. Right
TW_IMAGEINFO. ImageWidth

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.

Changing the Image Attributes

Normally, the user will select the desired attributes. However, the application may wish to do this initially during State 4. For example, if the user interface will not be displayed, the application may wish to select the frame. The application can use a DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET operation to define the area (frame) to be acquired. Although, there is no corresponding DG_IMAGE / DAT_IMAGEINFO / MSG_SET operation, the application can change those attributes by setting capabilities and the TW_IMAGELAYOUT data structure.

Here are the relationships:

TW_IMAGEINFO fields	Capability or data structure that impacts the attribute
XResolution	ICAP_XRESOLUTION

TW_IMAGEINFO fields	Capability or data structure that impacts the attribute
YResolution	ICAP_YRESOLUTION
ImageWidth	TW_IMAGELAYOUT.TW_FRAME.Right - TW_FRAME.Left **
ImageLength	TW_IMAGELAYOUT.TW_FRAME.Bottom - TW_FRAME.Top **
SamplesPerPixel	ICAP_PIXELTYPE (i.e. TWPT_BW has 1, TWPT_RGB has 3)
BitsPerSample	Calculated by BitsPerPixel divided by SamplesPerPixel
BitsPerPixel	ICAP_BITDEPTH
Planar	ICAP_PLANARCHUNKY
PixelType	ICAP_PIXELTYPE
Compression	ICAP_COMPRESSION

**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.

Resolving Conflict Between ICAP_FRAMES, ICAP_SUPPORTEDSIZES, DAT_IMAGELAYOUT

Since there are several ways to negotiate the scan area, it becomes confusing when deciding what should take precedence. It is logical to assume that the last method used to set the frame will dictate the current frame. However, it may still be confusing to decide how that is represented during a MSG_GET operation for any of the three methods. The following behavior is suggested.

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

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 effect 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).

Transfer of Multiple Images

Chapter 3, "Application Implementation" discussed the transfer of a single image. Transferring multiple images simply requires looping through the single-image transfer process repeatedly whenever more images are available. Two classes of issues arise when considering multiple image transfer under TWAIN:

- What state transitions are allowable when a session is at an inter-image boundary?
- What facilities are available to support the operation of a document feeder? This includes issues related to high-performance scanning.

This section starts with a review of the single-image transfer process. This is followed by a discussion of options available to an application once the transfer of a single image is complete. Finally, document feeder issues are presented.

To briefly review the single-image transfer process:

- The application enables the Source and the session moves from State 4 to State 5.
- The Source sends the application a `MSG_XFERREADY` when an image is ready for transfer.
- The application uses `DG_IMAGE` / `DAT_IMAGEINFO` / `MSG_GET` and `DG_IMAGE` / `DAT_IMAGELAYOUT` / `MSG_GET` to get information about the image about to be transferred.
- The application initiates the transfer using a `DG_CONTROL` / `DAT_IMAGExxxxFER` / `MSG_GET` operation. The transfer occurs.
- Following a successful transfer, the Source returns `TWRC_XFERDONE`.

- The application sends the `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation to acknowledge the end of the transfer and learn the number of pending transfers.

If the intent behind transferring a single image is to simply flush it from the Source (for example, an application may want to scan only every other page from a stack placed in a scanner with a document feeder), the following operation suffices:

- Issue a `CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` operation. As with normal image transfer, this operation tells the Source that the application has completed acquisition of the current image, and the Source responds by reporting the number of pending transfers.

Preparing for Multiple Image Transfer

The `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` operation issued by the application at the end of every image transfer performs two important functions:

- It returns a count of pending transfers (in `TW_PENDINGXFERS.Count`)
- It transitions the session to State 6 (Transfer Ready) if the count of pending transfers is nonzero, or to State 5 (Source Enabled) if the count is zero. Recall that the count returned is a positive value if the Source knows the number of images available for acquisition. If the Source does not know the number of images available, the count returned is -1. The latter situation can occur if, for example, a document feeder is in use. Note that not knowing the number of images available includes the possibility that no further images are available; see the description of `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` for more on this.

We have just seen that after the `MSG_ENDXFER` operation is issued following an image transfer, the session is either in State 6 or State 5; that is, the session is still very much in an active state. If the session is in State 6 (i.e. "an image is available"), the application takes one of two actions so as to eventually transition the session to State 5 (i.e. "Source is ready to acquire an image, though none is available"):

- It continues to perform the single-image transfer process outlined earlier until no more images are available, or
- It issues a `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_RESET` to flush all pending transfers from the Source.

Once the session is back in State 5, the application has to decide whether to stay in State 5 or transition down to State 4 ("Source is open, and ready for capability negotiation".) Two scenarios are possible here.

In one scenario, the application lets the Source control further state transitions. If the Source sends it a `MSG_XFERREADY`, the application restarts the multiple image transfer loop described above. If the Source sends it a `MSG_CLOSEDSREQ` (e.g. because the user activated the "Done" trigger on the UI displayed by the Source), the application sends back a `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`, thereby putting the session in State 4.

In the other scenario, the application directly controls session state transitions. For example, the application may want to shut down the current session as soon as the current batch of images have been transferred. In this case, the application issues a `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS` as soon as the pending transfers count reaches zero.

It should be noted that there is no "right", "wrong" or "preferred" scenario for an application to follow when deciding what to do once all images in the current set have been transferred. If an application wants to let the user control the termination of a session explicitly, it may well wait for the Source to send it a MSG_CLOSEDSREQ. On the other hand, the application may have a strong sense of what constitutes a session; for example, it may want to terminate a scan session as soon as a blank page is transferred. In such a case, the application will want to control the condition under which the MSG_DISABLEDS is sent.

Use of a Document Feeder

The term document feeder can refer to a physical device's automatic document feeder, such as might be available with a scanner, or to the logical feeding ability of an image database. Both input mechanisms apply although the following text uses the physical feeder for its discussion. The topics covered in this section are:

- Controlling whether to scan pages from the document feeder or the platen
- Detecting whether or not paper is ready for scanning
- Controlling scan lookahead

Note that these concepts are applicable to scanners that do not have feeders; see the discussion below for details.

Selecting the Document Feeder

Sometimes the use of a document feeder actually alters how the image is acquired. For instance, a scanner may move its light bar over a piece of paper if the paper is placed on a platen. When a document feeder is used, however, the same scanner might hold the light bar stable and scan the moving paper. To prepare for such variations the application and Source can explicitly agree to use the document feeder. The negotiation for this action must occur during State 4 **before** the Source is enabled using the following capability.

CAP_FEEDERENABLED

Determine if a Source has a document feeder available and, if so, select that option.

- To determine if this capability is supported, use a DG_CONTROL / DAT_CAPABILITY / MSG_GET operation. TWRC_FAILURE / TWCC_CAPUNSUPPORTED indicates this Source does not have the ability to select the document feeder.
- If supported, use the DG_CONTROL / DAT_CAPABILITY / MSG_SET operation during State 4.
- Set TW_CAPABILITY.Cap to CAP_FEEDERENABLED.
- Create a container of type TW_ONEVALUE and set it to TRUE. Reference TW_CAPABILITY.hContainer to the container.
- Execute the MSG_SET operation and check the Return Code.

If TWRC_SUCCESS then the feeder is available and your request to use it was accepted. The application can now set other document feeder capabilities.

If TWRC_FAILURE and TWCC_CAPUNSUPPORTED, TWCC_CAPBADOPERATION, or TWCC_BADVALUE then this Source does not have a document feeder capability or does not allow it to be selected explicitly.

Note: If an application wanted to prevent the user from using a feeder, the application should use a MSG_SET operation to set the CAP_FEEDERENABLED capability to FALSE.

Detecting Whether an Image is Ready for Acquisition

Having an image ready for acquisition in the Source device is independent of having a selectable document feeder. There are three possibilities here:

- The Source cannot tell whether an image is available,
- An image is available for acquisition, or
- No image is available for acquisition

These cases can be detected by first determining whether a Source can tell that image data is available for acquisition (case 1. vs. cases 2. and 3.) and then determining whether image data is available (case 2. vs. case 3.)The capabilities used to do so are as follows:

CAP_PAPERDETECTABLE

First, determine if the Source can tell that documents are loaded.

- To check if a Source can detect documents, use the `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GET` operation.
- Set the `TW_CAPABILITY.Cap` field to `CAP_PAPERDETECTABLE`.
- The Source returns `TWRC_SUCCESS` with the `hContainer` structure's value set to `TRUE` if it can detect a loaded document that is ready for acquisition. If the result code is `TWRC_FAILURE` with `TWCC_CAPUNSUPPORTED` or `TWCC_BADVALUE`, then the Source cannot detect that paper is loaded.

Note: `CAP_PAPERDETECTABLE` can be used independently of `CAP_FEEDERENABLED`. Also, an automatic document feeder need not be present for a Source to support this capability; e.g. a scanner that can detect paper on its platen should return `TRUE`.

The application cannot set this capability. The Source is simply reporting on a condition.

`CAP_FEEDERLOADED`

Next, determine if there are documents loaded in the feeder.

- To check if pages are present, use the `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_GET` operation.
- Set the `TW_CAPABILITY`.Cap field to `CAP_FEEDERLOADED`.
- The Source returns `TRUE` if there are documents loaded. The information is in the container structure pointed to by the hContainer field of the `TW_CAPABILITY` structure.

Note: Neither `CAP_FEEDERENABLED` nor `CAP_PAPERDETECTABLE` need be `TRUE` to use this capability. A `FALSE` indication from this capability simply indicates that the feeder is not loaded or that the Source's feeder cannot tell. For a definitive answer, be sure to check `CAP_PAPERDETECTABLE`.

Controlling Scan Lookahead

With low-end scanners there is usually ample time for the CPU handling the image acquisition to process incoming image data on-the-fly or in the scan delay between pages. However, with higher performance scanners the CPU image processing time for a given page can become a significant fraction of the scan time. This problem can be alleviated if the scanner can scan ahead image data that the CPU has yet to acquire. This data can be buffered in scanner-local memory, or stored in main memory by the Source via a DMA operation while the CPU processes the current image.

Scan look-ahead is not always desirable, however. This is because the decision to continue a scan may be determined by the results of previously scanned images. For example, a scanning application may decide to stop a scan whenever it sees a blank page. If scan look-ahead were always enabled, one or more pages past the blank page may be scanned and transferred to the scanner's output bin. Such behavior may be incorrect from the point of view of the application's design.

We have argued that the ability to control scan look-ahead is highly desirable. However, a single "enable scan look-ahead" command is insufficient to capture the richness of function provided by some scanners. In particular, TWAIN's model of document feeding has each image (e.g., sheet of paper) transition through a three stage process.

1. **Image is in input bin.** This action is taken by the user (for example, by placing a stack of paper into an auto-feeder.)
2. **Image is ready for scan.** This action causes the next available image to be placed at the start of the scan area. Set the `CAP_AUTOFEED` capability(described below)to automatically feed images to the start of the scan area.
3. **Image is scanned.** This action actually causes the image to be scanned. For example, the `DG_IMAGE/DAT_IMAGEMEMXFER/MSG_GET` operation initiates image transfer to an application via buffered memory. TWAIN allows a Source to pre-fetch images into Source-local memory (even before the application requests them) by setting the `CAP_AUTOSCAN` capability.

CAP_AUTOFEED

Enable the Source's automatic document feeding process.

- Use `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_SET`.
- Set the `TW_CAPABILITY.Cap` field to `CAP_AUTOFEED` and set the capability to `TRUE`.
- When set to `TRUE`, the behavior of the Source is to eject one page and feed the next page after all frames on the first page are acquired. This automatic feeding process will continue whenever there is image data ready for acquisition (and the Source is in an enabled state). `CAP_FEEDERLOADED` is `TRUE` showing that pages are in the document feeder.

Note: `CAP_FEEDERENABLED` must be set to `TRUE` to use this capability. If not, the Source should return `TWRC_FAILURE` / `TWCC_CAPUNSUPPORTED`.

CAP_AUTOSCAN

Enable the Source's automatic document scanning process.

- Use `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_SET`.
- Set the `TW_CAPABILITY.Cap` field to `CAP_AUTOSCAN` and set the capability to `TRUE`.
- When set to `TRUE`, the behavior of the Source is to eject one page and scan the next page after all frames on the first page are acquired. This automatic scanning process will

continue whenever there is image data ready for acquisition (and the Source is in an enabled state.

Note: Setting CAP_AUTOSCAN to TRUE implicitly sets CAP_AUTOFEED to TRUE also.

When your application uses automatic document feeding:

- Set CAP_XFERCOUNT to -1 indicating your application can accept multiple images.
- Expect the Source to return the TW_PENDINGXFERS.Count as -1. It indicates the Source has more images to transfer but it is not sure how many.
- Using automatic document feeding does not change the process of transferring multiple documents described earlier and in Chapter 3, "Application Implementation".

Control of the Document Feeding by the Application

In addition to automatic document feeding, TWAIN provides an option for an application to manually control the feeding of documents. This is only possible if the Source agrees to negotiate the following capabilities during States 5, 6 and 7, as indicated by CAP_EXTENDEDCAPS. If CAP_AUTOFEED is set to TRUE, it can impact the way the Source responds to the following capabilities as indicated below.

CAP_FEEDPAGE

- If the application sets this capability to TRUE, the Source will eject the current page (if any) and feed the next page.
- To work as described requires that CAP_FEEDERENABLED and CAP_FEEDERLOADED be TRUE.
- If CAP_AUTOFEED is TRUE, the action is the still the same.
- The page ejected corresponds to the image that the application is acquiring (or is about to acquire). Therefore, if CAP_AUTOSCAN is TRUE and one or more pages have been scanned speculatively, the page ejected may correspond to a page that has already been scanned into Source-local buffers.

CAP_CLEARPAGE

- If the application sets this capability to TRUE, the Source will eject the current page and leave the feeder acquire area empty (that is, with no image ready to acquire).
- To work as described, this requires that CAP_FEEDERENABLED be TRUE and there be a paper in the feeder acquire area to begin with.
- If CAP_AUTOFEED is TRUE, the next page will advance to the acquire area.
- If CAP_AUTOSCAN is TRUE, setting this capability returns TWRC_FAILURE with TWCC_BADVALUE.

CAP_REWINDPAGE

- If the application sets this capability to TRUE, the Source will return the current page to the input area and return the last page from the output area into the acquisition area.
- To work as described requires that CAP_FEEDERENABLED be TRUE.
- If CAP_AUTOFEED is TRUE, the normal automatic feeding will continue after all frames of this page are acquired.

- The page rewound corresponds to the image that the application is acquiring. Therefore, if CAP_AUTOSCAN is TRUE and one or more pages have been scanned speculatively, the page rewound may correspond to a page that has already been scanned into Source-local buffers.

Transfer of Compressed Data

When using the Buffered Memory mode for transferring images, some Sources may support the transfer of data in a compressed format.

To determine if a Source supports transfer of compressed data and to set the capability

1. Use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation.
2. Set the TW_CAPABILITY.Cap field to ICAP_COMPRESSION.
3. The Source returns information about the compression schemes they support in the container structure pointed to by the hContainer field of TW_CAPABILITY. The identifiers for the compression alternatives all begin with TWCP_, such as TWCP_PACKBITS, and can be seen in the Constants section of Chapter 8, "Data Types and Data Structures" and in the TWAIN.H file.
4. If you wish to negotiate for the transfer to use one of the compression schemes shown, use the DG_CONTROL / DAT_CAPABILITY / MSG_SET operation.

The TW_IMAGEMEMXFER structure is used with the DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET operation. The structure looks like this:

```
typedef struct {
  TW_UINT16   Compression; /* A TWCP_xxxx constant */
  TW_UINT32   BytesPerRow;
  TW_UINT32   Columns;
  TW_UINT32   Rows;
  TW_UINT32   XOffset;
  TW_UINT32   YOffset;
  TW_UINT32   BytesWritten;
  TW_MEMORY   Memory;
} TW_IMAGEMEMXFER, FAR *pTW_IMAGEMEMXFER;
```

When compressed strips of data are transferred:

- The BytesPerRow field will be set to 0. The Columns, Rows, XOffset, and YOffset fields will contain TWON_DONTCARE32 indicating the fields hold invalid values. (The original image height and width are available by using the DG_IMAGE / DAT_IMAGEINFO / MSG_GET operation during State 6 prior to the transfer.)
- Transfer buffers are always completely filled by the Source. For compressed data, it is very likely that at least one partial line will be written into the buffer.
- The application is responsible for deallocating the buffers.

When compressed, tiled data are transferred:

- All fields in the structure contain valid data. `BytesPerRow`, `Columns`, `Rows`, `XOffset`, and `YOffset` all describe the uncompressed tile. `Compression` and `BytesWritten` describe the compressed tile.
- In this case, unlike with compressed, strip data transfer, the Source allocates the transfer buffers. This allows the Source to create buffers of differing sizes so that complete, compressed tiles can be transferred to the application intact (not split between sequential buffers). Under these conditions, the application should set the fields of the `TW_MEMORY` structure so Flags is `TWMF_DSOWNS`, Length is `TWON_DONTCARE32` and TheMem is `NULL`. The Source must assume that the application will keep the previous buffer rather than releasing it. Therefore, the Source must allocate a new buffer for each transfer.
- The application is responsible for deallocating the buffers.
- Finally, the application cannot assume that the tiles will be transferred in any particular, logical order.

JPEG Compression

TWAIN supports transfer of several forms of compressed data. JPEG compression is one of them. The JPEG compression algorithm provides compression ratios in the range of 10:1 to 25:1 for grayscale and full-color images, often without causing visible loss of image quality. This compression, which is created by the application of a series of “perceptual” filters, is achieved in three stages:

Color Space Transformation and Component Subsampling (Color Images Only, Not for Grayscale)

The human eye is far more sensitive to light intensity (luminance) than it is to light frequency (chrominance, or “color”) since it has, on average, 100 million detectors for brightness (the “rods”) but only about 6 million detectors for color (the “cones”). Substantial image compression can be achieved simply by converting a color image into a more efficient luminance/chrominance color space and then subsampling the chrominance components.

This conversion is provided for by the `TW_JPEGCOMPRESSION` structure. By specifying the `TW_JPEGCOMPRESSION.ColorSpace = TWPT_YUV`, Source RGB data is converted into more space-efficient YUV data (better known as CCIR 601-1 or YCbCr). `TW_JPEGCOMPRESSION.SubSampling` specifies the ratio of luminance to chrominance samples in the resulting YUV data stream, and a typical choice calls for two luminance samples for every chrominance sample. This type of subsampling is specified by entering `0x21102110` into the `TW_JPEGCOMPRESSION.SubSampling` field. A larger ratio of four luminance samples for every chrominance sample is represented by `0x41104110`. To sample two luminance values for every chrominance sample in both the horizontal and vertical axes, use a value of `0x21102110`.

Application of the Discrete Cosine Transform (DCT) and Quantization

The original components (with or without color space conversion) are next mathematically converted into a spatial frequency representation using the DCT and then filtered with quantization matrices (each frequency component is divided by its corresponding member in a quantization matrix). The quantization matrices are specified by `TW_JPEGCOMPRESSION.QuantTable[]` and up to four quantization matrices may be defined for up to four different original components. `TW_JPEGCOMPRESSION.QuantMap[]` maps the particular original component to its respective quantization matrix.

Note: Suggested defaults for the quantization map and tables are in Section K of the JPEG Draft International Standard, version 10918-1. These defaults are used in the tables for QuantTable, HuffmanDC, and HuffmanAC by TWAIN. The default tables are selected by putting `NULL` into each of the `TW_JPEGCOMPRESSION.QuantTable[]` entries.

Huffman encoding

The resulting coefficients from the DCT and quantization steps are further compressed in one final stage using a loss-less compression algorithm called Huffman encoding. Application developers can provide Huffman tables, though typically the default tables—selected by writing `NULL` into `TW_JPEGCOMPRESSION.HuffmanDC[]` and `TW_JPEGCOMPRESSION.HuffmanAC[]`— yield very good results.

The algorithm optionally supports the use of restart marker codes. The purpose of these markers is to allow random access to strips of compressed data in a JPEG data stream. They are more fully described in the JPEG specification.

See Chapter 8, "Data Types and Data Structures" for the definition of the `TW_JPEGCOMPRESSION` data structure. Example data structures are shown below for RGB image compression and grayscale image compression:

```
/* RGB image compression - YUV conversion and 2:1:1 chrominance */
/* subsampling                                                   */
typedef struct TW_JPEGCOMPRESSION  myJPEG;
myJPEG.ColorSpace      = TWPT_YUV;       // convert RGB to YUV
myJPEG.SubSampling     = 0x21102110;      // 2 Y for each U, V
myJPEG.NumComponents    = 3;             // Y, U, V
myJPEG.RestartFrequency = 0;             // No restart markers
myJPEG.QuantMap[0]     = 0;             // Y component uses table0
myJPEG.QuantMap[1]     = 1;             // U component uses table 1
myJPEG.QuantMap[2]     = 1;             // V component uses table 1
myJPEG.QuantTable[0]    = NULL;          // select defaults for quant
                                         // tables
myJPEG.QuantTable[1]    = NULL;          //
myJPEG.QuantTable[2]    = NULL;          //
myJPEG.HuffmanMap[0]    = 0;             // Y component uses DC & AC
                                   // table 0
myJPEG.HuffmanMap[1]    = 1;             // U component uses DC & AC
                                   // table 1
myJPEG.HuffmanMap[2]    = 1;             // V component uses DC & AC
                                   // table 1
myJPEG.HuffmanDC[0]     = NULL;          // select default for Huffman
                                   // tables
myJPEG.HuffmanDC[1]     = NULL;          //
myJPEG.HuffmanAC[0]     = NULL;          //
myJPEG.HuffmanAC[1]     = NULL;          //
/* Grayscale image compression - no color space conversion or  */
/* subsampling                                                 */
typedef struct TW_JPEGCOMPRESSION  myJPEG;
myJPEG.ColorSpace      = TWPT_GRAY;      // Grayscale data
myJPEG.SubSampling     = 0x10001000;      // no chrominance components
myJPEG.NumComponents    = 1;             // Grayscale
myJPEG.RestartFrequency = 0;             // No restart markers
myJPEG.QuantMap[0]      = 0;            // select default for quant
                                        // map
myJPEG.QuantTable[0]    = NULL;         //
myJPEG.HuffmanMap[0]    = 0;            // select default for Huffman
                                        // tables
myJPEG.HuffmanDC[0]     = NULL;         //
myJPEG.HuffmanAC[0]     = NULL;         //
```

The resulting compressed images from these examples will be compatible with the JPEG File Interchange Format (JFIF version 1.1) and will therefore be usable by a variety of applications that are JFIF-aware.

Alternative User Interfaces

Alternatives to Using the Source Manager's Select Source Dialog

TWAIN ships its Source Manager code to act as the communication vehicle between application and Source. One of the services the Source Manager provides is locating all available Sources that meet the application's requirements and presenting those to the user for selection.

It is recommended that the application use this approach. However, the application is not required to use this service. Two alternatives exist:

- The application can develop and present its own custom selection interface to the user. This is presented in response to the user choosing **Select Source...** from its menu.
- Or, if the application is dedicated to control of a specific Source, the application can transparently select the Source. In this case, the application does not functionally need to have a Select Source... option in the menu but a grayed-out one should be displayed for consistency with all other TWAIN-compliant applications.

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 `MSG_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:

If the application wants to open the system default Source , use the DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT operation to have the Source Manager locate the default Source and fill the TW_IDENTITY structure with information about it. The name of the Source is contained in the TW_IDENTITY.ProductName field. Save the TW_IDENTITY structure.

OR

If you know the ProductName of the Source you wish to use and it is not the system default Source, use the DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST and DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT operations to have the Source Manager locate each Source. You must continue looking at Sources until you verify that the desired Source is available. Save the TW_IDENTITY structure when you locate the Source you want. If the Return Code TWRC_ENDOFLIST appears before the desired Source is located, it is not available.

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, rather than MSG_USERSELECT, the Source Manager does not update the system default Source information to reflect your choice.)

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.

Modal Versus Modeless User Interfaces

The Source Manager's user interface is a modal interface but the Source may provide a modeless or modal interface. Here are the differences:

Modeless

When a Source uses a modeless user interface, although the Source's interface is displayed, the user is still able to access the application by clicking on the application's window and making it active.

The user is expected to click on a Close button on the Source's user interface when they are ready for that display to go away. The application must NOT automatically close a modeless Source after the first (or any subsequent) transfer, even if the application is only interested in receiving a single transfer. If the application closes the Source before the user requests it, the user is likely to become confused about why the window disappeared. Wait until the user indicates the desire to close the Source's window and the Source sends this request (MSG_CLOSEDSREQ) to the application before closing the Source.

Modal

A Source using a modal user interface prevents the user from accessing other windows.

For Windows only, if the interface is application modal, the user cannot access other applications but can still access system utilities. If the interface is system modal (which is rare), the user cannot access anything else at an application or system level. A system modal dialog might be used to display a serious error message, like a UAE (Unrecoverable Application Error).

If using a modal interface, the Source can perform only one acquire during a session although there may be multiple frames per acquisition. The Source will send a close request to the application following the completion of the data transfer. Again, the application waits to receive this request.

The Source indicates if it is using a modeless or modal interface after the application enables it using the `DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS` operation. The data structure used in the operation (`TW_USERINTERFACE`) contains a field, `ShowUI`, which is set by the application to indicate whether the Source should display its user interface. If the application requests the user interface be shown, it may also set the `ModalUI` field to indicate if it wishes the Source's GUI to run modal (`TRUE`) or modeless (`FALSE`).

When requested by the Source, the application uses the `DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_DISABLEDS` operation to remove the Source's user interface.

Grayscale and Color Information for an Image

There are operation triplets in TWAIN that allow the application developer to interact with and influence the grayscale or color aspect of the images that a Source transfers to the application. The following operations provide these abilities:

- CIE Color Descriptors

  `DG_IMAGE` / `DAT_CIECOLOR` / `MSG_GET`
- Grayscale Changes

  `DG_IMAGE` / `DAT_GRAYRESPONSE` / `MSG_RESET`

  `DG_IMAGE` / `DAT_GRAYRESPONSE` / `MSG_SET`
- Palette Color Data

  `DG_IMAGE` / `DAT_PALETTE8` / `MSG_GET`

  `DG_IMAGE` / `DAT_PALETTE8` / `MSG_GETDEFAULT`

  `DG_IMAGE` / `DAT_PALETTE8` / `MSG_RESET`

  `DG_IMAGE` / `DAT_PALETTE8` / `MSG_SET`
- RGB Response Curve Data

  `DG_IMAGE` / `DAT_RGBRESPONSE` / `MSG_RESET`

  `DG_IMAGE` / `DAT_RGBRESPONSE` / `MSG_RESET`

CIE Color Descriptors

The CIE XYZ approach is a method for storing color data which simplifies doing mathematical manipulations on the data. Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

If your application wishes to receive the image data in this format:

1. You must ensure that the Source is able to provide data in CIE XYZ format. To check this, use the DG_CONTROL / DAT_CAPABILITY / MSG_GET operation and get information on the ICAP_PIXELTYPE. If TWPT_CIEXYZ is returned as one of the supported types, the Source can provide data in CIE XYZ format.
2. After verifying that the Source supports it, the application can specify that data transfers should use the CIE XYZ format by invoking a DG_CONTROL / DAT_CAPABILITY / MSG_SET operation on the ICAP_PIXELTYPE. Use a TW_ONEVALUE container whose value is TWPT_CIEXYZ.

To determine the parameters that were used by the Source in converting the color data into the CIE XYZ format, use the DG_IMAGE / DAT_CIECOLOR / MSG_GET operation following the transfer of the image.

Grayscale Changes

(The grayscale operations assume that the application has instructed the Source to provide grayscale data by setting its ICAP_PIXELTYPE to TWPT_GRAY and the Source is capable of this.)

The application can request that the Source apply a transfer curve to its grayscale data prior to transferring the data to the application. To do this, the application uses the DG_IMAGE / DAT_GRAYRESPONSE / MSG_SET operation. The desired transfer curve information is placed by the application within the TW_GRAYRESPONSE structure (the actual array is of type TW_ELEMENT8). The application must be certain to check the Return Code following this request. If the Return Code is TWRC_FAILURE and the Condition Code shows TWCC_BADPROTOCOL, this indicates the Source does not support grayscale response curves (despite supporting grayscale data).

If the Source allows the application to set the grayscale transfer curve, there must be a way to reset the curve to its original non-altered value. Therefore, the Source must have an “identity response curve” which does not alter grayscale data but transfers it exactly as acquired. When the application sends the DG_IMAGE / DAT_GRAYRESPONSE / MSG_RESET operation, the Source resets the grayscale response curve to its identity response curve.

Palette Color Data

(The palette8 operations assume that the application has instructed the Source to use the TWPT_PALETTE type for its ICAP_PIXELTYPE and that the Source has accepted this.)

The DAT_PALETTE8 operations allow the application to inquire about a Source’s support for palette color data and to set up a palette color transfer. The operations are specialized for 8-bit data, whether grayscale or color (8-bit or 24-bit). The MSG_GET operation allows the application to learn what palette was used by the Source during the image acquisition. The application should always execute this operation immediately after an image transfer rather than before because the Source may optimize the palette during the acquisition process. Some Sources may allow an application to define the palette to be used during image acquisition via the MSG_SET operation. Be sure to check the Return Code to verify that it is TWRC_SUCCESS following a MSG_SET operation. That is the only way to be certain that your requested palette will actually be used during subsequent palette transfers.

RGB Response Curve Data

(The RGB Response curve operations assume that the application has instructed the Source to provide RGB data by setting its `ICAP_PIXELTYPE` to `TWPT_RGB` and the Source is capable of this.)

The application can request that the Source apply a transfer curve to its RGB data prior to transferring the data to the application. To do this, the application uses the `DG_IMAGE` / `DAT_RGBRESPONSE` / `MSG_SET` operation. The desired transfer curve information is placed by the application within the `TW_RGBRESPONSE` structure (the actual array is of type `TW_ELEMENT8`). The application must be certain to check the Return Code following this request. If the Return Code is `TWRC_FAILURE` and the Condition Code shows `TWCC_BADPROTOCOL`, this indicates the Source does not support RGB response curves (despite supporting RGB data).

If the Source allows the application to set the RGB response curve, there must be a way to reset the curve to its original non-altered value. Therefore, the Source must have an "identity response curve" which does not alter RGB data but transfers it exactly as acquired. When the application sends the `DG_IMAGE / DAT_RGBRESPONSE / MSG_RESET` operation, the Source resets the RGB response curve to its identity response curve.

5

Source Implementation

Chapter Contents

Companies that produce image-acquisition devices, and wish to gain the benefits of being TWAIN-compliant, must develop TWAIN-compliant Source software to drive their device. The Source software is the interface between TWAIN's Source Manager and the company's physical (or logical) device. To write effective Source software, the developer must be familiar with the application's expectations as discussed in the other chapters of this document.

The Structure of a Source

The following sections describe the structure of a source. Also see Chapter 12, "Operating System Dependencies".

Implementation

The Source is implemented as a Shared Library (DLL on Windows). The Source will not run stand-alone.

Naming and Location

TWAIN data sources' file name must end with a .DS extension. The Source Manager recursively searches for all Sources in the TWAIN sub-directory. To reduce the chance for naming collisions, each Source should create a sub-directory beneath TWAIN, giving it a name relevant to their product.

Entry Points

- Every Source is required to have a single entry point called `DS_Entry` (see Chapter 6, "Entry Points and Triplet Components"). The source should be able to quickly respond to the `DG_CONTROL / DAT_IDENTITY / MSG_GET` operation.

Resources

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

Operation Triplets

In Chapter 3, "Application Implementation", we introduced all of the triplets that an application can send to the Source Manager or ultimately to a Source. There are several other triplet operations which do not originate from the application. Instead, they originate from the Source Manager or Source and are introduced in this chapter. All defined operation triplets are listed in detail in Chapter 7, "Operation Triplets".

Triplets from the Source Manager to the Source

There are three operation triplets that are originated by the Source Manager. They are:

DG_CONTROL / DAT_IDENTITY

`MSG_GET`	Returns the Source's identity structure
`MSG_OPENDS`	Opens and initializes the Source
`MSG_CLOSEDS`	Closes and unloads the Source

The `DG_CONTROL / DAT_IDENTITY / MSG_GET` operation is used by the Source Manager to identify available Sources. It may send this operation to the Source **at any time** and the Source **must be prepared** to respond informatively to it. That means, the Source must be able to return its identity structure before being opened by the Source Manager (with the `MSG_OPENDS` command). The Source's initially loaded code segment must be able to perform this function without loading any additional code segments. This allows quick identification of all available Sources and is the only operation a Source must support before it is formally opened.

The `TW_IDENTITY` structure looks like this:

```
typedef struct {
      TW_UINT32     Id;
      TW_VERSION    Version;
```

```
        TW_UINT16     ProtocolMajor;
        TW_UINT16     ProtocolMinor;
        TW_UINT32     SupportedGroups;
        TW_STR32      Manufacturer;
        TW_STR32      ProductFamily;
        TW_STR32      ProductName;
} TW_IDENTITY, FAR *pTW_IDENTITY;
```

The `ProductName` field in the Source's `TW_IDENTITY` structure should uniquely identify the Source. This string will be placed in the Source Manager's Select Source... dialog for the user. (The file name of the Source should also approximate the `ProductName`, if possible, to add clarity for the user at installation time.) Fill in all fields except the Id field which will be assigned by the Source Manager. The unique Id number that identifies your Source during its current session will be passed to your Source when it is opened by the `MSG_OPENDS` operation. Sources on Windows must save this `TW_IDENTITY.Id` information for use when sending notifications from the Source to the application.

Sources and the Event Loop

Handling Events

See Chapter 12, "Operating System Dependencies" on how to implement the Event Loop.

Communicating to the Application

As explained in Chapter 3, "Application Implementation", there are four instances where the Source must originate and transmit a notice to the application:

- **When it has data ready to transfer** (`MSG_XFERREADY`)

  The Source must send this message when the user clicks the "GO" button on the Source's user interface or when the application sends a `DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS` operation with `ShowUI = FALSE`. The Source will transition from State 5 to State 6. The application should now perform their inquiries regarding `TW_IMAGEINFO` and capabilities. Then, the application issues a `DG_IMAGE` / `DAT_IMAGExxxxXFER` / `MSG_GET` operation to begin the transfer process. Typically, though it is not required, it is at this time that a flatbed scanner (for example) will begin *simultaneously* to acquire and transfer the data using the specified transfer mode.

- **When it needs to have its user interface disabled** (`MSG_CLOSEDSREQ`)

  Typically, the Source will send this only when the user clicks on the **CLOSE** button on the Source's user interface or when an error occurs which is serious enough to require terminating the session with the application. The Source should be in (or transition to) State 5. The application should respond by sending a `DG_CONTROL` / `DAT_USERINTERFACE /` `MSG_DISABLEDS` operation to transition the session back to State 4.

- **When the user has pressed the OK button** (`MSG_CLOSEDSOK`)

  When the user has pressed the OK button in a Source's dialog that was brought up with `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY`.

  Applications should use this event as the indicator that the user has set all the desired attributes from the Source's GUI.

- **When the Source needs to report a Device Event.**

  Note that the application must explicitly request the Source to supply Device Events (`MSG_DEVICEEVENT`). Sources must only provide those Device Events requested by a Source through the `CAP_DEVICEEVENT` capability. The default for this capability when the Source starts up is an empty `TW_ARRAY`, indicating that no Device Events are being reported. Applications that turn on Device Events must issue a `DG_CONTROL / DAT_DEVICEEVENT / MSG_GET` command as soon as possible after receiving a Device Event.

  The Source creates a call to `DSM_Entry` (the entry point in the Source Manager) and fills the destination with the `TW_IDENTITY` structure of the application. The Source uses one of the following triplets:

  ```
  DG_CONTROL / DAT_NULL / MSG_XFERREADY
  DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ
  DG_CONTROL / DAT_NULL / MSG_CLOSEDSOK
  ```

  The Source Manager recognizes the notice and makes sure the message is received correctly by the application.

On Macintosh legacy 1.x sources refer to the TWAIN 1.9 Specification.

User Interface Guidelines

Each TWAIN-compliant Source provides a user interface to assist the user in acquiring data from their device. Although each device has its own unique needs, the following guidelines are provided to increase consistency among TWAIN-compliant devices.

Displaying the User Interface

The application issues `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` to transition the session from State 4 to 5.

The `TW_USERINTERFACE` data structure contains these fields:

- **`ShowUI`** - If set to `TRUE`, the Source displays its user interface.
  If `FALSE`, the application will be providing its own.
- **`hParent`** - Used by Sources on Windows only. It indicates the application's window handle. This is to be designated as the Source's parent for the dialog so it is a proper child of its parent application.
- **`ModalUI`** - To be set by the Application to `TRUE` or `FALSE`.

Sources are not required to allow themselves to be enabled without showing their user interface (`ShowUI` = `FALSE`) but it is strongly recommended that they allow this. If your Source cannot be used without its user interface, it should enable showing the user interface (just as if `ShowUI = TRUE`) and return `TWRC_CHECKSTATUS`. All Sources, however, must report whether or not they honor `ShowUI` set to `FALSE` via the `CAP_UICONTROLLABLE` capability. This allows applications to know whether the Source-supplied user interface can be suppressed before it is displayed.

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.

When the user interface is disabled (by `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`), a pointer to a `TW_USERINTERFACE` is included in the `pData` parameter.

Modal versus Modeless Interfaces

As stated in Chapter 4, "Advanced Application Implementation", the Source's user interface may be modal or modeless. The modeless approach gives the user more control and is recommended whenever practical. Refer to Chapter 12, "Operating System Dependencies" about implementation.

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.

Impact of Capability Negotiation

If the Source has agreed to limit the Available Values and/or to set the Current Value, the interface should reflect the negotiation. However, if a capability has not been negotiated by the application, the interface should not be modified (don't gray out a control because it wasn't negotiated.)

Advanced Topics

If a Source can acquire from more than one device, the Source should allow the user to choose which device they wish to acquire from. Provide the user with a selection dialog that is similar to the one presented by the Source Manager's Select Source... dialog.

Capability Negotiation

Capability negotiation is a critical area for a Source because it allows the application to understand and influence the images that it receives from your Source.

Inquiries From the Application

While the Source is open but not yet enabled (from DG_CONTROL / DAT_IDENTITY / MSG_OPENDS until DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS), the application can inquire the values of all supported capabilities, and request to set those values.

Once the Source is enabled, the application may only **inquire** about capabilities. An attempt to set a capability fails with TWRC_FAILURE / TWCC_SEQERROR, unless allowed by the CAP_EXTENDEDCAPS capability.

Responding to Inquiries

Sources must be able to respond to capability inquiries with current values at any time the Source is open (i.e. from MSG_OPENDS until MSG_CLOSEDS or before posting a MSG_CLOSEDSREQ).

A Source should respond with information to any capability that applies to your device. Only if a capability has no match with your device's features should you respond with TWRC_FAILURE / TWCC_CAPUNSUPPORTED.

Refer to Chapter 10, "Capabilities" for the complete list of TWAIN-defined capabilities.

Responding to Requests to Set Capabilities

If the requested value is invalid or the Source does not support the capability, then return TWRC_FAILURE / TWCC_CAPUNSUPPORTED. If the requested operation (MSG_SET, MSG_RESET, etc.) is not supported, then return TWRC_FAILURE / TWCC_CAPBADOPERATION. If the capability is unavailable because of a dependency on another capability (i.e., ICAP_CCITTKFACTOR is not available unless ICAP_COMPRESSION is TWCP_GROUP32D), then return TWCC_CAPSEQERROR. Returning these condition codes makes it possible for an application using its own UI to intelligently make dependent capabilities available or unavailable for user access.

If the request was fulfilled, return TWRC_SUCCESS.

If the requested value is close to an acceptable value but doesn't match exactly, set it as closely as possible and then return TWRC_CHECKSTATUS.

A Source supports MSG_SET operations using the same containers it returns through MSG_GET, MSG_GETCURRENT and MSG_GETDEFAULT operations.

- Example #1, a call to DG_CONTROL / DAT_CAPABILITY / MSG_GET returns a TW_ENUMERATION container. The application changes the CurrentIndex and uses DG_CONTROL / DAT_CAPABILITY / MSG_SET to update the capability.
- Example #2, a call to DG_CONTROL / DAT_CAPABILITY / MSG_GET returns a TW_RANGE container. The application changes the CurrentValue and uses DG_CONTROL / DAT_CAPABILITY / MSG_SET to update the capability.

This does not imply or require support for constraining capabilities, the Source is only obligated to update the current value of the capability. If the Source does not support constraints for a capability, and the constraining values have been changed by the application, then the Source should apply the current value according to its own constraints, and if that value is valid, return TWRC_CHECKSTATUS to alert that application that it needs to do a MSG_GET to validate its changes.

- Example #3, if a Source supports the following range for ICAP_BRIGHTNESS: -1000.0 to -1000.0 in steps of 20.0, and if the current value is 0.0, then a call to DG_CONTROL / DAT_CAPABILITY / MSG_GET results in the following:

  ```
  twrange.ItemType     = TWTY_FIX32
  twrange.MinValue     = -1000.0
  twrange.MaxValue     = 1000.0
  twrange.StepSize     = 20.0
  twrange.DefaultValue = 0.0
  twrange.CurrentValue = 0.0
  ```

  If the application sets twrange.CurrentValue to 900.0 and sends this structure to the Source using DG_CONTROL / DAT_CAPABILITY / MSG_SET, the call succeeds and returns TWRC_SUCCESS.

  If the application sets both twrange.CurrentValue and twrange.MaxValue to 900.0, then the status return depends on the Source. A Source that supports constraints accepts the new value and limits MaxValue to 900.0. A Source that does not support constraints accepts the value 900.0, because it falls in the range of -1000 to 1000, step 20; but it returns TWRC_CHECKSTATUS because it was unable to accept the request to limit MaxValue to 900.0.

Memory Allocation

The TW_CAPABILITY structure used in capability negotiation is both allocated and deallocated by the application. The Container structure pointed to by the hContainer field in TW_CAPABILITY is allocated by the Source for "get" operations (MSG_GET, MSG_GETCURRENT, MSG_GETDEFAULT, MSG_RESET) and by the application for the MSG_SET operation. Regardless of which one allocates the container, the application is responsible for deallocating it when it is done with it.

Limitations Imposed by the Negotiation

If a Source agrees to allow an application to restrict a capability, it is critical that the Source abide by that agreement. If at all possible, the Source's user interface should reflect the agreement and not offer invalid options. The Source should never transfer data that violates the agreement reached during capability negotiation. In that situation, the Source can decide to fail the transfer or somehow adjust the values.

Data Transfers

Transfer Modes

All Sources must support Native and Buffered Memory data transfers. It is strongly suggested that they support Disk File mode, too. The default mode is Native. To select one of the other modes, the application must negotiate the `ICAP_XFERMECH` capability (whose default is `TWSX_NATIVE`). Sources must support negotiation of this capability. Refer to Chapter 12, "Operating System Dependencies" for information on each Operating System.

Initiating a Transfer

Transfers are initiated by the application (using the `DG_IMAGE / DAT_IMAGExxxxFER / MSG_GET` operations). A successful transfer transitions the session to State 7. If the transfer fails, the Source returns `TWRC_FAILURE` with the appropriate Condition Code and remains in State 6.

Concluding a Successful Transfer

To signal that the transfer is complete (i.e. the file is completed or the last buffer filled), the Source should return `TWRC_XFERDONE`. In response, the application must send a `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` operation. Only then may the Source transition from State 7 back to State 6 or to State 5 if no more images are ready to be transferred.

If more images are pending transfer, the Source must wait in State 6 until the application either requests the transfer or aborts the transfers. The Source may not "time-out" on any TWAIN transaction.

Aborting a Transfer

Either the application or Source can originate the termination of a transfer (although the application cannot do this in the middle of a Native or Disk File mode transfer). The Source generally terminates the transfer if the user cancels the transfer or a device error occurs which the Source determines is fatal to the transfer or the connection with the application. If the user canceled the transfer, the Source should return `TWRC_CANCEL` to signal the premature termination. The session remains in State 7 until the application sends the `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` operation. If the Source aborts the transfer, it returns `TWRC_FAILURE` and the session typically remains in State 6. (If the failure occurs during the second buffer, or a later buffer, of a Buffered Memory transfer, the session remains in State 7.)

Native Mode Transfers

On Native mode transfers, the data parameter in the `DSM_Entry` call is a pointer to the image handle. Refer to Chapter 12, "Operating System Dependencies" about each OS native file format.

On Windows

Data points to a handle to a DIB (Device Independent Bitmap) located in memory.

On Macintosh

If both the application and the data source are TWAIN 2.4 and later: Data points to a handle to a TIFF image in memory.
If either the application or the data source is TWAIN 2.3 and earlier: Data points to a handle to a Picture (a PicHandle). It is a Quick Draw picture located in memory.

On Linux

Data points to a handle to a TIFF image. It is a TIFF file located in memory.

Native transfers require the data to be transferred to a single large block of RAM. Therefore, they always face the risk of having an inadequate amount of RAM available to perform the transfer successfully.

If inadequate memory prevents the transfer, the Source has these options:

- Fail the transfer operation- Return `TWRC_FAILURE / TWCC_LOWMEMORY`
- Allow the user to clip the data to fit into available memory - Return `TWRC_XFERDONE`
- Allow the user to cancel the operation - Return `TWRC_CANCEL`

If the operation fails, the session remains in State 6. If the operation is canceled, the session remains in State 7 awaiting the `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` or `MSG_RESET` from the application. The application can return the session to State 4 and attempt to renegotiate the transfer mechanism (`ICAP_XFERMECH`) to Disk File or Buffered Memory mode.

The Source cannot be interrupted by the application when it is acquiring an image through Native Mode Transfer. The Source's user interface may allow the user to abort the transfer, but the application will not be able to do so even if the application presents its own acquisition user interface.

Disk File Mode Transfers

The Source selects a default file format and file name (typically, `TWAIN.TMP` in the current directory). It reports this information to the application in response to the `DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET`.

The application may determine all of the Source's supported file formats by using the `ICAP_IMAGEFILEFORMAT` capability. Based on this information, the application can request a particular file format and define its own choice of file name for the transfer. The desired file format and file name will be communicated to the Source in a `DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET`.

When the Source receives the DG_IMAGE / DAT_IMAGEFILEXFER / MSG_SET operation, it should transfer the data into the designated file. The following conditions may exist:

Condition	How to Handle
No file name and/or file format was specified by the application during setup	Use either the Source's default file name or the last file information given to the Source by the application. Create the file.
The application specified a file but failed to create it	Create the application's defined file.
The application's specified file exists but has data in it	Overwrite the existing data.

The Source cannot be interrupted by the application when it is acquiring a file. The Source's user interface may allow the user to abort the transfer, but the application will not be able to do so even if the application presents its own acquisition user interface.

Buffered Memory Mode Transfers

When the Source transfers strips of data, the application allocates and deallocates buffers used for a Buffered Memory mode transfer. However, the Source must recommend appropriate sizes for those buffers and should check that the application has followed its recommendations.

When the Source transfers tiles of data, the Source allocates the buffers. The application is responsible for deallocating the memory.

To determine the Source's recommendations for buffer sizes, the application performs a DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation. The Source fills in the MinBufSize, MaxBufSize, and Preferred fields to communicate its buffer recommendations. Buffers must be double-word aligned and padded with zeros per raster line.

When an application issues a DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET operation, check the TW_IMAGEMEMXFER.Memory.Length field to determine the size of the buffer being presented to you. If it does not fit the recommendations, fail the operation with TWRC_FAILURE / TWCC_BADVALUE.

If the buffer is an appropriate size, fill in the required information.

- Sources must write one or more complete lines of image data (the full width of a strip or tile) into the buffer. Partial lines of image data are not allowed. If some of the buffer is unused, fill it in with zeros. Additionally, each line must be aligned to a 32-bit boundary. Return TWRC_SUCCESS after each successful buffer except for the last one (return TWRC_XFERDONE after that one).
- If the Source is transferring data whose bit depth is not 8 bits, it should fill the buffer without padding the data. If a 5-bit device wants the application to interpret its data as 8-bit data, it should report that it is supplying 8-bit data, make the valid data bits the most significant bits in the data byte, and pad the least significant bits with bits of whichever sense is "lightest". Otherwise, the Source should pack the data bits together. For a 5-bit R-G-B device, that means the data for the green channel should immediately follow the last bit of the red channel. The application is responsible for "unpacking" the data. The Source reports how many bits it is providing per pixel in the BitsPerPixel field of the TW_IMAGEINFO data structure.

Error Handling

Operation Triplet and State Verification

- Sources support all defined TWAIN triplets. A Source must verify every operation triplet they receive. If the operation is not recognized, the Source should return `TWRC_FAILURE` and `TWCC_BADPROTOCOL`.
- Sources must also maintain an awareness of what state their session is in. If an application invokes an operation that is invalid in the current state, the Source should fail the operation and return `TWRC_FAILURE` and `TWCC_SEQERROR`. Valid states for each operation are listed in Chapter 7, "Operation Triplets".
- Anytime a Source fails an operation that would normally cause the session to transition to another state, the session should not transition but should remain in the original state.
- Each triplet operation has its own set of valid Return and Condition Codes as listed in Chapter 7, "Operation Triplets". The Source must return a valid Return Code and set a valid Condition Code, if applicable, following every operation.
- All Return Codes and Condition Codes in the Source should be cleared upon the next call to `DS_Entry( )`. Clearing is delayed when a `DG_CONTROL / DAT_STATUS / MSG_GET` operation is received. In this case, the Source will fill the `TW_STATUS` structure with the current condition information and then clear that information.
- If an application attempts to connect to a Source that only supports a single connection when the source is already opened, the Source should respond with `TWRC_FAILURE` and `TWCC_MAXCONNECTIONS`.
- For Windows Sources only, the DLL implementation makes it possible to be connected to more than one application. Unless the operation request is to open the Source, the Source must verify the application originating an operation is currently connected to the Source. To do this:

  The Source must maintain a list containing the Id value for each connected application. (The `Id` value comes from the application's `TW_IDENTITY` structure which is referenced by the `pOrigin` parameter in the *`DS_Entry( )`* call.)

  The Source should check the `TW_IDENTITY.Id` information of the application sending the operation and verify that it appears in the Source's list of connected applications.
- For Windows only, if connected to multiple applications, the Source is responsible for maintaining a separate, current Condition Code for each application it is connected to. The Source writer should also maintain a temporary, and separate, Condition Code for any application that is attempting to establish a connection with the Source. This is true both for Sources that support only a single connection or have reached the maximum connections.

Unrecoverable Error Situations

The Source is solely responsible for determining whether an error condition within the Source is recoverable or not. The Source must determine when, and what, error condition information to present to the user. The application relies on the Source to specify when a failure occurs. If a Source is in an unrecoverable error situation, it may send a `MSG_CLOSEDSREQ` to the application to request to have its user interface disabled and have an opportunity to begin again.

Memory Management

The Source does not have unlimited memory available, so it should be conservative in its use. It is valid for an application to open a Source and leave it open between several acquires. Therefore, Sources should minimize the time and resources required to load and remain open (in State 4). It is important for the Source writer to recognize that their Source will be using the memory heap of the host application, not its own heap. Therefore, the Source should be conscientious with allocation and de-allocation of memory.

General Guidelines

The following are some general guidelines:

- Check, when the Source is launched, to assure that enough memory space is available for adequate execution.
- Always verify that allocations were successful.
- Work with relocatable objects whenever possible - the heap you fragment is not your own.
- Deallocate temporary memory objects as soon as they are no longer needed.
- Maintain as small a non-operating memory footprint as can prudently be done - the Source will be “compatible” with more applications on more machines.
- Clean up after yourself. When about to be closed, deallocate all locally allocated RAM, eliminate any other objects on the heap, and prepare as appropriate to terminate.

Local Variables

- The Source may allocate and maintain local variables and buffers. Remember that you are borrowing RAM from the application so be efficient about how much RAM is allocated simultaneously.

Instances Where the Source Allocates Memory

In general, the application allocates all necessary structures and passes them to the Source. There are a few exceptions to this rule:

- The Source must create the container, pointed to by the hContainer field, needed to hold capability information on `DG_CONTROL / DAT_CAPABILITY / MSG_GET`, `MSG_GETCURRENT`, `MSG_GETDEFAULT`, or `MSG_RESET` operations. The application deallocates the container.
- The Source allocates the buffer for Native mode data transfers. The application deallocates the buffer.
- Normally, the application creates the buffers used in a Buffered Memory transfer (`DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET`). However, if the Source is transferring tiled data, rather than strips of data, it is responsible for allocating the buffers. The application deallocates the buffers.

See the `DG_IMAGE / DAT_JPEGCOMPRESSION` operations.

Requirements for a Source to be TWAIN-Compliant

The following lists of triplets and capabilities map out the minimum required set of features that a Source must offer programmatically to be TWAIN compliant. Sources, though, are strongly encouraged to go beyond this list and implement as many of their capabilities as possible for programmatic access.

Initially, this list is organized by versions of TWAIN to help Source writers decide which version they wish to support. It is also intended for Applications writers, who can use this information to identify the real level of TWAIN support provided by a Source if its reported version is not matched by the items in this list. Further in this section, additional mandatory capabilities are listed based on the value set for a Capability that has been implemented, or when a Source with a specific feature is being used.

Operations	**Version Required**
DG_CONTROL / DAT_CAPABILITY / MSG_GET	1.5
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT	1.5
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT	1.5
DG_CONTROL / DAT_CAPABILITY / MSG_RESET	1.5
DG_CONTROL / DAT_CAPABILITY / MSG_SET	1.5
DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT	1.5
DG_CONTROL / DAT_IDENTITY / MSG_GET	1.5
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS	1.5
DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS	1.5
DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER	1.5
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET	1.5
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET	1.5
DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET	1.5
DG_CONTROL / DAT_STATUS / MSG_GET	1.5
DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS	1.5
DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS	1.5
DG_CONTROL / DAT_XFERGROUP / MSG_GET	1.5
DG_IMAGE / DAT_IMAGEINFO / MSG_GET	1.5
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET	1.5
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT	1.5
DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET	1.5
DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET	1.5
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET	1.5
DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET	1.5
DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT	1.9

Operations	Version Required
DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS * * Support ***both*** UI and Programmatic Control Through: (Show UI == TRUE) [UI Control] (Show UI == FALSE) [Programmatic Control]	1.9
DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL	1.91
DG_CONTROL / DAT_CAPABILITY / MSG_GET * *For TW_BOOL capabilities return enumerations when the Application is 2.0 or greater. Return one value when the application less than 2.0	2.0

Capabilities	Requirements	Version Required
CAP_SUPPORTEDCAPS	MSG_GET required	1.5
CAP_XFERCOUNT	All MSG_* operations required	1.5
ICAP_COMPRESSION	All MSG_GET* operations required	1.5
ICAP_BITDEPTH	All MSG_* operations required	1.5
ICAP_BITORDER	All MSG_* operations required	1.5
ICAP_PLANARCHUNKY	All MSG_GET* operations required	1.5
ICAP_PHYSICALHEIGHT	All MSG_GET* operations required	1.5
ICAP_PHYSICALWIDTH	All MSG_GET* operations required	1.5
ICAP_PIXELFLAVOR	All MSG_GET* operations required	1.5
ICAP_PIXELTYPE	All MSG_* operations required	1.5
ICAP_UNITS	All MSG_* operations required	1.5
ICAP_XFERMECH	All MSG_* operations required	1.5
ICAP_XRESOLUTION	All MSG_* operations required	1.5
ICAP_YRESOLUTION	All MSG_* operations required	1.5
CAP_DEVICEONLINE	MSG_GET required	1.6
CAP_UICONTROLLABLE	MSG_GET required	1.6
CAP_UICONTROLLABLE	(Value = TRUE)	1.9
CAP_SUPPORTEDDATS	All MSG_GET* operations required	2.2
ICAP_XNATIVERESOLUTION	All MSG_GET* operations required for scanners	2.2
ICAP_YNATIVERESOLUTION	All MSG_GET* operations required for scanners	2.2

Mandatory Features Dependencies

SUPPORTED GROUPS

When source supports:	Must support:	Version intro
DF_DS2	DG_CONTROL / DAT_ENTRYPOINT / MSG_SET	2.0

CUSTOM CONTENT

When source supports:	Must respond to:	Mandatory capability:	Version intro
Custom Content	DG_CONTROL / DAT_CAPABILITY MSG_GET MSG_GETLABEL MSG_GETLABELENUM	CAP_CUSTOMINTERFACEGUID	2.1

CAP_SEGMENTED

When value is:	Must respond to:	Mandatory capability:	Version intro
TWSG_MANUAL	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_SUPPORTEDCAPSSEGMENTUNIQUE	2.2

ICAP_PIXELTYPE

When value is:	Must respond to:	Mandatory capability:	Version intro
TWPT_BW	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_BITDEPTHREDUCTION	1.5

ICAP_BITDEPTHREDUCTION

When value is:	Must respond to:	Mandatory capability:	Version intro
TWBR_HALFTONE	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_HALFTONES	1.0
TWBR_CUSTHALFTONE	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_CUSTHALFTONE	1.0
TWBR_THRESHOLD	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_THRESHOLD	1.5

ICAP_XFERMECH

When value is:	Must support:	Version intro
TWSX_FILE	ICAP_IMAGEFILEFORMAT	1.0
	DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET, MSG_SET	
	DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET	

ICAP_SUPPORTEDSIZES

When source supports:	Must respond to:	Mandatory capability:	Version intro

Document Scanning using Fixed Frame Sizes	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`ICAP_SUPPORTEDSIZES`	1.0

Document Feeders

Basic document feeder devices are those that have paper trays to hold one or more documents for transfer. Unique aspects of a document feeder include the ability to transfer more than one image, the typical inability to re-scan the same page twice, and the fact that if there is no paper loaded, it is usually impossible to scan.

ALL DOCUMENT FEEDERS

When source is a:	Must respond to:	Mandatory capability:	Version intro

Document Feeder	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_FEEDERENABLED`
		`CAP_PAPERDETECTABLE`
		`CAP_AUTOFEED`
			1.0
			1.6
			1.0

CAP_PAPERDETECTABLE

When value is:	Must respond to:	Mandatory capability:	Version intro

`TRUE`	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_FEEDERLOADED`	1.0

CAP_AUTOFEED

When value is:	Must provide *advanced paper handling* through:	Version intro

`FALSE`	`CAP_EXTENDEDCAPS`
	`CAP_FEEDPAGE`
	`CAP_CLEARPAGE`
	`CAP_REWINDPAGE`
		1.0
		1.0
		1.0
		1.0

Special Case

ADF/FLATBED COMBO SCANNER

When source supports:	Must respond to:	Mandatory capability:	Version intro

Flatbed / ADF combo scanner	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_AUTOMATICSENSEMEDIUM`	2.1

DUPLEX

When source supports:	Must respond to:	Mandatory capability:	Version intro
Duplex scanning	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_DUPLEX CAP_DUPLEXENABLED	1.7 1.7

PRINTERS

When source controls:	Must respond to:	Mandatory capabilities:	Version intro
Any printer type device	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_PRINTERENABLED CAP_PRINTER CAP_PRINTERINDEX CAP_PRINTERSTRING CAP_PRINTERSUFFIX	1.8 1.8 2.2 2.2 2.2
When source controls:	**Must respond to:**	**Mandatory capability:**	
Printer vertical position	DG_CONTROL / DAT_CAPABILITY MSG_GET, MSG_GETCURRENT, MSG_GETDEFAULT, MSG_RESET and MSG_SET	CAP_PRINTERVERTICALOFFSET	2.2

CAP_PRINTERMODE

When source supports:	Must respond to:	Mandatory capability:	Version intro
CAP_PRINTERSTRING	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_PRINTERMODE	1.8
CAP_PRINTERSUFFIX	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_PRINTERMODE	1.8

ENDORSER

When source controls:	Must respond to:	Mandatory capabilities:	Version intro
Endorser	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_SUPPORTEDEXTIMAGEINFO CAP_ENDORSER	2.1 1.7

Production Quality High Speed/Volume Scanners

Production Quality High Speed/Volume scanners have greater demands on TWAIN. With diverse features like bar code reading, imprinting and compressions, they require much more attention to detail. Production drivers should be prepared to serve applications that wish to achieve complete programmatic control of all typical and custom features and this requires a VERY robust TWAIN implementation.

Mid- and High-volume scanners must support the following operational triplets:

```
DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY
```

`DG_CONTROL / DAT_CUSTOMDSDATA / MSG_GET & MSG_SET`

INTERNAL IMAGE BUFFER

When source supports:	Must respond to:	Mandatory capabilities:	Version intro
Transfer of multiple images ahead of retrieval	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_AUTOSCAN CAP_MAXBATCHBUFFERS	1.6 1.8

ICAP_UNDEFINEDIMAGESIZE

When source supports:	Must respond to:	Mandatory capability:	Version intro
ICAP_AUTOSIZE	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_UNDEFINEDIMAGESIZE	2.0
ICAP_AUTOMATICBORDERDETECTION	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_UNDEFINEDIMAGESIZE	1.8
ICAP_AUTOMATICLENGTHDETECTION	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_UNDEFINEDIMAGESIZE	2.1
ICAP_AUTOMATICROTATE	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_UNDEFINEDIMAGESIZE	1.8
ICAP_FLIPROTATION	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_UNDEFINEDIMAGESIZE	1.8

ICAP_COMPRESSION

When value is:	Must respond to:	Mandatory capabilities:	Version intro
TWCP_JPEG	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_JPEGPIXELTYPE ICAP_JPEGQUALITY ICAP_JPEGSUBSAMPLING	1.5 1.9 2.2
TWCP_GROUP32D	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_CCITTKFACTOR	1.0

EXTENDED IMAGE INFO

When source supports::	Must respond to:	Mandatory capabilities:	Version intro
Extended image info	DG_CONTROL / DAT_CAPABILITY MSG_GET	ICAP_EXTIMAGEINFO ICAP_SUPPORTEDEXTIMAGEINFO	1.7 2.1
		Mandatory values:	

When source supports::	Must respond to:	Mandatory capabilities:	Version intro
Extended image info	DG_CONTROL / DAT_EXTIMAGEINFO	TWEI_DOCUMENTNUMBER TWEI_PAGENUMBER TWEI_CAMERA TWEI_FRAMENUMBER TWEI_FRAME TWEI_PIXELFLAVOR TWEI_PAPERCOUNT	1.9 1.9 1.9 1.9 1.9 1.9 2.2
		Mandatory value:	
Extended image info and duplex	DG_CONTROL / DAT_EXTIMAGEINFO	TWEI_PAGESIDE	2.1

PATCH CODE DETECTION *

When source controls:	Must respond to:	Mandatory capabilities:	Version intro
Patch Code Detection	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_PATCHCODEDETECTIONENABLED ICAP_SUPPORTEDPATCHCODETYPES CAP_JOBCONTROL	1.8 1.8 1.7

* Note: Source must also fill in the extended EOJ field of the TW_PENDINGXFERS structure when CAP_JOBCONTROL is enabled. See DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER

BARCODE DETECTION

When source controls:	Must respond to:	Mandatory capabilities:	Version intro
Bar Code Detection	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_EXTIMAGEINFO ICAP_BARCODEDETECTIONENABLED ICAP_SUPPORTEDBARCODETYPES	1.7 1.8 1.8

ALARMS

When source controls:	Must respond to:	Mandatory capability:	Version intro
Audible alarms	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_ALARMS	1.8
Alarm volume	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_ALARMVOLUME	1.8

MICR DETECTION

When source supports:	Must respond to:	Mandatory capabilities:	Version intro
Reading Micr data	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_EXTIMAGEINFO ICAP_SUPPORTEDEXTIMAGEINFO CAP_MICRENABLED	1.7 2.1 2.0

Permanent Storage/Retrieval Devices

Permanent storage/retrieval devices are unique in that more than one image is stored and the dimensions and bit depth may vary from image to image. These devices could be just a database of images, or a PCMCIA card from a Digital Camera. Such devices need features for browsing the available images, retrieving properties and selecting sets of images for transfer.

PERMANENT STORAGE/RETRIEVAL

When source supports:	**Required operations:**		**Version intro**
Permanent Storage Retrieval	DG_CONTROL / DAT_FILESYSTEM / MSG_COPY, MSG_DELETE, MSG_CREATEDIRECTORY, MSG_AUTOMATICCAPTURED, MSG_FORMATMEDIA, MSG_GETFIRSTFILE, MSG_GETINFO, MSG_GETNEXTFILE, MSG_RENAME		
	Must respond to::	**Mandatory capability:**	
	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_IMAGEDATASET	1.7

ANNOTATION

When source supports:	**Must respond to:**	**Mandatory capabilities:**	**Version intro**
Annotation	DG_CONTROL / DAT_CAPABILITY All MSG_*	CAP_AUTHOR CAP_CAPTION CAP_TIMEDATE	1.0 1.0 1.0

FLASH

When source supports:	**Must respond to:**	**Mandatory capability:**	**Version intro**
Flash	DG_CONTROL / DAT_CAPABILITY All MSG_*	ICAP_FLASHUSED2	1.8

AUDIO DEVICES

When source supports:	**Must respond to:**	**Mandatory capability:**	**Version intro**
Audio snippets to be associated with an image	DG_CONTROL / DAT_CAPABILITY All MSG_*	ACAP_XFERMECH	1.8

When source supports:	**Must respond to:**
Transfer of Audio snippets	DG_CONTROL / DAT_XFERGROUP / MSG_SET with a value of DG_AUDIO
	And support these operations:
	DG_AUDIO/DAT_AUDIOFILEXFER/MSG_GET DG_AUDIO/DAT_AUDIONATIVEXFER/MSG_GET

Portable Capture Devices

Portable capture devices are very similar to permanent storage and retrieval devices in that they typically store a number of images, however they differ in that they often have real time capture opportunities and limitations related to battery life and lenses. Examples of such devices would be Digital Camera's and Camcorders.

ASYNCHRONOUS DEVICE EVENTS

When source supports:	Must respond to:	Mandatory capability:	Version intro

Asynchronous Device Events	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_DEVICEEVENT`	1.8

STREAM IMAGES

When source supports:	Must respond to:	Mandatory capability:	Version intro

Stream of images for Live Preview	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_CAMERAPREVIEWUI`	1.8

AUTOMATIC CAPTURE

When source supports:	Must respond to:	Mandatory capabilities:	Version intro

Automatic capture	`DG_CONTROL / DAT_CAPABILITY`
	`All MSG_*`
		`CAP_AUTOMATICCAPTURE`
		`CAP_TIMEBEFOREFIRSTCAPTURE`
		`CAP_TIMEBETWEENCAPTURES`
			1.8
			1.8
			1.8

Other Topics

Custom Operations

Manufacturers may add custom operations to their Sources. These can also be made known to application manufacturers. This mechanism allows an application to access functionality not normally available from a generic TWAIN Source.

One use of this mechanism might be to implement device-specific diagnostics for a hardware diagnostic program. These custom operations should be used sparingly and never in place of pre-defined TWAIN operations.

Custom operations are defined by specifying special values for Data Groups (DGs), Data Argument Types (DATs), Messages (MSGs), and Capabilities (CAPs). The following areas have been reserved for custom definitions:

Data Groups	**Top 8 bit flags (bits 24 - 31) in the DG identifiers reserved for custom use.**
`DATs`	Designators with values greater than 8000 hex.
`Messages`	Designators with values greater than 8000 hex.
`Capabilities`	Designators with values greater than 8000 hex.

The responsibility for naming and managing the use of custom designators lies wholly upon the TWAIN element originating the designator and the element consuming it. Prior to interpreting a custom designator, the consuming element must check the originating element's `ProductName` string from its `TW_IDENTITY` structure. Since custom operation numbers may overlap, this is the only way to insure against confusion.

6

Entry Points and Triplet Components

Chapter Contents

Entry Points

TWAIN has the following possible entry points:

- **`DSM_Entry( )`** - located in the Source Manager and typically called by applications, with the following exceptions where a Source calls the Source Manager to communicate with an Application:

```
DG_CONTROL / DAT_NULL / MSG_XFERREADY
DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ
DG_CONTROL / DAT_NULL / MSG_CLOSEDSOK
DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT
```

- **`DS_Entry( )`** - located in the Source and called only by the Source Manager.

Programming Basics

- Upon entry, the parameters must be ordered on the stack in Pascal form. Be sure that your code expects this ordering rather than the reverse order that C uses.
- Refer to Chapter 12, "Operating System Dependencies" about each OS Programming Basics.

Data Flags and Data Groups

Versions of the TWAIN Specification up to and including TWAIN 2.0 indicate that the high 8-bits (24 – 31) in the `TW_IDENTITY.SupportedGroups` are reserved for custom use.

TWAIN 2x has taken these bits for use by the Data Flags (DF_APP2, DF_DSM2 and DF_DS2). This breaks backwards capability with previous versions of the Specification. The risk is considered to be very low, since very few Sources or Applications work with these bits. However, the conflict can be managed in the following ways.

- Avoid the use of `0x10000000`, `0x20000000` and `0x40000000`, these correspond to `DF_DSM2`, `DF_APP2` and `DF_DS2`. The remaining bits: `0x01000000`, `0x02000000`, `0x04000000`, `0x08000000` and `0x80000000` are still in the custom space for Applications and Sources, and they will remain free for that use in all subsequent versions of TWAIN.
- Applications can modify their code to recognize when these bits are in use by a particular Source, which has always been a necessary pre-requisite for custom features, since the bits are guaranteed to have different meaning for different vendors.
- These flags are of most interest to the Data Source Manager, which is now open source (they dictate when `DAT_ENTRYPOINT` is called). If a legacy driver is using one of the custom bits, then propose a possible work-around to the TWAIN Working Group.

Declaration of DSM_Entry( )

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

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

Parameters of DSM_Entry( )

pOrigin

This points to a `TW_IDENTITY` structure, allocated by the application, that describes the application making the call. One of the fields in this structure, called Id, is an arbitrary and unique identifier assigned by the Source Manager to tag the application as a unique TWAIN entity. The Source Manager maintains a copy of the application's identity structure, so the application must not modify that structure unless it first breaks its connection with the Source Manager, then reconnects to cause the Source Manager to store the new, modified identity.

pDest

This is set `either` to `NULL` if the application is aiming the operation at the Source Manager or to the `TW_IDENTITY` structure of the `Source` that the application is attempting to reach. The application allocated the space for the Source's identity structure when it decided which `Source` was to be connected. The Source's `TW_IDENTITY.Id` is also uniquely set by the Source Manager when the Source is opened and should not be modified by the `Source`. The application should not count on the value of this field being consistent from one session to the next because the Source Manager reallocates these numbers every time it is opened. The Source Manager keeps a copy of the `Source's` identity structure as should the application and the `Source`.

DG

The Data Group of the operation triplet. Currently, only `DG_CONTROL`, `DG_IMAGE`, and `DG_AUDIO` are defined.

DAT

The Data Argument Type of the operation triplet. A complete list appears later in this chapter.

MSG

The Message of the operation triplet. A complete list appears later in this chapter.

pData

The `pData` parameter is of type `TW_MEMREF` and is a pointer to the data (a variable or, more typically, a structure) that will be used according to the action specified by the operation triplet.

Declaration of DS_Entry( )

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

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

Declaration of TWAIN_Callback( )

This function is registered by the Application and is only called by the Source Manager. The actual name of the function is up to the application. Written in C code form, the declaration looks like this:

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

Data Groups

TWAIN operations can be broadly classified into three data groups:

Control Oriented (DG_CONTROL)

Controls the TWAIN session. Consumed by both Source Manager and Source. It is always available, no matter what the current setting of `DG_CONTROL / DAT_XFERGROUP`.

Image Data Oriented (DG_IMAGE)

Indicates the kind of data to be transferred. Change between `DG_AUDIO` and `DG_IMAGE` as needed using `DG_CONTROL / DAT_XFERGROUP / MSG_SET`. The default at startup is for a Source to be ready to transfer `DG_IMAGE` data.

Audio Data Oriented (DG_AUDIO)

Indicates the kind of data to be transferred. Change between `DG_AUDIO` and `DG_IMAGE` as needed using `DG_CONTROL / DAT_XFERGROUP / MSG_SET`.

Currently, only image and audio data are supported but this could be expanded to include text, etc. This has several future implications. If more than one data type exists, an application and a Source will need to decide what type(s) of data the *Source* can, and will be allowed to, produce before a transfer can occur. Further, if multiple transfers are being generated from a single acquisition—such as when image and text are intermixed and captured from the same page—it must be unambiguous which type of data is being returned from each data transfer.

Programming Basics

Note the following:

- Data Group designators are 32-bit, unsigned values. The actual values that are assigned are powers of two (bit flags) so that the DGs can be easily masked.
- There are 24 DGs designated as *reserved* for pre-defined DGs . Four are currently in use. The top 8 bits are reserved for custom DGs.

Data Argument Types

Data Argument Types, or DATs, are used to allow programmatic identification of the TWAIN type for the structure of status variable referenced by the entry point parameter `pData`. `pData` will always point to a variable or data structure defined by TWAIN. If the consuming application or Source switches (cases, etc.) on the DAT specified in the formal parameter list of the entry point call, it can handle the form of the referenced data correctly.

Data Type	Used by	Associated structure or type
`DAT_NULL`	`ANY DG`	Null structure. No data required for the operation

Data Type	Used by	Associated structure or type
DAT_CUSTOMBASE	n/a	Not a DAT in itself, but the baseline a Source must use when creating a custom DAT.
DAT_AUDIOFILEXFER	DG_AUDIO	Operates on null data. Filename / Format already negotiated.
DAT_AUDIONATIVEXFER	DG_AUDIO	TW_HANDLE On Windows - WAV handle On Macintosh - audio handle On Linux - WAV handle
DAT_CAPABILITY	DG_CONTROL	TW_CAPABILITY structure
DAT_ENTRYPOINT	DG_CONTROL	TW_ENTRYPOINT structure
DAT_EVENT	DG_CONTROL	TW_EVENT structure
DAT_FILESYSTEM	DG_CONTROL	TW_FILESYSTEM structure
DAT_IDENTITY	DG_CONTROL	TW_IDENTITY structure
DAT_PARENT	DG_CONTROL	TW_HANDLE On Windows - Window handle On Macintosh - Not used. Set to NULL On Linux - Not used. Set to NULL
DAT_PASSTHRU	DG_CONTROL	TW_PASSTHRU structure
DAT_PENDINGXFERS	DG_CONTROL	TW_PENDINGXFERS structure
DAT_SETUPFILEXFER	DG_CONTROL	TW_SETUPFILEXFER structure
DAT_SETUPMEMXFER	DG_CONTROL	TW_SETUPMEMXFER structure
DAT_STATUS	DG_CONTROL	TW_STATUS structure
DAT_USERINTERFACE	DG_CONTROL	TW_USERINTERFACE structure
DAT_XFERGROUP	DG_CONTROL	TW_UINT32 A DG designator describing data to be transferred (currently only image data is supported)
DAT_CIECOLOR	DG_IMAGE	TW_CIECOLOR structure
DAT_GRAYRESPONSE	DG_IMAGE	TW_GRAYRESPONSE structure
DAT_IMAGEFILEXFER	DG_IMAGE	Operates on NULL data. Filename/Format already negotiated
DAT_IMAGEINFO	DG_IMAGE	TW_IMAGEINFO structure
DAT_IMAGELAYOUT	DG_IMAGE	TW_IMAGELAYOUT structure
DAT_IMAGEMEMXFER	DG_IMAGE	TW_IMAGEMEMXFER structure
DAT_IMAGEMEMFILEXFER	DG_IMAGE	TW_IMAGEMEMXFER structure

Data Type	Used by	Associated structure or type
DAT_IMAGENATIVEXFER	DG_IMAGE	TW_HANDLE; On Windows - DIB handle On Macintosh - handle to TIFF image if data source and application are version 2.4 or later. PicHandle if either the application or the data source is TWAIN 2.3 and earlier. On Linux - handle to TIFF image
DAT_JPEGCOMPRESSION	DG_IMAGE	TW_JPEGCOMPRESSION structure
DAT_PALETTE8	DG_IMAGE	TW_PALETTE8 structure
DAT_RGBRESPONSE	DG_IMAGE	TW_RGBRESPONSE structure

Messages

A Message, or MSG, is used to communicate between TWAIN elements what action is to be taken upon a particular piece of data, or for a data-less operation, what action to perform. If an application wants to make anything happen in, or inquire any information from, a Source or the Source Manager, it must make a call to `DSM_Entry( )` with the proper `MSG` as one parameter of the operation triplet. The data to be acted upon is also specified in the parameter list of this call.

A MSG is always associated with a Data Group (`DG`) identifier and a Data Argument Type (`DAT`) identifier in an operation triplet. This operation unambiguously specifies what action is to be taken on what data. Refer to Chapter 7, "Operation Triplets" for the list of defined operation triplets.

Message ID	Valid DAT(s)	Description of Specified Action
MSG_AUTOMATICCAPTURE DIRECTORY	DAT_FILESYSTEM	Place to store images acquired during automatic capture
MSG_CHANGEDIRECTORY	DAT_FILESYSTEM	Change device, domain, host, or image directory
MSG_CLOSEDS	DAT_IDENTITY	Close the specified Source
MSG_CLOSEDSM	DAT_PARENT	Close the Source Manager
MSG_CLOSEDSOK	DAT_NULL	Source requests for application to close `Source`
MSG_CLOSEDSREQ	DAT_NULL	Source requests for application to close `Source`
MSG_COPY	DAT_FILESYSTEM	Copy images across storage devices
MSG_CREATEDIRECTORY	DAT_FILESYSTEM	Create an image directory
MSG_CUSTOMBASE	n/a	Not a message in itself, but the baseline a `Source` must use when creating a custom message
MSG_DELETE	DAT_FILESYSTEM	Delete an image or an image directory

Message ID	Valid DAT(s)	Description of Specified Action
MSG_DEVICEEVENT	DAT_NULL	Report an event from the Source to the Source Manager
MSG_DISABLEDS	DAT_USERINTERFACE	Disable data transfer in the Source
MSG_ENABLEDS	DAT_USERINTERFACE	Enable data transfer in the Source
MSG_ENDXFER	DAT_PENDINGXFERS	Application tells Source that transfer is over
MSG_FORMATMEDIA	DAT_FILESYSTEM	Format a storage device
MSG_GET	various DATs	Get all Available Values including Current & Default
MSG_GETCLOSE	DAT_FILESYSTEM	Close a file context created by MSG_GETFIRSTFILE
MSG_GETCURRENT	various DATs	Get Current value
MSG_GETDEFAULT	various DATs	Get Source's preferred default value
MSG_GETFIRST	DAT_IDENTITY	Get first element from a "list"
MSG_GETFIRSTFILE	DAT_FILESYSTEM	Get the first file in a directory
MSG_GETINFO	DAT_FILESYSTEM	Get information about the current file
MSG_GETNEXT	DAT_IDENTITY	Get next element from a "list"
MSG_GETNEXTFILE	DAT_FILESYSTEM	Get the next file in a directory
MSG_NULL	None	No action to be taken
MSG_OPENDS	DAT_IDENTITY	Open and Initialize the specified Source
MSG_OPENDSM	DAT_PARENT	Open the Source Manager
MSG_PASSTHRU	DAT_PASSTHRU	For use by Source Vendors only
MSG_PROCESSEVENT	DAT_EVENT	Tells Source to check if event/message belongs to it
MSG_RENAME	DAT_FILESYSTEM	Rename an image or an image directory
MSG_RESET	various DATs	Return specified item to power-on (TWAIN default) condition
MSG_SET	various DATs	Set one or more values
MSG_USERSELECT	DAT_IDENTITY	Presents dialog of all Sources to select from
MSG_XFERREADY	DAT_NULL	The Source has data ready for transfer to the application

Custom Components of Triplets

Custom Data Groups

A manufacturer may choose to implement custom data descriptors that require a new Data Group. This would be needed if someone decides to extend TWAIN to, say, satellite telemetry.

- The top 8 bits of every `DG_xxxx` identifier are `reserved` for use as custom `DG`s. Custom `DG` identifiers must use one of the upper 8 bits of the DG_xxxx identifier. Remember, DGs are bit flags.
- The originator of the custom DG must fill the `ProductName` field in the application or Source's `TW_IDENTITY` structure with a uniquely descriptive name. The consumer will look at this field to determine whose custom DG is being used.
- TWAIN provides no formal allocation (or vendor-specific "identifier blocks") for custom data group identifiers nor does it do any coordination to avoid collisions.
- The `DG_CUSTOMBASE` value resides in the `TWAIN.H` file. All custom IDs must be numerically greater than this base. A similar custom base "address" is defined for Data Argument Types, Messages, Capabilities, Return Codes, and Condition Codes. The only difference in concept is that DGs are the only designators defined as bit flags. All other custom values can be any integer value larger than the `xxxx_CUSTOMBASE` defined for that type of designator.

Custom Data Argument Types

`DAT_CUSTOMBASE` is defined in the `TWAIN.H` file to allow a Source vendor to define "custom" DATs for their particular device(s). The application can recognize the Source by checking the `TW_IDENTITY`.ProductName and the `TW_IDENTITY.TW_VERSION` structure. If an application is aware that this particular Source offers custom DATs, it can use them. No changes to TWAIN or the Source Manager are required to support such identifiers (or the data structures which they imply).

Refer to the `TWAIN.H` file for the value of `DAT_CUSTOMBASE` for custom `DAT`s. All custom values must be numerically greater than this constant.

Custom Messages

As with the `DAT`s, `MSG_CUSTOMBASE` is included in `TWAIN.H` so that the Source writer can create custom messages specific to their Source. If the applications understand these custom messages, actions beyond those defined in this specification can be performed through the normal TWAIN mechanism. No modifications to TWAIN or the Source Manager are required.

Remember that the consumer of these custom values will look in your `TW_IDENTITY.ProductName` field to clarify what the identifier's value means—there is no other protection for overlapping custom definitions. Refer to the `TWAIN.H` file for the value of `MSG_CUSTOMBASE` for custom Messages. All custom values must be numerically greater than this value.

7

Operation Triplets

Chapter Contents

Triplet Overview

From Application to Source Manager (Control Information)

Data Group	Data Argument Type	Message	Page
DG_CONTROL	DAT_IDENTITY	MSG_CLOSEDS	7-58
		MSG_GETDEFAULT	7-61
		MSG_GETFIRST	7-62
		MSG_GETNEXT	7-64
		MSG_OPENDS	7-66
		MSG_SET	7-69
		MSG_USERSELECT	7-70
DG_CONTROL	DAT_PARENT	MSG_CLOSEDSM	7-78
		MSG_OPENDSM	7-79
DG_CONTROL	DAT_STATUS	MSG_GET	7-98

From Application to Source (Control Information)

Data Group	Data Argument Type	Message	Page
DG_CONTROL	DAT_CAPABILITY	MSG_GET	7-13
		MSG_GETCURRENT	7-16
		MSG_GETDEFAULT	7-19
		MSG_GETHELP	7-21
		MSG_GETLABEL	7-22
		MSG_GETLABELENUM	7-23
		MSG_QUERYSUPPORT	7-25
		MSG_RESET	7-27
		MSG_RESETALL	7-30
		MSG_SET	7-32
		MSG_SETCONSTRAINT	7-35
DG_CONTROL	DAT_CUSTOMDSDATA	MSG_GET	7-38
		MSG_SET	7-39
DG_CONTROL	DAT_DEVICEEVENT	MSG_GET	7-40
DG_CONTROL	DAT_FILESYSTEM	MSG_AUTOMATICCAPTURE DIRECTORY	7-45
		MSG_CHANGEDIRECTORY	7-46
		MSG_COPY	7-48
		MSG_CREATEDIRECTORY	7-49
		MSG_DELETE	7-50
		MSG_FORMATMEDIA	7-51
		MSG_GETCLOSE	7-52
		MSG_GETFIRSTFILE	7-53
		MSG_GETINFO	7-55
		MSG_GETNEXTFILE	7-56
		MSG_RENAME	7-57
DG_CONTROL	DAT_EVENT	MSG_PROCESSEVENT	7-43
DG_CONTROL	DAT_METRICS	MSG_GET	7-72
DG_CONTROL	DAT_PASSTHRU	MSG_PASSTHRU	7-80
DG_CONTROL	DAT_PENDINGXFERS	MSG_ENDXFER	7-81
		MSG_GET	7-83
		MSG_RESET	7-85
		MSG_STOPFEEDER	7-87

Data Group	Data Argument Type	Message	Page
DG_CONTROL	DAT_SETUPFILEXFER	MSG_GET	7-88
		MSG_GETDEFAULT	7-90
		MSG_RESET	7-92
		MSG_SET	7-94
DG_CONTROL	DAT_SETUPMEMXFER	MSG_GET	7-96
DG_CONTROL	DAT_STATUS	MSG_GET	7-98
DG_CONTROL	DAT_STATUSUTF8	MSG_GET	7-100
DG_CONTROL	DAT_TWAINDIRECT	MSG_SETTASK	7-101
DG_CONTROL	DAT_USERINTERFACE	MSG_DISABLEDS	7-103
		MSG_ENABLEDS	7-104
		MSG_ENABLEDSUIONLY	7-107
DG_CONTROL	DAT_XFERGROUP	MSG_GET	7-108
		MSG_SET	7-109

From Application to Source (Image Information)

Data Group	Data Argument Type	Message	Page #
DG_IMAGE	DAT_CIECOLOR	MSG_GET	7-110
DG_IMAGE	DAT_EXTIMAGEINFO	MSG_GET	7-111
DG_IMAGE	DAT_GRAYRESPONSE	MSG_RESET	7-118
		MSG_SET	7-119
DG_IMAGE	DAT_ICCPROFILE	MSG_GET	7-120
DG_IMAGE	DAT_IMAGEFILEXFER	MSG_GET	7-122
DG_IMAGE	DAT_IMAGEINFO	MSG_GET	7-124
DG_IMAGE	DAT_IMAGELAYOUT	MSG_GET	7-126
		MSG_GETDEFAULT	7-128
		MSG_RESET	7-129
		MSG_SET	7-130
DG_IMAGE	DAT_IMAGEMEMFILEXFER	MSG_GET	7-132
DG_IMAGE	DAT_IMAGEMEMXFER	MSG_GET	7-135
DG_IMAGE	DAT_IMAGENATIVEXFER	MSG_GET	7-138

Data Group	Data Argument Type	Message	Page #
DG_IMAGE	DAT_JPEGCOMPRESSION	MSG_GET	7-141
		MSG_GETDEFAULT	7-142
		MSG_RESET	7-143
		MSG_SET	7-144
DG_IMAGE	DAT_PALETTE8	MSG_GET	7-145
		MSG_GETDEFAULT	7-146
		MSG_RESET	7-147
		MSG_SET	7-148
DG_IMAGE	DAT_RGBRESPONSE	MSG_RESET	7-149
		MSG_SET	7-150

From Application to Source (Audio Information)

Data Group	Data Argument Type	Message	Page #
DG_AUDIO	DAT_AUDIOFILEXFER	MSG_GET	7-7
DG_AUDIO	DAT_AUDIOINFO	MSG_GET	7-8
DG_AUDIO	DAT_AUDIONATIVEXFER	MSG_GET	7-9

From Source Manager to Source (Control Information)

Data Group	Data Argument Type	Message	Page #
DG_CONTROL	DAT_IDENTITY	MSG_CLOSEDS	7-59
		MSG_GET	7-60
		MSG_OPENDS	7-68

From Source to Application (Control Information via the Source Manager) (Used by Windows Sources only

)

Data Group	Data Argument Type	Message	Page #
DG_CONTROL	DAT_NULL	MSG_CLOSEDSOK	7-74
		MSG_CLOSEDSREQ	7-75
		MSG_DEVICEEVENT	7-76
		MSG_XFERREADY	7-77

TWAIN 2.0 (Entry Points)

Data Group	Data Argument Type	Message	Page #
`DG_CONTROL`	`DAT_ENTRYPOINT`	`MSG_GET`	7-41
		`MSG_SET`	7-42

Format of the Operation Triplet Descriptions

The following pages describe the operation triplets. They are all included and are arranged in alphabetical order using the Data Group, Data Argument Type, and Message identifier list.

There are three operations that are duplicated because that have a different originator and/or destination in each case. They are:

- `DG_CONTROL` / `DAT_IDENTITY` / `MSG_CLOSEDS`
  - 4 from Application to Source Manager
  - 4 from Source Manager to Source
- `DG_CONTROL` / `DAT_IDENTITY` / `MSG_OPENDS`
  - 4 from Application to Source Manager
  - 4 from Source Manager to Source
- `DG_CONTROL` / `DAT_STATUS` / `MSG_GET`
  - 4 from Application to Source Manager
  - 4 from Application to Source

The format of each page is:

Triplet - The Concise DG / DAT / MSG Information

Call

Actual format of the routine call (parameter list) for the operation. Identification of the data structure used for the `pData` parameter is included.

Valid States

The states in which the application, Source Manager, or Source may legally invoke the operation.

Description

General description of the operation.

Origin of the Operation (Application, Source Manager or, Source)

The action(s) the application, Source Manager, or Source should take before invoking the operation.

Destination of the Operation (Source Manager or Source)

The action that the destination element (Source Manager or Source) of the operation will take.

Return Codes

The Return Codes and Condition Codes that are defined and valid for this operation.

See Also

Lists other related operation triplets, capabilities, constants, etc.

Operation Triplets

DG_AUDIO / DAT_AUDIOFILEXFER / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_AUDIO, DAT_AUDIOFILEXFER, MSG_GET, NULL);
```

Valid States

6 (transitions to state 7)

Description

(Similar operation to `DAT_IMAGEFILEXFER`).

This operation is used to initiate the transfer of audio from the Source to the application via the disk-file transfer mechanism. It causes the transfer to begin.

No special set up or action required. Application should have already invoked the `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation, unless the Source's default transfer format and file name (typically, `TWAINAUD.TMP`) are acceptable to the application. The application need only invoke this operation once per image transferred.

Source should acquire the audio data, format it, create any appropriate header information, and write everything into the file specified by the previous `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation, and close the file.

Audio transfers are optional. If an application transfers only the images and never changes to **`DG_AUDIO`**, then the audio snippets will be automatically discarded or skipped by the Source.

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
```

See Also

`ACAP_XFERMECH`

DG_AUDIO / DAT_AUDIOINFO / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_AUDIO, DAT_AUDIOINFO, MSG_GET,
    pSourceAudioInfo);
```

`pSourceAudioInfo` = A pointer to a `TW_AUDIOINFO` structure

Valid States

6 and 7

Description

Used to get the information of the current audio data ready to transfer. (Similar operation to `DAT_IMAGEINFO`)

Return Codes

```
TWRC_SUCCESS
TWRC_FAILURE
    TWCC_BADPROTOCOL
    TWCC_SEQERROR
```

See Also

None

DG_AUDIO / DAT_AUDIONATIVEXFER / MSG_GET

Call

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

`pHandle` = A pointer to a variable of type Handle

On Windows - This is a handle variable to WAV data located in memory.

On Macintosh - This is a handle to AIFF data.

On Linux - This is a handle to WAV data.

Valid States

6 (transitions to state 7)

Description

(Similar operation to `DAT_IMAGENATIVEXFER`).

Causes the transfer of an audioÆs data from the Source to the application, via the Native transfer mechanism, to begin. The resulting data is stored in main memory in a single block. The data is stored in AIFF format on the Macintosh and as a WAV format under Microsoft Windows. The size of the audio snippet that can be transferred is limited to the size of the memory block that can be allocated by the Source.

Note: This is the default transfer mechanism. All Sources support this mechanism if `DG_AUDIO` is supported. The Source will use this mechanism unless the application explicitly negotiates a different transfer mechanism with `ACAP_XFERMECH`.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

    TWCC_BADPROTOCOL.

    TWCC_SEQERROR - not state 6.

See Also

`ACAP_XFERMECH`

DG_CONTROL / DAT_CALLBACK / MSG_INVOKE_CALLBACK

`MSG_INVOKE_CALLBACK` is deprecated. It was added for Mac OS X, TWAIN 2.0 DS should use `DAT_NULL`. Refer to the TWAIN 1.9 spec for implementation.

Call

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

Valid States

4, 5, 7 (depending on the message)

Description

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

The `TW_CALLBACK` structure should be initialized as follows:

`Msg`	`Initialized` to any valid `DG_CONTROL / DAT_NULL` message.

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

`MSG_XFERREADY`

`MSG_CLOSEDSREQ`

`MSG_CLOSEDSOK`

`MSG_DEVICEEVENT`

`MSG_INVOKE_CALLBACK` is the **only** way for a Mac OS X TWAIN 1.9 DS to inform the application of these events.

Return Codes

`TWRC_FAILURE`

See Also

`DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK`
`DG_CONTROL / DAT_CALLBACK2 / MSG_REGISTER_CALLBACK`

DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK

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_CALLBACK2 / MSG_REGISTER_CALLBACK`

DG_CONTROL / DAT_CALLBACK2 / MSG_REGISTER_CALLBACK

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_CALLBACK2,
MSG_REGISTER_CALLBACK, (TW_MEMREF)&callback);
```

`callback =` A pointer to a `TW_CALLBACK2` structure

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_CALLBACK2` structure. The `TW_CALLBACK2` structure should be initialized as follows:

`CallBackProc`	The callback function's entry point, used by `MSG_REGISTER_CALLBACK`.
`RefCon`	An application defined reference constant.

Return Codes

`TWRC_FAILURE`

See Also

`DG_CONTROL / DAT_CALLBACK / MSG_INVOKE_CALLBACK`

DG_CONTROL / DAT_CAPABILITY / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_GET,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 through 7

Description

Returns the Source's Current, Default and Available Values for a specified capability.

These values reflect previous `MSG_SET` or `MSG_SETCONSTRAINT` operations on the capability, or Source's automatic changes. (See `MSG_SET`).

Note: This operation does not change the Current or Available Values of the capability.

Application

Set the `pCapability` fields as follows:

```
pCapability->Cap = the CAP_xxxx or ACAP_xxxx or ICAP_xxxx identifier
pCapability->ConType = TWON_DONTCARE16
pCapability->hContainer = NULL
```

The Source will allocate the memory for the necessary container structure but the application must free it when the operation is complete and the application no longer needs to maintain the information.

Use `MSG_GET`:

- As the first step in negotiation of a capability's Available Values.
- To check the results if a `MSG_SET` returns `TWRC_CHECKSTATUS`.
- To check the Available, Current and Default Values with one command.

This operation may fail for a low memory condition. Either recover from a `TWCC_LOWMEMORY` failure by freeing memory for the Source to use so it can continue, or terminating the acquisition and notifying the user of the low memory problem.

Source

If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to), disregard the operation, but return `TWRC_FAILURE` with `TWCC_CAPUNSUPPORTED`.

If you support the capability, fill in the fields listed below and allocate the container structure and place its handle into `pCapability->hContainer`. The container should be referenced by a "handle" of type `TW_HANDLE`.

Fill the fields in `pCapability` as follows:

```
pCapability->ConType = TWON_ARRAY,
TWON_ONEVALUE,
TWON_ENUMERATION, or
TWON_RANGE
```

`pCapability->hContainer = TW_HANDLE` referencing a container of `ConType`

Set ConType to the container type your Source uses for this capability. For the container type of `TWON_ONEVALUE` provide the Current Value. For the container type of `TWON_ARRAY` provide the Available Values. For container types `TWON_ENUMERATION` and `TWON_RANGE` provide the Current, Default and Available Values.

This is a memory allocation operation. It is possible for this operation to fail due to a low memory condition. Be sure to verify that the allocation is successful. If it is not, attempt to reduce the amount of memory occupied by the Source. If the allocation cannot be made, return `TWRC_FAILURE` with `TWCC_LOWMEMORY` to the application and set the `pCapability->hContainer` handle to `NULL`.

Note: The `Source` *must* be able to respond to an inquiry about any of its capabilities at *any time* that the `Source` is open.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

  TWCC_BADCAP      /* Unknown capability--Source does not recognize  */
                   /* this capability. This code should not be used  */
                   /* by sources after 1.6. Applications still need  */
                   /* to test for it for backward compatibility.     */

  TWCC_CAPUNSUPPORTED  /* Capability not supported by source. Sources*/
                       /* 1.6 and newer must use this instead of     */
                       /* using TWCC_BADCAP.                         */

  TWCC_CAPBADOPERATION /* Operation not supported by capability.*/
                       /* Sources 1.6 and newer must use this   */
                       /* instead of using TWCC_BADCAP.

  TWCC_CAPSEQERROR     /* Capability has a dependency on another  */
                       /* capability. Sources 1.6 and newer must  */
                       /* use this instead of using TWCC_BADCAP.  */

  TWCC_BADDEST         /* No such Source in session with application */
  TWCC_LOWMEMORY       /* Not enough memory to complete the operation*/
  TWCC_SEQERROR        /* Operation invoked in invalid state         */
```

See Also

```
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT
```

“Capability Containers” on page 2-15 and `TW_ONEVALUE`, `TW_ENUMERATION`, `TW_RANGE`, `TW_ARRAY`.

“Capability Constants” on page 8-75 (in Chapter 8, "Data Types and Data Structures")

Capability Containers: `TW_ONEVALUE`, `TW_ENUMERATION`, `TW_RANGE`, `TW_ARRAY` (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities “The Capability Listings” on page 10-12 (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_GETCURRENT,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 through 7

Description

Returns the Source's Current Value for the specified capability.

The Current Value reflects previous `MSG_SET` operations on the capability, or Source's automatic changes. (See `MSG_SET`).

Note: This operation does not change the Current Values of the capability.

Application

Set the `pCapability` fields as follows:

`pCapability->Cap` = the `CAP_xxxx` or `ACAP_xxxx` or `ICAP_xxxx` identifier
`pCapability->ConType` = `TWON_DONTCARE16`
`pCapability->hContainer` = `NULL`

The Source will allocate the memory for the necessary container structure but the application must free it when the operation is complete and the application no longer needs to maintain the information.

Use `MSG_GETCURRENT`:

- To check the Source's power-on Current Values (see Chapter 10, "Capabilities" for TWAIN-defined defaults for each capability).
- To check just the Current Value (in place of using `MSG_GET`).
- In State 6 to determine the settings. They could have been set by the user (if `TW_USERINTERFACE`.ShowUI = `TRUE`) or be the results of automatic processes used by the Source.

This operation may fail for a low memory condition. Either recover from a `TWCC_LOWMEMORY` failure by freeing memory for the Source to use so it can continue, or terminating the acquisition and notifying the user of the low memory problem.

Source

If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to), disregard the operation, but return `TWRC_FAILURE` with `TWCC_CAPUNSUPPORTED`.

If you support the capability, fill in the fields listed below and allocate the container structure and place its handle into `pCapability->hContainer`. The container should be referenced by a "handle" of type `TW_HANDLE`.

Fill the fields in `pCapability` as follows:

```
pCapability->ConType = TWON_ARRAY or TWON_ONEVALUE
pCapability->hContainer = TW_HANDLE referencing a container of ConType
```

Set `ConType` to the container type that matches the type for this capability. Fill the fields in the container structure with the Current Value of the capability.

This is a memory allocation operation. It is possible for this operation to fail due to a low memory condition. Be sure to verify that the allocation is successful. If it is not, attempt to reduce the amount of memory occupied by the Source. If the allocation cannot be made, return `TWRC_FAILURE` with `TWCC_LOWMEMORY` to the application and set the `pCapability->hContainer` handle to `NULL`.

Note that the Source **must** be able to respond to an inquiry about any of its capabilities at **any time** that the Source is open.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

  TWCC_BADCAP         /* Unknown capability--Source does not recognize */
                      /* this capability. This code should not be used */
                      /* by sources after 1.6. Applications still need */
                      /* to test for it for backward compatibility.    */

  TWCC_CAPUNSUPPORTED   /* Capability not supported by source. Sources*/
                        /* 1.6 and newer must use this instead of     */
                        /* using TWCC_BADCAP.                         */

  TWCC_CAPBADOPERATION  /* Operation not supported by capability.     */
                        /* Sources 1.6 and newer must use this instead*/
                        /* of using TWCC_BADCAP.                      */

  TWCC_CAPSEQERROR      /* Capability has a dependency on another     */
                        /* capability. Sources 1.6 and newer must use */
                        /* this instead of using TWCC_BADCAP.         */

  TWCC_BADDEST          /* No such Source in-session with             */
                        /* application                                */

  TWCC_LOWMEMORY        /* Not enough memory to complete the          */
                        /* operation                                  */

  TWCC_SEQERROR         /* Operation invoked in invalid  state.       */
```

See Also

```
DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT
```

Capability Constants (in Chapter 8, "Data Types and Data Structures")

Capability Containers: `TW_ONEVALUE`, `TW_ENUMERATION`, `TW_RANGE`, `TW_ARRAY` (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities").

DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_GETDEFAULT,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 through 7

Description

Returns the Source's Default Value. This is the Source's preferred default value.

The Source's Default Value cannot be changed.

Application

Set the `pCapability` fields as follows:

```
pCapability->Cap = the CAP_xxxx or ACAP_xxxx or ICAP_xxxx identifier
pCapability->ConType = TWON_DONTCARE16
pCapability->hContainer = NULL
```

The Source will allocate the memory for the necessary container structure but the application must free it when the operation is complete and the application no longer needs to maintain the information.

Use `MSG_GETDEFAULT`:

- To check the Source's preferred Values. Using the Source's preferred default as the Current Value may increase performance in some Sources.

This operation may fail for a low memory condition. Either recover from a `TWCC_LOWMEMORY` failure by freeing memory for the Source to use so it can continue, or terminating the acquisition and notifying the user of the low memory problem.

Source

If the application requests this operation on a capability your Source does not recognize (and you are sure you have implemented all the capabilities that you're required to), disregard the operation, but return `TWRC_FAILURE` with `TWCC_CAPUNSUPPORTED`.

If you support the capability, fill in the fields listed below and allocate the container structure and place its handle into `pCapability->hContainer`. The container should be referenced by a "handle" of type `TW_HANDLE`.

- Fill the fields in `pCapability` as follows:

```
pCapability->ConType = TWON_ARRAY or TWON_ONEVALUE
```

```
pCapability->hContainer = TW_HANDLE referencing a container of ConType
```

Set `ConType` to the container type that matches for this capability. Fill the fields in the container with the Default Value of this capability.

The Default Value is the preferred value for the Source. This value is used as the power-on value for capabilities if TWAIN does not specify a default.

This is a memory allocation operation. It is possible for this operation to fail due to a low memory condition. Be sure to verify that the allocation is successful. If it is not, attempt to reduce the amount of memory occupied by the Source. If the allocation cannot be made return `TWRC_FAILURE` with `TWCC_LOWMEMORY` to the application and set the `pCapability->hContainer` handle to `NULL`.

Note that the Source **must** be able to respond to an inquiry about any of its capabilities at **any time** that the Source is open.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

  TWCC_BADCAP          /* Unknown capability--Source does not recognize */
                       /* this capability. This code should not be used */
                       /* by sources after 1.6. Applications still need */
                       /* to test for it for backward compatibility.    */

  TWCC_CAPUNSUPPORTED     /* Capability not supported by source. Sources*/
                          /* 1.6 and newer must use this instead of     */
                          /* using TWCC_BADCAP.                         */

  TWCC_CAPBADOPERATION    /* Operation not supported by capability.    */
                          /* Sources 1.6 and newer must use this instead*/
                          /* of using TWCC_BADCAP.

  TWCC_CAPSEQERROR        /* Capability has a dependency on another     */
                          /* capability. Sources 1.6 and newer must use */
                          /* this instead of using TWCC_BADCAP.         */

  TWCC_BADDEST           /* No such Source in-session with             */
                         /* application                                 */

  TWCC_LOWMEMORY          /* Not enough memory to complete the          */
                          /* operation                                  */

  TWCC_SEQERROR           /* Operation invoked in invalid state         */
```

See Also

DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT

Capability Constants (in Chapter 10, "Capabilities")

Capability Containers: TW_ONEVALUE, TW_ENUMERATION, TW_RANGE, TW_ARRAY (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CAPABILITY / MSG_GETHELP

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_CAPABILITY, MSG_GETHELP,
pTwCapability);
```

`pTwCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4

Description

Returns help text suitable for use in a GUI; for instance: "Specify the amount of detail in an image. Higher values result in more detail." for `ICAP_XRESOLUTION`.

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`

See Also

`DG_CONTROL / DAT_CAPABILITY / MSG_GETLABEL`

DG_CONTROL / DAT_CAPABILITY / MSG_GETLABEL

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_CAPABILITY, MSG_GETLABEL,
pTwCapability);
```

`pTwCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4

Description

Returns a label suitable for use in a GUI, for instance “Resolution:” for ICAP_XRESOLUTION.

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

See Also

DG_CONTROL / DAT_CAPABILITY / MSG_GETHELP

DG_CONTROL / DAT_CAPABILITY / MSG_GETLABELENUM

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_CAPABILITY, MSG_GETLABELENUM,
pTwCapability);
```

```
pTwCapability = A pointer to a TW_CAPABILITY structure.
```

Valid States

4

Description

Return all of the labels for a capability of type TW_ARRAY or TW_ENUMERATION, for example "US Letter" for ICAP_SUPPORTEDSIZES' TWSS_USLETTER.

Application

The Application receives a TW_ARRAY with a TW_STR255 type. Each index in the array corresponds to the same index of a TW_ARRAY or a TW_ENUMERATION returned by a MSG_GET for that same capability.

For example, if ICAP_SUPPORTEDSIZES returns the following for MSG_GET:

```
ptwenumeration->ItemType = TWTY_UINT16
ptwenumeration->NumItems = 3
ptwenumeration->CurrentIndex = 0
ptwenumeration->DefaultIndex = 0
((TW_UINT16*)&ptwenumeration->ItemList)[0] = TWSS_USLETTER
((TW_UINT16*)&ptwenumeration->ItemList)[1] = TWSS_A4LEDGER
((TW_UINT16*)&ptwenumeration->ItemList)[2] = TWSS_USEXECUTIVE
```

It should return something like the following for MSG_GETLABELENUM:

```
ptwarray->ItemType = TWTY_STR255
ptwarray ->NumItems = 3
((char*)&ptwarray->ItemList)[0*sizeof(TW_STR255)] is "US Letter"
((char*)&ptwarray->ItemList)[1*sizeof(TW_STR255)] is "A4 Letter"
((char*)&ptwarray->ItemList)[2*sizeof(TW_STR255)] is "US Executive"
```

Source

The Source returns a TW_ARRAY container with a TW_STR255 item type. The string data is UTF-8 encoded. The language is determined by the TW_IDENTITY.TW_VERSION.Language reported back by the Source, unless overridden by CAP_LANGUAGE.

This feature is only supported for capabilities that return TW_ARRAY or TW_ENUMERATION for MSG_GET. Other capabilities (like TW_RANGE or TW_ONEVALUE) return TWRC_FAILURE / TWCC_BADPROTOCOL.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

TWCC_CAPUNSUPPORTED
TWCC_BADPROTOCOL

See Also

DG_CONTROL / DAT_CAPABILITY / MSG_GETLABEL

DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_QUERYSUPPORT,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 through 7

Description

Returns the Source's support status of this capability.

Application

Set the `pCapability` fields as follows:

```
pCapability->Cap = the CAP_xxxx or ACAP_xxxx or ICAP_xxxx identifier
pCapability->ConType = TWON_ONEVALUE
pCapability->hContainer = NULL
```

The Source will allocate the memory for the necessary container structure but the application must free it when the operation is complete and the application no longer needs to maintain the information.

Use `MSG_QUERYSUPPORT`:

- To check the whether the Source supports a particular operation on the capability.

This operation may fail for a low memory condition. Either recover from a `TWCC_LOWMEMORY` failure by freeing memory for the Source to use so it can continue, or terminating the acquisition and notifying the user of the low memory problem.

Source

Fill the fields in `pCapability` as follows:

```
pCapability->ConType = TWON_ONEVALUE
```

```
pCapability->hContainer = TW_HANDLE referencing a container of type
TW_ONEVALUE.
```

Fill the fields in `TW_ONEVALUE` as follows:

1. `ItemType = TWTY_INT32;`
2. Item = Bit pattern representing the set of operations that are supported by the Data Source on this capability (`TWQC_GET, TWQC_SET, TWQC_GETCURRENT, TWQC_GETDEFAULT, TWQC_RESET, TWQC_SETCONSTRAINT`);

If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to), do not disregard the operation, but fill out the `TWON_ONEVALUE` container with a value of zero(0) for the Item field, indicating no support for any of the `DAT CAPABILITY` operations, and return a status of `TWRC_SUCCESS`.

If the capability will currently return `TWRC_FAILURE / TWCC_CAPSEQERROR`, because its availability depends on that of other capabilities, then fill out the `TWON_ONEVALUE` container with a value of zero (0) for the Item field, indicating no support for any of the `DAT CAPABILITY` operations, and return a status of `TWRC_SUCCESS`.

This is a memory allocation operation. It is possible for this operation to fail due to a low memory condition. Be sure to verify that the allocation is successful. If it is not, attempt to reduce the amount of memory occupied by the Source. If the allocation cannot be made return `TWRC_FAILURE` with `TWCC_LOWMEMORY` to the application and set the `pCapability->hContainer` handle to `NULL`.

Note that the Source **must** be able to respond to an inquiry about any of its capabilities at **any time** that the Source is open.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

TWCC_BADDEST        /* No such Source in-session with    */
                    /* application                       */

TWCC_LOWMEMORY      /* Not enough memory to complete the */
                    /* operation                         */
```

See Also

```
DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SET
```

Capability Constants (in Chapter 8, "Data Types and Data Structures")

Capability Container: `TW_ONEVALUE` (in Chapter 8, "Data Types and Data Structures").

Listing of all capabilities (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CAPABILITY / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_RESET,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 (when indicated by `MSG_QUERYSUPPORT`)

5, 6, 7 (when the capability appears in the `CAP_EXTENDEDCAPS` array, and when indicated by `MSG_QUERYSUPPORT`)

Description

Change the Current Value of the specified capability back to the `MSG_RESET/MSG_RESETALL` value and return the new Current Value.

These values are listed in capability section (in Chapter 10, "Capabilities"). If "no default" is specified, the Source uses it preferred default value (returned from `MSG_GETDEFAULT`).

Application

Set the `pCapability` fields as follows:

```
pCapability->Cap = the CAP_xxxx or ACAP_xxxx or ICAP_xxxx identifier
pCapability->ConType = TWON_DONTCARE16
pCapability->hContainer = NULL
```

The Source will allocate the memory for the necessary container structure but the application must free it when the operation is complete and the application no longer needs to maintain the information.

Use `MSG_RESET`:

- To set the Current Value of the specified capability a known default value, and to remove any constraints from the allowed values supported by the Source.

This operation may fail for a low memory condition. Either recover from a `TWCC_LOWMEMORY` failure by freeing memory for the Source to use so it can continue, or terminating the acquisition and notifying the user of the low memory problem.

Source

If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to), disregard the operation, but return `TWRC_FAILURE` with `TWCC_CAPUNSUPPORTED`.

If you support the capability, reset the Current Value of the capability back to its known default value. This value must also match the `MSG_RESET/MSG_RESETALL` value listed in capability section of Chapter 10, "Capabilities".

Also return the new Current Value (just like in a `MSG_GETCURRENT`). Fill in the fields listed below and allocate the container structure and place its handle into `pCapability->hContainer`. The container should be referenced by a "handle" of type `TW_HANDLE`.

Fill the fields in `pCapability` as follows:

`pCapability->ConType = TWON_ARRAY` or `TWON_ONEVALUE`

`pCapability->hContainer = TW_HANDLE` referencing a container of `ConType`

Set `ConType` to the container type that matches the type for this capability. Fill the fields in the container structure with the Current Value of the capability (after resetting it as stated above).

This is a memory allocation operation. It is possible for this operation to fail due to a low memory condition. Be sure to verify that the allocation is successful. If it is not, attempt to reduce the amount of memory occupied by the Source. If the allocation cannot be made return `TWRC_FAILURE` with `TWCC_LOWMEMORY` to the application and set the `pCapability->hContainer` handle to `NULL`.

Note that this operation is only valid in State 4, unless permitted by the presence of the capability in the `CAP_EXTENDEDCAPS` array. Any attempt to invoke it in any other state should be disregarded, though the Source should return `TWRC_FAILURE` with `TWCC_SEQERROR`.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

  TWCC_BADCAP      /* Unknown capability--Source does not recognize */
                     /* this capability. This code should not be used */
                     /* by sources after 1.6. Applications still need */
                     /* to test for it for backward compatibility.    */

  TWCC_CAPUNSUPPORTED   /* Capability not supported by source Sources*/
                        /* 1.6 and newer must use this instead of    */
                        /* using TWCC_BADCAP.                        */

  TWCC_CAPBADOPERATION  /* Operation not supported by capability.    */
                         /* Sources 1.6 and newer must use this instead*/
                        /* of using TWCC_BADCAP.

  TWCC_CAPSEQERROR      /* Capability has a dependency on another     */
                        /* capability. Sources 1.6 and newer must use */
                       /* this instead of using TWCC_BADCAP.          */

  TWCC_BADDEST       /* No such Source in-session with               */
                      /* application                                  */

  TWCC_LOWMEMORY     /* Not enough memory to complete the            */
                      /* operation                                    */

  TWCC_SEQERROR     /* Operation invoked in invalid state           */
```

See Also

```
DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_SET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT
```

“Capability Constants” on page 8-75.

Capability Containers: `TW_ONEVALUE`, `TW_ENUMERATION`, `TW_RANGE`, `TW_ARRAY` (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities".)

DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_RESETALL,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 only

Description

This command resets all current values back to the `MSG_RESET/MSG_RESETALL` values. All current values are set to their known default value. These values are listed in the capabilities section (in Chapter 10, "Capabilities"). All constraints are removed for all of the negotiable capabilities supported by the driver.

Application

Set the `pCapability` fields as follows:

```
pCapability->Cap = CAP_SUPPORTEDCAPS
pCapability->ConType = TWON_DONTCARE16
pCapability->hContainer = NULL
```

The Source will not allocate any memory as a part of this call. It will only return a status to indicate success or failure. If this call succeeds then the application must assume that all capabilities have been reset, as well as any DAT structures that are associated with capabilities (such as `DAT_IMAGELAYOUT` or `DAT_JPEGCOMPRESSION`).

Source

The `TW_CAPABILITY` structure has no special meaning for this call. It is not required that the application set the Cap field to `CAP_SUPPORTEDCAPS`, so do not test for it. Do not change the structure in any way. Do not allocate any memory for this call.

When this call is complete the driver should be restored to factory defaults, matching the settings it had when first installed on the user’s machine.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

  TWCC_BADCAP        /* Unknown capability--Source does not  */
                     /* recognize this capability. This code */
                     /* should not be used by sources after  */
                     /* 1.6. Applications still need to test */
                     /* for it for backward compatibility.   */
```

```
TWCC_CAPUNSUPPORTED   /* Capability not supported by   */
                      /* source. Sources 1.6 and newer */
                      /* must use this instead of      */
                      /* using TWCC_BADCAP.  */

TWCC_CAPBADOPERATION  /* Operation not supported by        */
                      /* capability. Sources 1.6 and newer */
                      /* must use this instead of using    */
                      /* TWCC_BADCAP.  */

TWCC_CAPSEQERROR  /* Capability has a dependency on another */
                  /* capability. Sources 1.6 and newer      */
                  /* must use this instead of using         */
                  /* TWCC_BADCAP. */

TWCC_BADDEST      /* No such Source in-session with */
                  /* application */

TWCC_LOWMEMORY    /* Not enough memory to complete the */
                  /* operation */

TWCC_SEQERROR     /* Operation invoked in invalid state */
```

See Also

DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT

Capability Constants (in Chapter 10, "Capabilities")

Capability Containers: TW_ONEVALUE, TW_ENUMERATION, TW_RANGE, TW_ARRAY (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CAPABILITY / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPAB ILITY, MSG_SET,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 (when indicated by `MSG_QUERYSUPPORT`)

5, 6, 7 (when the capability appears in the `CAP_EXTENDEDCAPS` array, and when indicated by `MSG_QUERYSUPPORT`)

Description

Changes the Current Value of the capability to that specified by the application. As of TWAIN 2.2 `MSG_SET` only modifies the Current Value of the specified capability, constraints cannot be changed with `MSG_SET`. The original functionality of `MSG_SET` has been addressed in `MSG_SETCONSTRAINT` for TWAIN 2.2 Sources and higher. (Please refer to `DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT`.)

Application

An application will use the setting of a capability's Current and Available Values differently depending on how the Source was enabled (`DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS`). As of TWAIN 2.2 `MSG_SET` can only change the Current Value, any attempt to change Default or Constraint Values should return `TWRC_CHECKSTATUS` with only the Current Value changed.

If `TW_USERINTERFACE.ShowUI = TRUE`

- In State 4, set the Current Value to be displayed to the user as the current value. This value will be used for acquiring the image unless changed by the user or an automatic process (such as `ICAP_AUTOBRIGHT`).
- In State 6, get the Current Value which was chosen by the user or automatic process. This is the setting used in the upcoming transfer.

If `TW_USERINTERFACE.ShowUI = FALSE`

- In State 4, set the Current Value to the setting that will be used to acquire images (unless automatic settings are set to `TRUE`, for example: `ICAP_AUTOBRIGHT`).
- In State 6, get the Current Value which was chosen by any automatic processes. This is the setting used in the upcoming transfer.

If possible, use the same container type in a `MSG_SET` that the Source returned from a `MSG_GET`. Allocate the container structure. This is where you will place the value(s) you wish to have the Source set. Store the handle into `pCapability->hContainer`. The container must be referenced by a "handle" of type `TW_HANDLE`.

Set the following:

```
pCapability->ConType = TWON_ARRAY, TWON_ONEVALUE, TWON_ENUMERATION, or
                       TWON_RANGE
```

`pCapability->Cap = CAP_xxxx` designator of capability of interest

`pCapability->hContainer = TW_HANDLE` referencing a container of `ConType`

Place the value(s) that you wish the Source to use in the container. If successful, these values will supersede any previous negotiations for this capability.

The application must free the container it allocated when the operation is complete and the application no longer needs to maintain the information.

Source

Return `TWRC_FAILURE` / `TWCC_CAPUNSUPPORTED`:

- If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to). Disregard the operation.

Return `TWRC_FAILURE` / `TWCC_BADVALUE`:

- If the application requests that a value be set that lies outside the supported range of values for the capability (smaller than your minimum value or larger than your maximum value). Set the value to that which most closely approximates the requested value.
- If the application sends a container that you do not support, or do not support in a `MSG_SET`.

Return `TWRC_CHECKSTATUS`:

- If the application requests one or more values that lie within the supported range of values (but that value does not exactly match one of the supported values), set the value to the nearest supported value. The application should then do a `MSG_GET` to check these values.

Return `TWRC_FAILURE` / `TWCC_SEQERROR`:

- If the application sends `MSG_SET` in State 5, 6 or 7 and the capability is not allowed by `CAP_EXTENDEDCAPS`.

If the request is acceptable, use the container structure referenced by `pCapability->hContainer` to set the Current value for the capability.

Return `TWRC_FAILURE` / `TWCC_CAPSEQERROR`:

- If the capability cannot be modified due to a setting for a related capability.

Return Codes

```
TWRC_SUCCESS
TWRC_CHECKSTATUS  /* Capability value(s) could not be matched exactly */
TWRC_FAILURE
```

```
TWCC_BADCAP     /* Source does not recognize this capability. This  */
                /* code should not be used by sources after 1.6.    */
                /* Applications still need to test it for backward  */
                /* compatibility.                                   */

TWCC_CAPUNSUPPORTED      /* Capability not supported by source.     */
                         /* Sources 1.6 and newer must use this.    */

TWCC_CAPBADOPERATION     /* Operation not supported by capability.  */
                         /* Sources 1.6 and newer must use this.    */

TWCC_CAPSEQERROR   /* Capability has a dependency on another    */
                   /* capability Sources 1.6 and newer must use */
                   /* this. */

 TWCC_BADDEST     /* No such Source in-session with application      */

 TWCC_BADVALUE    /* Value outside Source's range for the capability */

 TWCC_SEQERROR    /* Operation invoked in invalid state              */
```

See Also

DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT

Capability Constants (in Chapter 8, "Data Types and Data Structures")

Capability Containers: TW_ONEVALUE, TW_ENUMERATION, TW_RANGE, TW_ARRAY (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CAPABILITY / MSG_SETCONSTRAINT

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CAPABILITY, MSG_SETCONSTRAINT,
pCapability);
```

`pCapability` = A pointer to a `TW_CAPABILITY` structure.

Valid States

4 (when indicated by `MSG_QUERYSUPPORT`)

5, 6, 7 (when the capability appears in the `CAP_EXTENDEDCAPS` array, and when indicated by `MSG_QUERYSUPPORT`)

Description

Changes the Current Value(s) and Available Value(s) of the specified capability to those specified by the application.

Current Values are set when the container is a `TW_ONEVALUE` or `TW_ARRAY`. Available and Current Values are set when the container is a `TW_ENUMERATION`, `TW_ARRAY` or `TW_RANGE`.

Note: Sources are not required to allow restriction of their Available Values, however, this is strongly recommended.

Application

An application will use the setting of a capability's Available Values when the Source was enabled (`DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS`) with `TW_USERINTERFACE.ShowUI = TRUE`.

- In State 4, set the Current Value(s) to be displayed to the user as the current value. This value will be used for acquiring the image unless changed by the user or an automatic process (such as `ICAP_AUTOBRIGHT`).
- In State 4, set the Available Values to restrict the settings displayed to the user and available for use by the Source.
- In State 6, get the Current Value(s) which was chosen by the user or automatic process. This is the setting used in the upcoming transfer.

Store the handle into `pCapability->hContainer`. The container must be referenced by a "handle" of type `TW_HANDLE`.

Set the following:

`pCapability->ConType = TWON_ARRAY, TWON_ONEVALUE, TWON_ENUMERATION, or TWON_RANGE`

`pCapability->Cap = CAP_xxxx` designator of capability of interest

`pCapability->hContainer = TW_HANDLE` referencing a container of `ConType`

Place the value(s) that you wish the Source to use in the container. If successful, these values will supersede any previous negotiations for this capability.

The application must free the container it allocated when the operation is complete and the application no longer needs to maintain the information.

Source

Return `TWRC_FAILURE` / `TWCC_CAPUNSUPPORTED`:

- If the application requests this operation on a capability your Source does not recognize (and you're sure you've implemented all the capabilities that you're required to). Disregard the operation.

Return `TWRC_FAILURE` / `TWCC_BADVALUE`:

- If the application requests that all values be set which are outside the supported values for the capability.
- If the application sends a container that you do not support, or do not support in a `MSG_SETCONSTRAINT`.
- If the application attempts to set the Available Values and the Source does not support restriction of the capability's Available Values.

Returns `TWRC_CHECKSTATUS`:

- If the application requests one or more values that are supported (but all values do not exactly match one of the supported values). The application should then do a `MSG_GET` to check these values.

Return `TWRC_FAILURE` / `TWCC_SEQERROR`:

- If the application sends `MSG_SETCONSTRAINT` in State 5, 6 or 7 and the capability is not allowed by `CAP_EXTENDEDCAPS`.

Return `TWRC_FAILURE` / `TWCC_CAPSEQERROR`:

- If the capability cannot be modified due to a setting for a related capability.

If the request is acceptable, use the container structure referenced by `pCapability->hContainer` to set the Available Values for the capability. If the container type is `TWON_ONEVALUE` set the Current Value for the capability to that value.

If the container type is `TWON_RANGE`, `TWON_ARRAY` or `TWON_ENUMERATION`, set the Current Value for the capability to that value and optionally limit the Available Values for the capability to match those provided by the application, masking all other internal values so that the user cannot select them.

Important: Sources should accommodate requests to limit Available Values. In the interest of adoptability for the breadth of Source manufacturers, such accommodation is not required. It is recommended, however, that the Sources do so, and that the Source's user interface be modified (from its power-on state, and when the user interface is raised) to reflect any limitation of choices implied by the newly negotiated settings.

Note: For example, if an application can only accept black and white image data, it tells the Source of this limitation by doing a `MSG_SET` on `ICAP_PIXELTYPE` with a `TW_ENUMERATION` or `TW_RANGE` container containing only `TWPT_BW` (black and white).

Note: If the Source disregards this negotiated value and fails to modify its user interface, the user may select to acquire a color image. Either the user's selection would fail (for reasons unclear to the user) or the transfer would fail (also for unclear reasons for the user). The Source should strive to prevent such situations.

Return Codes

```
TWRC_SUCCESS

TWRC_CHECKSTATUS  /* Capability value(s) could not be matched exactly */

TWRC_FAILURE

   TWCC_CAPUNSUPPORTED   /* Capability not supported by source.       */

   TWCC_CAPBADOPERATION  /* Operation not supported by capability.    */

   TWCC_CAPSEQERROR      /* Capability has a dependency on another    */
                         /* capability. */

   TWCC_BADDEST         /* No such Source in-session with application */

   TWCC_BADVALUE    /* Value(s) outside Source's range for capability */

   TWCC_SEQERROR    /* Operation invoked in invalid state            */
```

See Also

`DG_CONTROL / DAT_CAPABILITY / MSG_GET`
`DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT`
`DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT`
`DG_CONTROL / DAT_CAPABILITY / MSG_RESET`
`DG_CONTROL / DAT_CAPABILITY / MSG_SET`

Capability Constants (in Chapter 8, "Data Types and Data Structures")

Capability Containers: `TW_ONEVALUE`, `TW_ENUMERATION`, `TW_RANGE`, `TW_ARRAY` (in Chapter 8, "Data Types and Data Structures")

Listing of all capabilities (in Chapter 10, "Capabilities")

DG_CONTROL / DAT_CUSTOMDSDATA / MSG_GET

Call

```
DSM_Entry(
        pOrigin, pDest, DG_CONTROL, DAT_CUSTOMDSDATA,
        MSG_GET, pCustomData
);
```

```
pCustomData = A pointer to a TW_CUSTOMDSDATA structure.
```

Valid States

4 only

Description

This operation is used by the application to query the data source for its current settings, e.g. DPI, paper size, color format. The sources settings will be returned in a `TW_CUSTOMDSDATA` structure. The actual format of the data in this structure is data source dependent and not defined by TWAIN.

Application

`pDest` references the sources identity structure. `pCustomData` points to a `TW_CUSTOMDSDATA` structure.

Source

Fills the `pCustomData` pointer with source specific settings. If supported, `CAP_ENABLEDSUIONLY` and `CAP_CUSTOMDSDATA` *are required.*

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

`TWCC_SEQERROR`

See Also

Capability `CAP_CUSTOMDSDATA`

DG_CONTROL / DAT_CUSTOMDSDATA / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_CUSTOMDSDATA,
MSG_SET, pCustomData);

pCustomData = A pointer to a TW_CUSTOMDSDATA structure.
```

Valid States

4 only

Description

This operation is used by the application to set the current settings for a data source to a previous state as defined by the data contained in the `pCustomData` data structure. The actual format of the data in this structure is data source dependent and not defined by TWAIN.

Application

pDest references the sources identity structure. `pCustomData` points to a `TW_CUSTOMDSDATA` structure.

Source

Changes its current settings to the values specified in the `pCustomData` structure.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

    TWCC_SEQERROR

See Also

Capability `CAP_CUSTOMDSDATA`

DG_CONTROL / DAT_DEVICEEVENT / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_DEVICEEVENT, MSG_GET,
   pSourceDeviceEvent);
```

`pSourceDeviceEvent` = A pointer to a `TW_DEVICEEVENT` structure

Valid States

4 through 7

Description

Upon receiving a `DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT` from the `Source`, the Application must immediately make this call to obtain the event information.

Sources must queue the data for each event so that it is available for this call.

Return Codes

`TWRC_SUCCESS`	
`TWRC_FAILURE`	
`TWCC_BADPROTOCOL`	Capability not supported.
`TWCC_SEQERROR`	No events in the queue, or not in States 4 through 7.

See Also

`DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT (from Source to Application)`
`CAP_DEVICEEVENT`

DG_CONTROL / DAT_ENTRYPOINT / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_ENTRYPPINT, MSG_GET,
pEntryPoint);
```

`pEntryPoint` = A pointer to a `TW_ENTRYPOINT` structure

Valid States

3

Description

A TWAIN 2.0 Application examines the Source's `TW_IDENTITY .SupportedGroups`. If `DF_DSM2` is set, then it must issue this call to get the entry points for the Source Manager. If the conditions are not met then the Source Manager will return `TWRC_FAILURE` / `TWCC_BADPROTOCOL`, and the Application must assume TWAIN 1.x behavior.

The Application gets five entry points in the `TW_ENTRYPOINT` structure:

* the `DSM_Entry` function, this may be ignored

* the `DSM_MemAllocate` function, used by the Application to allocate memory that will be freed by the Source

* the `DSM_MemFree` function, used by the Application to free memory allocated by the Source

* the `DSM_MemLock` function, used by the Application to get a usable pointer from a handle it got from the Source.

* the `DSM_MemUnlock` function, used when the Application is done with the memory it got from the Source. This call is usually made just before `DSM_MemFree`.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

  `TWCC_BADPROTOCOL`

  `TWCC_SEQERROR`

See Also

Identifying TWAIN 2.0 Elements, in Chapter 2, "Technical Overview".

DG_CONTROL / DAT_ENTRYPOINT / MSG_SET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_ENTRYPOINT, MSG_SET,
pEntryPoint);
```

`pEntryPoint` = A pointer to a `TW_ENTRYPOINT` structure

Valid States

The TWAIN 2.0 Source Manager issues this command to Sources (that set DF_DS2) prior of any other command sent by the Application. In most cases it will immediately precede the call to `DG_CONTROL` / `DAT_IDENTITY` / `MSG_OPEN`.

The Source gets five entry points in the `TW_ENTRYPOINT` structure:

- the pointer to the `DSM_Entry` function, used for any `DAT_NULL` operations such as `DG_CONTROL` / `DAT_NULL` / `MSG_XFERREADY`.
- the `DSM_MemAllocate` function, used by the Source to allocate memory that will be freed by the Application
- the `DSM_MemFree` function, used by the Source to free memory allocated by the Application
- *the `DSM_MemLock` function, used by the Source to get a usable pointer from a handle it got from the Application.
- the `DSM_MemUnlock` function, used when the Source is done with the memory it got from the Application. This call is usually made just before `DSM_MemFree`.

Note: TWAIN 1.x Sources must continue to find and load the Source Manager `DSM_Entry` on their own.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

  `TWCC_BADPROTOCOL`

  `TWCC_SEQERROR`

See Also

Identifying TWAIN 2.0 Elements, in Chapter 2, "Technical Overview".

DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT

Windows only; `MSG_PROCESSEVENT` is not available on Mac OS X nor Linux. Refer to Chapter 12, "Operating System Dependencies" for more information.

Call

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

`pEvent` = A pointer to a `TW_EVENT` structure.

Valid States

5 through 7

Description

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

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

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

Application

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

Macintosh: Make `pEvent->pEvent` point to an EventRecord.

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

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

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

MSG_CLOSEDSREQ  /* Source wants to be closed,    */
                /* usually initiated by a            */
                 /* user-generated event              */

MSG_NULL       /* no message for application       */
```

Source

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

Return Codes

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

TWRC_NOTDSEVENT    /* Event belongs to application -  */
                     /* process as usual                  */

TWRC_FAILURE

    TWCC_BADDEST   /* No such Source in-session      */
                     /* with application                */

    TWCC_SEQERROR  /* Operation invoked in invalid   */
                     /* state                             */
```

See Also

DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ (from Source to Application)
DG_CONTROL / DAT_NULL / MSG_XFERREADY (from Source to Application)

Event loop information (in Chapter 7, "Operation Triplets".)

DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_AUTOMATICCAPTUREDIRECTORY, pSourceFileSystem);

pSourceFileSystem = A pointer to a TW_FILESYSTEM structure
```

Valid States

4 only

Description

This operation selects the destination directory within the Source (camera, storage, etc), where images captured using CAP_AUTOMATICCAPTURE will be stored. This command only selects the destination directory (a file of type TWFT_DIRECTORY). The directory must exist and be accessible to the Source. The creation of images within the directory is at the discretion of the Source, and may result in the creation of additional sub-directories.

In all other regards the behavior of this operation is the same as DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY.

If the application does not specify a directory for automatic capture, then the destination of the images is left to the discretion of the Source. A directory named /Images is recommended, but not required.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

- TWCC_BADPROTOCOL - operation not supported.
- TWCC_DENIED - operation denied (device not ready).
- TWCC_FILENOTFOUND - specified InputName does not exist.
- TWCC_SEQERROR - not state 4.

See Also

DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
CAP_AUTOMATICCAPTURE

DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_CHANGEDIRECTORY, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation selects the current device within the Source (camera, storage, etc). If the device is a `TWFT_DOMAIN`, then this command enters a directory that can contain `TWFT_HOST` files. If the device is a `TWFT_HOST`, then this command enters a directory that can contain `TWFT_DIRECTORY` files. If the device is a `TWFT_DIRECTORY`, then this command enters a directory that can contain `TWFT_DIRECTORY` or `TWFT_IMAGE` files.

Sources can support part or all of the storage hierarchy that is one of the following:

```
/Domain/Host/Directory/

/Host/Directory/…

/Directory/…

(Storage not supported)
```

It is permitted to mix domain, host, and directory names in the root file system of the Source. To help resolve any potential name conflict, Applications should set `TW_FILESYSTEM->FileType` to the appropriate value for the topmost file. If this is not done and there is a name conflict, the Source's default behavior must be to use the file of type `TWFT_DIRECTORY` or `TWFT_HOST`, in that order.

For example, consider two files named "abc" in the root of a Source:

```
/abc/123 (abc is a domain)

/abc/789 (abc is a directory)
```

Change directory to the first one by setting FileType to `TWFT_DOMAIN`, or to the second one by setting FileType to `TWFT_DIRECTORY`. The FileType for each will be discovered while browsing the directory using `DAT_GETFILEFIRST` and `DAT_GETFILENEXT`. If the FileType is not specified, then the Source must change to the "/abc/789" directory.

Example:

A Source supports two devices: `/Camera` and `/Disk`. If an application changes directory to /Camera, then it can negotiate imaging parameters and transfer images in a traditional fashion. If an application changes directory to `/Disk/abc/xyz`, then it cannot negotiate imaging parameters (the images have already been captured); all it can do is browse the directory tree and transfer the images it finds.

The Application sets the new current working directory by placing in the InputName field an absolute or relative path. The Source returns the absolute path and name of the new directory in the OutputName field. The special filename dot "." can be used to retrieve the name of the current directory. The special filename dot-dot ".." can be used to change to the parent directory. Refer to the section on File Systems for more information.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

TWCC_BADPROTOCOL - operation not supported.

TWCC_DENIED - operation denied (device not ready).

TWCC_FILENOTFOUND - specified InputName does not exist.

TWCC_SEQERROR - not state 4.

See Also

DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME

DG_CONTROL / DAT_FILESYSTEM / MSG_COPY

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_COPY, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation copies a file or directory. Absolute and relative pathnames are supported. A file may not be overwritten with this command. If an Application wishes to do this, it must first delete the unwanted file and then reissue the Copy command.

The Application specifies the path and name of the entry to be copied in `InputName`. The Application specifies the new patch and name in `OutputName`.

It is not permitted to copy files into the root directory.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED - file cannot be deleted (root file, or protected
                 by Source).

   TWCC_FILEEXISTS - specified OutputName already exists.

   TWCC_FILENOTFOUND - InputName not found or OutputName invalid.

   TWCC_SEQERROR - not state 4.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_CREATEDIRECTORY, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation creates a new directory within the current directory. Pathnames are not allowed, only the name of the new directory can be specified.

Example:

"abc" is valid.
"/Disk/abc" is not valid.

The Application specifies the name of the new directory in InputName.

On success, the Source returns the absolute path and name of the new directory in OutputName.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED - cannot create directory in current directory,
                 directories may not be created in root, or the
                 Source may opt to prevent the creation of new
                 directories in some instances, for instance if
                 the new directory would be too deep in the tree.

   TWCC_FILEEXISTS - the specified InputName already exists.

   TWCC_SEQERROR - not state 4.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_DELETE, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation deletes a file or directory on the device. Pathnames are not allowed, only the name of the file or directory to be deleted can be specified. Recursive deletion can be specified by setting the `pSourceFileSystem->Recursive` to `TRUE`.

Example:

"abc" is valid.
"/Disk/abc" is not valid.

The Application specifies the name of the entry to be deleted in `InputName`. There is no return in `OutputName` on success.

The Application cannot delete entries in the root directory. The Application cannot delete directories unless they are empty.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED - file cannot be deleted (root file, or protected
                 by Source).

   TWCC_FILENOTFOUND - filename not found.

   TWCC_NOTEMPTY - directory is not empty, and cannot be deleted.

   TWCC_SEQERROR - not state 4.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_FORMATMEDIA, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation formats the specified storage. This operation destroys all images and sub-directories under the selected device. Use with caution.

The Application specifies the name of the device to be deleted in `InputName`. There is no data returned by this call.

The Application cannot format the root directory. Sources may opt to protect their media from this command, so Applications must check return and condition codes.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED - format denied (root directory, or protected by Source).

   TWCC_FILENOTFOUND - filename not found.

   TWCC_SEQERROR - not state 4.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_GETCLOSE, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 through 6

Description

The operation frees the `Context` field in `pSourceFileSystem`.

Every call to `DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE` must be matched by a call to `MSG_GETCLOSE` to release the Context field in the `pSourceFileSystem` structure. Note that the .Context value must be preserved between calls.

An Application may (erroneously) issue this operation at any time (even if a `MSG_GETFIRSTFILE` has not been issued yet). Sources must protect themselves from such uses. See the section on File Systems for more information on why and how this must be done.

Return Codes

```
TWRC_SUCCESS
   TWRC_FAILURE
   TWCC_BADPROTOCOL - operation not supported.
   TWCC_BADVALUE - .Context contains an invalid value.
   TWCC_SEQERROR - invalid context calling MSG_GETCLOSE without
                   first calling MSG_GETFIRSTFILE.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_GETFIRSTFILE, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 through 6

Description

This operation gets the first filename in a directory, and returns information about that file (the same information that can be retrieved with `MSG_GETINFO`).

The Source positions the `Context` to point to the first filename. `InputName` is ignored. `OutputName` contains the absolute path and name of the file. Note that the `.Context` value must be preserved between calls.

Applications must not assume any ordering of the files delivered by the Source, with one exception: if `MSG_GETFIRSTFILE` is issued in the root directory, then the operation must return a `TWFT_CAMERA` device.

NB: "." and ".." are NEVER reported by this command.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_BADVALUE - .Context contains an invalid value.

   TWCC_DENIED - file exists, but information about it has not
                 been returned.

   TWCC_FILENOTFOUND - directory is empty.

   TWCC_SEQERROR - called MSG_GETFIRSTFILE again without first calling
                   MSG_GETCLOSE.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_GETINFO, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 through 7

Description

This operation fills the information fields in `pSourceFileSystem`.

`InputName` contains the absolute or relative path and filename of the requested file. `OutputName` returns the absolute path to the file.

Example *InputName*:

"abc" is valid.
"/Disk/abc" is valid.
The empty string "" returns information about the current file (if any).
"." returns information about the current directory.
".." returns information about the parent directory.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED -   - file exists, but information about it has not
                     been returned.

   TWCC_FILENOTFOUND - specified file does not exist.

   TWCC_SEQERROR - not state 4 - 7, or no current file.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_GETNEXTFILE, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 through 6

Description

This operation gets the next filename in a directory, and returns information about that file (the same information that can be retrieved with `MSG_GETINFO`).

The Source positions the Context to point to the next filename. `InputName` is ignored. `OutputName` contains the absolute path and name of the file. Note that the .Context value must be preserved between calls.

A call to `MSG_GETFIRSTFILE` must be issued on a given directory before the first call to `MSG_GETNEXTFILE`.

NB: The "." and ".." entries are NEVER reported by this command

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_BADVALUE - .Context contains an invalid value.

   TWCC_DENIED - file exists, but information about it has not
                 been returned.

   TWCC_FILENOTFOUND - directory is empty.

   TWCC_SEQERROR - invalid context calling MSG_GETNEXTFILE without
                   first calling MSG_GETFIRSTFILE.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_FILESYSTEM,
   MSG_RENAME, pSourceFileSystem);
```

`pSourceFileSystem` = A pointer to a `TW_FILESYSTEM` structure

Valid States

4 only

Description

This operation renames (and optionally moves) a file or directory. Absolute and relative path names are supported. A file may not be overwritten with this command. If an Application wishes to do this it must first delete the unwanted file, then issue the rename command.

The Application specifies the path and name of the entry to be renamed in InputName. The Application specifies the new path and name in OutputName.

Filenames in the root directory cannot be moved or renamed.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL - operation not supported.

   TWCC_DENIED - file cannot be deleted (root file, or protected
                 by Source).

   TWCC_FILEEXISTS - specified OutputName already exists.

   TWCC_FILENOTFOUND - InputName not found or OutputName invalid.

   TWCC_SEQERROR - not state 4.
```

See Also

```
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
```

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

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_CLOSEDS,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

4 only (Transitions to State 3, if successful)

Description

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

Application

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

Source Manager

Passes the message onto the Source as

```
DSM_Entry(pOrigin, DG_CONTROL, DAT_IDENTITY, MSG_CLOSEDS,
pSourceIdentity);
```

Following receipt of `TWRC_SUCCESS` from the Source, Closes the Source. If the Source has no more connections removes it from memory.

Return Codes

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

See Also

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

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

Call

```
DS_Entry(pOrigin, DG_CONTROL, DAT_IDENTITY, MSG_CLOSEDS,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

4 only (Transitions Source back to the "loaded but not open" State - approximately State 3.5)

Description

Closes the Source so it can be unloaded from memory. The Source responds by doing its shutdown and clean-up activities needed to ensure the heap will be "clean" after the Source is unloaded. Under Windows, the Source will only be unloaded if the connection with the last application accessing it is about to be broken. The Source will know this by its internal "connect count" that should be maintained by any Source that supports multiple application connects.

Source Manager

`pSourceIdentity` is filled from a previous `MSG_OPENDS` operation.

Source

Perform all necessary housekeeping in anticipation of being unloaded. Be sure to dispose of any memory buffers that the Source has allocated locally, or that may have become the Source's responsibility during the course of the TWAIN session. The Source exists in a shared memory environment. It is therefore critical that all remnants of the Source, except the entry point (initial) code, be removed as the Source prepares to be unloaded.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_OPERATIONERROR   /* Internal Source error; */
                         /* handled by the Source  */
```

See Also

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

DG_CONTROL / DAT_IDENTITY / MSG_GET (from Source Manager to Source)

Call

```
DS_Entry(pOrigin, DG_CONTROL, DAT_IDENTITY, MSG_GET, pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

3 through 7 (Yes, the Source must be able to return the identity *before* it is opened.)

Description

This operation triplet is generated only by the Source Manager and is sent to the Source. It returns the identity structure for the Source.

Source Manager

No special set up or action required.

Source

Fills in all fields of pSourceIdentity except the Id field which is only modified by the Source Manager. This structure was allocated by either the application or the Source Manager depending on which one initiated the `MSG_OPENDS` operation for the Source.

Note: Sources should locate the code that handles initialization of the Source (responding to `MSG_OPENDS`) and identification (`DAT_IDENTITY / MSG_GET`) in the segment first loaded when the DLL/code resource is invoked. Responding to the identification operation should not cause any other segments to be loaded. Code to handle all other operations and to support the user interface should be located in code segments that will be loaded upon demand. Remember, the Source is a "guest" of the application and needs to be sensitive to use of available memory and other system resources. The Source Manager's perceived performance may be adversely affected unless the Source efficiently handles identification requests.

Return Codes

```
TWRC_SUCCESS     /* This operation must succeed. */
```

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_GETDEFAULT,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

3 through 7

Description

Gets the identification information of the system default Source.

Application

No special set up or action required.

Source Manager

Fills the structure pointed to by `pSourceIdentity` with identifying information about the system default Source.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

```
   TWCC_NODS          /* no Sources found matching       */
                      /* application's SupportedGroups   */

   TWCC_LOWMEMORY     /* not enough memory to perform    */
                      /* this operation                  */
```

See Also

```
DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST
DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Application to Source
Manager)
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT
```

DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_GETFIRST,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

3 through 7

Description

The application may obtain a list of all Sources that are currently available on the system which match the application's supported groups (DGs, that the application specified in the SupportedGroups field of its `TW_IDENTITY` structure). To obtain the complete list of all available Sources requires invocation of a series of operations. The first operation uses `MSG_GETFIRST` to find the first Source on "the list" (whichever Source the Source Manager finds first). All the following operations use `DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT` to get the identity information, one at a time, of all remaining Sources.

Note: The application must invoke the `MSG_GETFIRST` operation before a `MSG_GETNEXT` operation. If the `MSG_GETNEXT` is invoked first, the Source Manager will fail the operation (`TWRC_ENDOFLIST`).

If the application wants to cause a specific Source to be opened, one whose ProductName the application knows, it must first establish the existence of the Source using the `MSG_GETFIRST/ MSG_GETNEXT` operations. Once the application has verified that the Source is available, it can request that the Source Manager open the Source using `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`. The application must not execute this operation without first verifying the existence of the Source because the results may be unpredictable.

Application

No special set up or action required.

Source Manager

Fills the `TW_IDENTITY` structure pointed to by `pSourceIdentity` with the identity information of the first Source found by the Source Manager within the TWAIN directory/folder.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

```
   TWCC_NODS           /* No Sources can be found     */
```

```
          TWCC_LOWMEMORY      /* Not enough memory to perform */
                             /* this operation              */
```

See Also

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT
DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Application to Source Manager)
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT

DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_GETNEXT,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

3 through 7

Description

The application may obtain a list of all Sources that are currently available on the system which match the application's supported groups (DGs, that the application specified in the SupportedGroups field of its `TW_IDENTITY` structure). To obtain the complete list of all available Sources requires invocation of a series of operations. The first operation uses `DG_CONTROL` / `DAT_IDENTITY` / `MSG_GETFIRST` to find the first Source on "the list" (whichever Source the Source Manager finds first). All the following operations use `MSG_GET`NEXT to get the identity information, one at a time, of all remaining Sources.

Note: The application must invoke the `MSG_GETFIRST` operation before a `MSG_GETNEXT` operation. If the `MSG_GETNEXT` is invoked first, the Source Manager will fail the operation (`TWRC_ENDOFLIST`).

If the application wants to cause a specific Source to be opened, one whose ProductName the application knows, it must first establish the existence of the Source using the `MSG_GETFIRST/` `MSG_GETNEXT` operations. Once the application has verified that the Source is available, it can request that the Source Manager open the Source using `DG_CONTROL` / `DAT_IDENTITY` / `MSG_OPENDS`. The application must not execute this operation without first verifying the existence of the Source because the results may be unpredictable.

Application

No special set up or action required.

Source Manager

Fills the `TW_IDENTITY` structure pointed to by pSourceIdentity with the identity information of the next Source found by the Source Manager within the TWAIN directory/folder.

Return Codes

```
TWRC_SUCCESS

TWRC_ENDOFLIST        /* after MSG_GETNEXT if no more */
                         /* Sources                          */

TWRC_FAILURE
```

```
          TWCC_LOWMEMORY     /* not enough memory to perform */
                                /* this operation               */
```

See Also

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT
DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Application to Source Manager)
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT

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

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_OPENDS,
pSourceIdentity);
```

pSourceIdentity = A pointer to a TW_IDENTITY structure.

Valid States

3 only (Transitions to State 4, if successful)

Description

Loads the specified Source into main memory and causes its initialization.

Application

The application may specify any available Source's TW_IDENTITY structure in pSourceIdentity. That structure may have been obtained using a MSG_GETFIRST, MSG_GETNEXT, or MSG_USERSELECT operation. If the session with the Source Manager was closed since the identity structure being used was obtained, the application must set the Id field to 0. This will cause the Source Manager to issue the Source a new Id. The application can have the Source Manager open the default Source by setting the ProductName field to "\0" (Null string) and the Id field to zero.

Source Manager

Opens the Source specified by pSourceIdentity and creates a unique Id value for this Source (under Microsoft Windows, this assumes that the Source hadn't already been opened by another application). This value is recorded in pSourceIdentity->Id. The Source Manager passes the triplet on to the Source to have the remaining fields in pSourceIdentity filled in.

Upon receiving the request from the Source Manager, the Source fills in all the fields in pSourceIdentity except for Id. If an application tries to connect to a Source that is already connected to its maximum number of applications, the Source returns TWRC_FAILURE/ TWCC_MAXCONNECTIONS.

Warning: The Source and application **must** not assume that the value written into pSourceIdentity.Id will remain constant between sessions. This value is used internally by the Source Manager to uniquely identify applications and Sources and to manage the connections between them. During a different session, this value may still be valid but might be assigned to a different application or Source! Don't use this value directly.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

```
TWCC_LOWMEMORY        /* not enough memory to */
                        /* open the Source      */
TWCC_MAXCONNECTIONS   /* Source cannot support*/
                        /* another connection   */
TWCC_NODS             /* specified Source was */
                        /* not found            */
TWCC_OPERATIONERROR   /* internal Source error;*/
                        /* handled by the Source */
```

See Also

DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS (from Application to Source Manager)
DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_GET (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT
DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST
DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT
DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT

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

Call

```
DS_Entry(pOrigin, DG_CONTROL, DAT_IDENTITY, MSG_OPENDS,
pSourceIdentity);
```

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

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

Description

Opens the Source for operation.

Source Manager

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

Source

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

Refer to Chapter 12, "Operating System Dependencies" for more information on `MSG_OPENDS`.

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

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE
   TWCC_LOWMEMORY        /* not enough memory to  */
                           /* open the Source       */
   TWCC_MAXCONNECTIONS     /* Source cannot support */
                           /* another connection    */
   TWCC_OPERATIONERROR     /* internal Source error;*/
                           /* handled by the Source */
```

See Also

`DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS (from Source Manager to Source)`
`DG_CONTROL / DAT_IDENTITY / MSG_GET (from Source Manager to Source)`

DG_CONTROL / DAT_IDENTITY / MSG_SET

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_IDENTITY, MSG_SET,
pTwIdentity);
```

`_pTwIdentity` = A pointer to a `TW_IDENTITY` structure containing a valid `TW_IDENTITY` for a Data source.

Valid States

3

Description

This operation triplet is generated by the application and is consumed by the Data Source Manager. It allows an application to set the default TWAIN driver, which is reported back by DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT.

Application

The application must specify an available Source's TW_IDENTITY structure in pTwIdentity. That structure must have been obtained using a MSG_GETFIRST, MSG_GETNEXT, or MSG_USERSELECT operation since the Source Manager was last opened.

Source Manager

Sets a new default TWAIN driver.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

     TWCC_BADVALUE    /* Invalid DS in TW_IDENTITY */
```

See Also

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT

DG_CONTROL / DAT_IDENTITY / MSG_USERSELECT

Windows and Macintosh only; `MSG_USERSELECT` is not available on Linux. Refer to Chapter 12, "Operating System Dependencies".

Call

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

`pSourceIdentity` = A pointer to a `TW_IDENTITY` structure.

Valid States

3 through 7

Description

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

Application

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

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

Source Manager

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

The "Select" button ("OK" button in the Macintosh Source Manager) will be grayed out if there are no Sources available matching the SupportedGroups specified in the application's identity structure, pOrigin. The user must click the "Cancel" button to exit the Select Source dialog. The application cannot discern from this Return Code whether the user simply canceled the selection or there were no Sources for the user to select. If the application really wants to know whether any Sources are available that match the specified SupportedGroups it can invoke a `MSG_GETFIRST` operation and check for a successful result.

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

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

Return Codes

```
TWRC_SUCCESS

TWRC_CANCEL         /* User clicked cancel button - maybe there   */
                      /* were no Sources                            */

TWRC_FAILURE

   TWCC_LOWMEMORY   /* not enough memory to perform this  */
                       /* operation                         */
```

See Also

DG_CONTROL / DAT_IDENTITY / MSG_GETDEFAULT
DG_CONTROL / DAT_IDENTITY / MSG_GETFIRST
DG_CONTROL / DAT_IDENTITY / MSG_GETNEXT
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Application to Source Manager)
DG_CONTROL / DAT_IDENTITY / MSG_OPENDS (from Source Manager to Source)
DG_CONTROL / DAT_IDENTITY / MSG_SET

DG_CONTROL / DAT_METRICS / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_METRICS, MSG_GET, pMetrics);
```

`pMetrics` = A pointer to a `TW_METRICS` structure.

Valid States

4 only

Description

Reads information relating to the last time `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` was sent. An application calls this to get final counts after scanning. This is necessary because some metrics cannot be detected during scanning, such as blank images discarded at the very end of a session.

In this example `ICAP_AUTODISCARDBLANKPAGES` is on, so the scanner will not offer some images for transfer if it determines that there's no content on the paper.

Physical Sheet	Image Count	TWEI_PAPERCOUNT	Note
1	1	1	Transferred
1	--	--	Blank
2	2	2	Transferred
2	3	2	Transferred
3	--	--	Blank
3	--	--	Blank
4	4	4	Transferred
4	4	4	Transferred
5	--	--	Blank
5	--	--	Blank

Five sheets of paper were processed by the scanner, but only five images were transferred to the application. The application is able to detect the gaps for sheets 1 – 4, but since sheet 5 is the last one captured there's no evidence that it was skipped.

In this example if the application calls `DG_CONTROL / DAT_METRICS / MSG_GET` in state 4, after scanning is complete, they can see that `TW_METRICS.SheetCount` is set to 5, and determine that the last sheet of paper was discarded.

Source

The source only processes fields that fit within the range of `TW_METRICS.SizeOf` (this is done in case new fields are added in the future).

The call reports metrics gathered since the last time `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` was issued. If the call has never been issued then the fields in the structure return zero

Application

The application sets `TW_METRICS.SizeOf` to the size of the `TW_METRICS` structure (this is done in case new fields are added in the future).

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

`TWCC_BADVALUE`

`TWCC_SEQERROR`

See Also

`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

DG_CONTROL / DAT_NULL / MSG_CLOSEDSOK (from Source to Application)

For Macintosh OS X 1.9 data sources, refer to the TWAIN 1.9 specification.

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

The Source sends this message to the application to indicate that the Source needs to be closed.

Source

Source creates this triplet with NULL data and sends it to the Source Manager via the Source Manager's DSM_Entry point.

Source Manager

Upon receiving this triplet, the Source Manager passes this message to the application either using the applications callback function or by posts a private message to the application's event/ message loop.

Application

The Application will either receive this message in its callback function or as an event in its event loop.

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_NULL / MSG_CLOSEDSREQ (from Source to Application)

For Macintosh OS X 1.9 data sources, refer to the TWAIN 1.9 specification.

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_CLOSEDSREQ, NULL);
```

This operation requires no data (`NULL`).

Valid States

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

Description

The Source sends this message to the application to indicate that the Source needs to be closed.

Source

Source creates this triplet with NULL data and sends it to the Source Manager via the Source Manager's DSM_Entry point.

Source Manager

Upon receiving this triplet, the Source Manager passes this message to the application either using the applications callback function or by posts a private message to the application's event/ message loop.

Application

The Application will either receive this message in its callback function or as an event in its event loop.

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_CALLBACK / MSG_REGISTER_CALLBACK`
`DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT (from Source to Application)

Call

`DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_DEVICEEVENT, NULL);`

This operation requires no data (`NULL`)

Valid States

4 through 7

Description

When enabled the source sends this message to the Application to alert it that some event has taken place. Upon receiving this message, the Application must immediately issue a call to `DG_CONTROL` / `DAT_DEVICEEVENT` / `MSG_GET` to obtain the event information.

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_DEVICEEVENT / MSG_GET`

Capability - `CAP_DEVICEEVENT`

DG_CONTROL / DAT_NULL / MSG_XFERREADY (from Source to Application)

For Macintosh OS X 1.9 data sources, refer to the TWAIN 1.9 specification.

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_NULL, MSG_XFERREADY, NULL);
```

This operation requires no data (`NULL`).

Valid States

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

Description

The Source sends this message to the application to indicate that the Source has data that is ready to be transferred.

Source

Source creates this triplet with `NULL` data and sends it to the Source Manager via the Source Manager's `DSM_Entry` point.

Source Manager

Upon receiving this triplet, the Source Manager passes this message to the application either using the applications callback function or by posts a private message to the application's event/ message loop.

Application

The Application will either receive this message in its callback function or as an event in its event loop.

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_CALLBACK / MSG_REGISTER_CALLBACK
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET
```

DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM

Call

`DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_PARENT, MSG_CLOSEDSM, pParent);`
`pParent` = the same `pParent` used for `OPENDSM`.

Valid States

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

Description

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

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

Application

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

Source Manager

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

If the Source Manager has been opened multiple times it will remain active and connected to the other connection(s).

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_SEQERROR      /* Operation invoked in invalid state */
```

See Also

`DG_CONTROL / DAT_PARENT / MSG_OPENDSM`

DG_CONTROL / DAT_PARENT / MSG_OPENDSM

Call

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

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

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

Valid States

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

Description

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

Application

The application must allocate a structure of type `TW_IDENTITY` and fill in all fields. The Id field must be `NULL`. Once the structure is prepared, this `pOrigin` parameter should point at that structure. If the Source Manager is opened successfully it will assign a value to Id.

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

Windows — Set `pParent` to point to a window handle (`hWnd`) of an open window that will remain open until the Source Manager is closed.

Macintosh —Set `pParent` to `NULL`.

Linux —Set `pParent` to `NULL`.

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.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_LOWMEMORY      /* not enough memory to perform */
                       /* this operation               */

   TWCC_SEQERROR       /* Operation invoked in invalid */
                       /* state                        */
```

See Also

`DG_CONTROL / DAT_PARENT / MSG_CLOSEDSM`

DG_CONTROL / DAT_PASSTHRU / MSG_PASSTHRU

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_PASSTHRU,MSG_PASSTHRU,
   pSourcePassthru);
```

`pSourcePassthru` = A pointer to a `TW_PASSTHRU` structure

Valid States

4 through 7

Description

`PASSTHRU` is intended for the use of Source writers writing diagnostic applications. It allows raw communication with the currently selected device in the Source.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

```
   TWCC_BADPROTOCOL - capability not supported.
   TWCC_SEQERROR - command could not be completed in this state.
```

See Also

`CAP_PASSTHRU`

DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_PENDINGXFERS, MSG_ENDXFER,
   pPendingXfers);
```

`pPendingXfers` = A pointer to a `TW_PENDINGXFERS` structure

Valid States

6 and 7

When `DAT_XFERGROUP` is set to `DG_IMAGE`:

(Transitions to State 5 if this was the last transfer (`pPendingXfers->Count == 0`).
Transitions to State 6 if there are more transfers pending (`pPendingXfers->Count != 0`).
To abort all remaining transfers and transition from State 6 to State 5, use `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_RESET`.

When `DAT_XFERGROUP` is set to `DG_AUDIO`:

Transitions to State 6 no matter what the value of `pPendingXfers->Count`.

Description

This triplet is used to cancel or terminate a transfer. Issued in state 6, this triplet cancels the next pending transfer, discards the transfer data, and decrements the pending transfers count. In state 7, this triplet terminates the current transfer. If any data has not been transferred (this is only possible during a memory transfer) that data is discarded.

The application can use this operation to cancel the next pending transfer (Source writers take note of this). For example, after the application checks `TW_IMAGEINFO` (or `TW_AUDIOINFO`, if transferring audio snippets), it may decide to not transfer the next image. The operation must be sent prior to the beginning of the transfer, otherwise the Source will simply abort the current transfer. The Source decrements the number of pending transfers.

Application

The application must invoke this operation at the end of every transfer to signal the Source that the application has received all the data it expected. The application should send this after receiving a `TWRC_XFERDONE` or `TWRC_CANCEL`.

No special set up or action required. Be sure to correctly track which state the Source will be in as a result of your action. Be aware of the value in `pPendingXfers->Count` both before and after the operation. Invoking this operation causes the loss of data that your user may not expect to be lost. Be very careful and prudent when using this operation.

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.

Source

Option #1) Fill pPendingXfers->Count with the number of transfers the Source is ready to supply to the application, upon demand. If pPendingXfers->Count > 0 (or equals -1), transition to State 6 and await initiation of the next transfer by the application. If pPendingXfers->Count == 0, transition all the way back to State 5 and await the next acquisition.

Option #2) Preempt the acquired data that is next in line for transfer to the application (pending transfers can be thought of as being pushed onto a FIFO queue as acquired and popped off the queue when transferred). Decrement pPendingXfers->Count. If already acquired, discard the data for the preempted transfer. Update pPendingXfers->Count with the new number of pending transfers. If this value is indeterminate, leave the value in this field at -1. Note: -1 is not a valid value for the number of audio snippets.

Option #3) Cancel the current transfer. Discard any local buffers or data involved in the transfer. Prepare the Source and the device for the next transfer. Decrement pPendingXfers->Count (don't decrement if already zero or -1). If there is a transfer pending, return to State 6 and prepare the Source to begin the next transfer. If no transfer is pending, return to State 5 and await initiation of the next acquisition from the application or the user. Note: when DAT_XFERGROUP is set to DG_AUDIO, the Source will not go lower than State 6 based on the value of pPendingXfers->Count.

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.

Note: If a Source supports simultaneous connections to more than one application, the Source should maintain a separate pPendingXfers structure for each application it is in-session with.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

```
TWCC_BADDEST  /* No such Source in-session with application */
TWCC_SEQERROR /* Operation invoked in invalid state         */
```

See Also

DG_AUDIO / DAT_AUDIOFILEXFER / MSG_GET
DG_AUDIO / DAT_AUDIONATIVEXFER / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET
DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER
DG_CONTROL / DAT_XFERGROUP / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

Capability - CAP_XFERCOUNT

DG_CONTROL / DAT_PENDINGXFERS / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_PENDINGXFERS,
   MSG_GET, pPendingXfers);
```

`pPendingXfers` = A pointer to a `TW_PENDINGXFERS` structure

Valid States

4 through 7

Description

Returns the number of transfers the Source is ready to supply to the application, upon demand. If `DAT_XFERGROUP` is set to `DG_IMAGE`, this is the number of images. If `DAT_XFERGROUP` is set to `DG_AUDIO`, this is the number of audio snippets for the current image. If there is no current image, this call must return `TWRC_FAILURE` / `TWCC_SEQERROR`.

Application

No special set up or action required.

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.

Source

Fill `pPendingXfers->Count` with the number of transfers the Source is ready to supply to the application, upon demand. This value should reflect the number of complete data blocks that have already been acquired or are in the process of being acquired.

When `CAP_JOBCONTROL` is set to other than `TWJC_NONE` then `pPendingXfers->EOJ` should reflect the current `TWEJ_xxx` Job control value.

When `DAT_XFERGROUP` is set to `DG_IMAGE`:

> If the Source is not sure how many transfers are pending, but is sure that the number is at least one, set `pPendingXfers->Count` to -1. A Source connected to a device with an automatic document feeder that cannot determine the number of pages in the feeder, or how many selections the user may make on each page, would respond in this way. A Source providing access to a series of images from a video camera or a data base may also respond this way.

When `DAT_XFERGROUP` is set to `DG_AUDIO`:

> -1 is not a valid value for `pPendingXfers->Count.`

Return Codes

`TWRC_SUCCESS`

```
TWRC_FAILURE
   TWCC_BADDEST     /* No such Source in-session with application */
   TWCC_SEQERROR  /* Operation invoked in invalid state         */
```

See Also

DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET
DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER
DG_CONTROL / DAT_XFERGROUP / MSG_SET

Capability - CAP_XFERCOUNT

DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_PENDINGXFERS,
   MSG_RESET, pPendingXfers);
```

`pPendingXfers` = A pointer to a `TW_PENDINGXFERS` structure

Valid States

When `DAT_XFERGROUP` is set to `DG_IMAGE`:

6 only (Transitions to State 5, if successful)

When `DAT_XFERGROUP` is set to `DG_AUDIO`:

6 only (State remains at 6)

Description

Sets the number of pending transfers in the Source to zero.

Application

When `DAT_XFERGROUP` is set to `DG_IMAGE`:

No special set up or action required. Be aware of the state transition caused by this operation. Invoking this operation causes the loss of data that your user may not expect to be lost. Be very careful and prudent when using this operation. The application may need to use this operation if an error occurs within the application that necessitates breaking off all TWAIN sessions. This will get the application, Source Manager, and Source back to State 5 together.

When `DAT_XFERGROUP` is set to `DG_AUDIO`:

The available audio snippets are discarded, but the Source remains in State 6.

Source

Set `pPendingXfers->Count` to zero. Discard any local buffers or data involved in any of the pending transfers.

When `DAT_XFERGROUP` is set to `DG_IMAGE`:

Return to State 5 and await initiation of the next acquisition from the application or the user.

When `DAT_XFERGROUP` is set to `DG_AUDIO`:

Remain in State 6.

Note: If a Source supports simultaneous sessions with more than one application, the Source should maintain a separate pPendingXfers structure for each application it is in-session with.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST   /* No such Source in-session with application */

   TWCC_SEQERROR  /* Operation invoked in invalid state         */
```

See Also

DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER
DG_CONTROL / DAT_XFERGROUP / MSG_SET

Capability - CAP_XFERCOUNT

DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_PENDINGXFERS,
     MSG_STOPFEEDER, pPendingXfers);
```

pPendingXfers = A pointer to a TW_PENDINGXFERS structure

Valid States

6 only

Description

If CAP_AUTOSCAN is TRUE, this command will stop the operation of the scanner's automatic feeder. No other action is taken.

Application

The DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET command stops a session (returning to State 5), but it also discards any images that have been captured by the scanner. The MSG_STOPFEEDER command solves this problem by stopping the feeder, but remaining in State 6. The application may then continue to transfer images, until pPendingXfers->Count goes to zero.

Source

This command should only perform successfully if CAP_AUTOSCAN is TRUE. If CAP_AUTOSCAN is FALSE, this command should return TWRC_FAILURE / TWCC_SEQERROR.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

```
TWCC_BADDEST – no such Source in session with application.
TWCC_BADPROTOCOL - Source does not support operation.
TWCC_SEQERROR - Operation invoked in invalid state.
```

See Also

DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET

Capabilities - CAP_AUTOSCAN

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
   MSG_GET, pSetupFile);
```

`pSetupFile` = A pointer to a `TW_SETUPFILEXFER` structure

Valid States

4 through 6

Description

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

Application

No special set up or action required.

Source

Set the following:

```
pSetupFile->Format = format of destination file
       (DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
       (DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
```

`pSetupFile->FileName` = name of file

Windows: include the complete path name

Macintosh: filename only

Linux: include the complete path name

`pSetupFile->VRefNum` = volume reference number

Windows: not used. Set to `TWON_DONTCARE16`.

Macintosh: Set to the `FSVolumeRefNum` of the folder of the file.

Linux: not used. Set to `TWON_DONTCARE16`.

Return Codes

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

See Also

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET

Capabilities - ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
   MSG_GETDEFAULT, pSetupFile);
```

`pSetupFile` = A pointer to a `TW_SETUPFILEXFER` structure

Valid States

4 through 6

Description

Returns information for the default `DG_IMAGE` or `DG_AUDIO` file.

Application

No special set up or action required.

Source

Set the following:

`pSetupFile->Format` = format of destination file

(`DG_IMAGE` Constants: `TWFF_TIFF`, `TWFF_PICT`, `TWFF_BMP`, etc.)

(`DG_AUDIO` Constants: `TWAF_WAV`, `TWAF_AIFF`, `TWAF_AU`, etc.)

`pSetupFile->FileName` = name of file

Windows: include the complete path name

Macintosh: filename only

Linux: include the complete path name

`pSetupFile->VRefNum` = volume reference number

Windows: not used. Set to `TWON_DONTCARE16`.

Macintosh: Set to the FSVolumeRefNum of the folder of the file.

Linux: not used. Set to `TWON_DONTCARE16`.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST     /* No such Source in-session with application */

   TWCC_BADPROTOCOL /* Source does not support file transfer      */

   TWCC_SEQERROR    /* Operation invoked in invalid state         */
```

See Also

```
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
```

Capabilities - `ICAP_XFERMECH`, `ICAP_IMAGEFILEFORMAT`, `ACAP_XFERMECH`

DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
   MSG_RESET, pSetupFile);
```

`pSetupFile` = A pointer to a `TW_SETUPFILEXFER` structure

Valid States

4 only

Description

Resets the current file information to the `DG_IMAGE` or `DG_AUDIO` default file information and returns that default information..

Application

No special set up or action required.

Source

Set the following:

`pSetupFile->Format` = format of destination file

(`DG_IMAGE` Constants: `TWFF_TIFF`, `TWFF_PICT`, `TWFF_BMP`, etc.)

(`DG_AUDIO` Constants: `TWAF_WAV`, `TWAF_AIFF`, `TWAF_AU`, etc.)

`pSetupFile->FileName = name of file`

`Windows``:  include the complete path name`

`Macintosh``: filename only`

`Linux``:  include the complete path name`

`pSetupFile->VRefNum = volume reference number`

`Windows``:  not used.  Set to TWON_DONTCARE16.`

`Macintosh``:  Set to the FSVolumeRefNum to reflect the default file only if it already exists. Otherwise, set this field to NULL.`

`Linux``:  not used.  Set to TWON_DONTCARE16.`

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
```

See Also

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET

Capabilities - ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_SETUPFILEXFER,
   MSG_SET, pSetupFile);
```

`pSetupFile` = A pointer to a `TW_SETUPFILEXFER` structure

Valid States

4 through 6

Description

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

Application

Set the following:

```
pSetupFile->Format = format of destination file
        (DG_IMAGE Constants: TWFF_TIFF, TWFF_PICT, TWFF_BMP, etc.)
        (DG_AUDIO Constants: TWAF_WAV, TWAF_AIFF, TWAF_AU, etc.)
pSetupFile->FileName = name of file
        Windows:  include the complete path name
        Macintosh: filename only
        Linux:  include the complete path name
pSetupFile->VRefNum = volume reference number
        Windows:  not used.  Set to TWON_DONTCARE16.
        Macintosh:  Set to the FSVolumeRefNum to reflect the default
          file only if it already exists. Otherwise, set this field to
          NULL.
        Linux:  not used.  Set to TWON_DONTCARE16.
```

Note: `ICAP_XFERMECH` or `ACAP_XFERMECH` (depending on the value of `DAT_XFERGROUP`) must have been set to `TWSX`data) and return `TWRC_FAILURE` with `TWCC_BADVALUE`. If the format and file name are OK, but a file error occurs when trying to open the file (other than "file does not exist"), return `TWCC_BADVALUE` and set up to use the default file. If the specified file does not exit, create it. If the file exists and has data in it, overwrite the existing data starting with the first byte of the file.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST     /* No such Source in-session with application */

   TWCC_BADPROTOCOL /* Source does not support file transfer      */

   TWCC_BADVALUE    /* Source cannot comply with one of the       */

                    /* settings                                   */

   TWCC_SEQERROR    /* Operation invoked in invalid state         */

   /* The following introduced for 2.0 or higher */

   TWCC_FILEWRITEERROR
```

See Also

DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GETDEFAULT
DG_CONTROL / DAT_SETUPFILEXFER / MSG_RESET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET

Capabilities - ICAP_XFERMECH, ICAP_IMAGEFILEFORMAT, ACAP_XFERMECH

DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_SETUPMEMXFER, MSG_GET,
pSetupMem);
```

`pSetupMem` = A pointer to a `TW_SETUPMEMXFER` structure.

Valid States

4 through 6

Description

Returns the Source's preferred, minimum, and maximum allocation sizes for transfer memory buffers. The application using buffered memory transfers must use a buffer size between MinBufSize and MaxBufSize in their `TW_IMAGEMEMXFER`.Memory.Length when using the `DG_IMAGE` / `DAT_IMAGEMEMXFER` / `MSG_GET` operation. Sources may return a more efficient preferred value in State 6 after the image size, etc. has been specified.

Application

No special set up or action required.

Source

Set the following:

pSetupMem->MinBufSize = minimum usable buffer size,
in bytes

pSetupMem->MaxBufSize = maximum usable buffer size,
in bytes (-1 means an indeterminately large buffer is acceptable)

pSetupMem->Preferred = preferred transfer buffer size, in bytes

If the Source doesn't care about the size of any of these specifications, set the field(s) to `TWON_DONTCARE32`. This signals the application that any value for that field is OK with the Source.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

```
TWCC_BADDEST     /* No such Source in-session with */
                 /* application                    */

TWCC_SEQERROR    /* Operation invoked in invalid   */
                 /* state                          */
```

See Also

DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET

Capabilities - ICAP_COMPRESSION, ICAP_XFERMECH

DG_CONTROL / DAT_STATUS / MSG_GET (from Application to Source Manager)

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_STATUS, MSG_GET,
pSourceStatus);
```

`pSourceStatus` = A pointer to a `TW_STATUS` structure.

Valid States

2 through 7

Description

Returns the current Condition Code for the Source Manager.

Application

`NULL` references the operation to the Source Manager.

Source Manager

Fills `pSourceStatus->ConditionCode` with its current Condition Code. Then, it will clear its internal Condition Code so you cannot issue a status inquiry twice for the same error (the information is lost after the first request).

Return Codes

```
TWRC_SUCCESS       /* This operation must succeed     */

TWRC_FAILURE

   TWCC_BADDEST    /* No such Source in-session with */
                   /* application                    */
```

See Also

Return Codes and Condition Codes (Chapter 11, "Return Codes and Condition Codes")

DG_CONTROL / DAT_STATUS / MSG_GET (from Application to Source)

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_STATUS, MSG_GET,
pSourceStatus);
```

`pSourceStatus` = A pointer to a `TW_STATUS` structure.

Valid States

4 through 7

Description

Returns the current Condition Code for the specified Source.

Application

`pDest` references a copy of the targeted Source's identity structure.

Source

Fills `pSourceStatus->ConditionCode` with its current Condition Code. Then, it will clear its internal Condition Code so you cannot issue a status inquiry twice for the same error (the information is lost after the first request).

Fills `pSourceStatus->Data` with its current custom code. If there is no custom code, the value must be 0.

Return Codes

```
TWRC_SUCCESS        /* This operation must succeed     */
TWRC_FAILURE
   TWCC_BADDEST     /* No such Source in-session with */
                    /* application                    */
```

See Also

Return Codes and Condition Codes (Chapter 11, "Return Codes and Condition Codes")

DG_CONTROL / DAT_STATUSUTF8 / MSG_GET

Call

```
DSM_Entry(pOrigin, NULL, DG_CONTROL, DAT_STATUSUTF8, MSG_GET,
pStatusUtf8);
```

`pStatusUtf8` = pointer to a `TW_STATUSUTF8` structure.

Valid States

3 through 7

Description

Translate the contents of a `TW_STATUS` structure received from a Source into a localized UTF-8 encoded string.

Application

This operation can be called at anytime, with the contents of any `TW_STATUS` structure that it has received from the Source. The Source returns a value indicating the number of bytes (not characters) of data, including the terminating NUL byte. It also returns a handle to a UTF-8 encoded string, which the Application must lock before accessing, and which it must unlock and free when it is done.

Source

Translates the full contents of a `TW_STATUS` structure into a localized UTF-8 encode string, returning back a handle to that string, and the number of bytes (not characters) in the string, including the terminating NUL byte.

The Source returns a generic message if it is asked to return a string for a status code that it does not recognize.

Return Codes

```
TWRC_SUCCESS
TWRC_FAILURE
    TWCC_BADVALUE// something is wrong with &StatusUtf8
```

See Also

`DG_CONTROL / DAT_STATUS / MSG_GET (from Application to Source Manager)`
`DG_CONTROL / DAT_STATUS / MSG_GET (from Application to Source)`

DG_CONTROL / DAT_TWAINDIRECT / MSG_SETTASK

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_TWAINDIRECT, MSG_SETTASK,
pTwainDirect);
```

`pUserInterface` = A pointer to a `TW_TWAINDIRECT` structure.

Valid States

4 only

Description

Sends a TWAIN Direct task from the application to the driver.

Application

The application sets `TW_TWAINDIRECT`.SizeOf to the size of the `TW_TWAINDIRECT` structure (this is done in case new fields are added in the future).

The application sets `TW_TWAINDIRECT`.CommunicationManager to the current system being used to connect the application to the scanner.

The application then creates a handle containing a TWAIN Direct task (in UTF-8 JSON format as a NUL-terminated C string). `TW_TWAINDIRECT`.Send receives this handle, and `TW_TWAINDIRECT`.SendSize is set to the number of bytes of data in the JSON task (not including the trailing zero). Refer to the See Also section below for more information about TWAIN Direct.

When the operation is successfully completed, the application frees the `TW_TWAINDIRECT`.Receive handle.

Source

The source only processes fields that fit within the range of `TW_TWAINDIRECT`.SizeOf (this is done in case new fields are added in the future).

At this time the `TW_TWAINDIRECT`.CommunicationManager is informational only, but it may be used at some future time to modify the way TWAIN Direct tasks are interpreted.

The task inside of the `TW_TWAINDIRECT`.Send buffer is processed. The response is returned in the `TW_TWAINDIRECT`.Receive buffer in UTF-8 JSON format as a NUL-terminated C-string. The source allocates the handle for this and sets `TW_TWAINDIRECT`.ReceiveSize to the number of bytes returned (not including the trailing zero).

The "scan" action is always ignored by TWAIN. The caller must use `DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS` with a `TW_USERINTERFACE`.ShowUI value of `FALSE` to start scanning

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

`TWCC_BADVALUE`

`TWCC_SEQERROR`

See Also

"The TWAIN Direct Specification" – http://www.twaindirect.org

DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_USERINTERFACE, MSG_DISABLEDS,
pUserInterface);
```

`pUserInterface` = A pointer to a `TW_USERINTERFACE` structure.

Valid States

5 only (Transitions to State 4, if successful)

Description

This operation causes the Source's user interface, if displayed during the `DG_CONTROL` / `DAT_USERINTERFACE` / `MSG_ENABLEDS` operation, to be lowered. The Source is returned to State 4, where capability negotiation can again occur. The application can invoke this operation either because it wants to shut down the current session, or in response to the Source "posting" a `MSG_CLOSEDSREQ` event to it. Rarely, the application may need to close the Source because an error condition was detected.

Application

References the same `pUserInterface` structure as during the `MSG_ENABLEDS` operation. This implies that the application keep a copy of this structure locally as long as the Source is enabled.

If the application did not display the Source's built-in user interface, it will most likely invoke this operation either when all transfers have been completed or aborted (`TW_PENDINGXFERS`.Count = 0).

Source

If the Source's user interface is displayed, it should be lowered. The Source returns to State 4 and is again available for capability negotiation.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE
   TWCC_BADDEST       /* No such Source in-session */
                      /* with application          */
   TWCC_SEQERROR      /* Operation invoked in      */
                      /* invalid state             */
```

See Also

`DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ (from Source to Application)`
`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

Event loop information (in Chapter 3, "Application Implementation")

DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_USERINTERFACE, MSG_ENABLEDS,
pUserInterface);
```

`pUserInterface` = A pointer to a `TW_USERINTERFACE` structure

Valid States

4 only (Transitions to State 5, if successful)

Description

This operation causes three responses in the Source:

- Places the Source into a "ready to acquire" condition. If the application raises the Source's user interface (see #2, below), the Source will wait to assert `MSG_XFERREADY` until the "GO" button in its user interface or on the device is clicked. If the application bypasses the Source's user interface, this operation causes the Source to become immediately "armed". That is, the Source should assert `MSG_XFERREADY` as soon as it has data to transfer.
- The application can choose to raise the Source's built-in user interface, or not, using this operation. The application signals the Source's user interface should be displayed by setting pUserInterface->ShowUI to `TRUE`. If the application does not want the Source's user interface to be displayed, or wants to replace the Source's user interface with one of its own, it sets pUserInterface->ShowUI to `FALSE`. If activated, the Source's user interface will remain displayed until it is closed by the user or explicitly disabled by the application (see Note).
- Terminates Source's acceptance of "set capability" requests from the application. Capabilities can only be negotiated in State 4 (unless special arrangements are made using the `CAP_EXTENDEDCAPS` capability). Values of capabilities can still be inquired in States 5 through 7.

Note: Once the Source is enabled, the application **must** begin sending the Source every event that enters the application's main event loop. The application must continue to send the Source events until it disables (`MSG_DISABLEDS`) the Source. This is true even if the application chooses not to use the Source's built-in user interface.

Application

Set `pUserInterface->ShowUI` to `TRUE` to display the Source's built-in user interface, or to `FALSE` to place the Source in an "armed" condition so that it is immediately prepared to acquire data for transfer. Set ShowUI to `FALSE` <u>only</u> if bypassing the Source's built-in user interface—that is, only if the application is prepared to handle <u>all</u> user interaction necessary to acquire data from the selected Source.

Sources are not required to be enabled without showing their User Interface (i.e. `TW_USERINTERFACE`.ShowUI = `FALSE`). If a Source does not support ShowUI = `FALSE`, they will continue to be enabled just as if ShowUI = `TRUE`, but return `TWRC_CHECKSTATUS`. The application can check for this Return Code and continue knowing the Source's User Interface is being displayed.

Watch the value of pUserInterface->ModalUI after the operation has completed to see if the Source's user interface is modal or modeless.

The application must maintain a local copy of pUserInterface while the Source is enabled.

- **Windows**: Set pUserInterface->hParent to a handle (hWnd) to the window that will act as the Source's parent.
- **Macintosh**: Set pUserInterface->hParent to NULL.
- **Linux**: Set pUserInterface->hParent to NULL.

Note: Application should establish that the Source can supply compatible ICAP_PIXELTYPEs and ICAP_BITDEPTHs prior to enabling the Source. The application **must** verify that the Source can supply data of a type it can consume. If this operation fails, the application should notify the user that the device and application are incompatible due to data type mismatch. If the application diligently sets SupportedGroups in its identity structure before it tries to open the Source, the Source Manager will, in the Select Source dialog or through the MSG_GETFIRST/MSG_GETNEXT mechanism, filter out the Sources that don't match these SupportedGroups.

Source

If pUserInterface->ShowUI is TRUE, the Source should display its user interface and wait for the user to initiate an acquisition. If pUserInterface->ShowUI is FALSE, the Source should immediately begin acquiring data based on its current configuration (a device that requires the user to push a button on the device, such as a hand-scanner, will be "armed" by this operation and will assert MSG_XFERREADY as soon as the Source has data ready for transfer). The Source should fail any attempt to set a capability value (TWRC_FAILURE / TWCC_SEQERROR) until it returns to State 4 (unless an exception approval exists via a CAP_EXTENDEDCAPS agreement).

Set pUserInterface->ModalUI to TRUE if your built-in user interface is modal. Otherwise, set it to FALSE.

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.

Note: It is strongly recommended that all Sources support being enabled without their User Interface if the application requests (TW_USERINTERFACE.ShowUI = FALSE). But if your Source cannot be used without its User Interface, it should enable showing the Source User Interface (just as if ShowUI = TRUE) but return TWRC_CHECKSTATUS. All Sources, however, must support the CAP_UICONTROLLABLE. This capability reports whether or not a Source allows ShowUI = FALSE. An application can use this capability to know whether the Source-supplied user interface can be suppressed before it is displayed.

Return Codes

```
TWRC_SUCCESS
TWRC_CHECKSTATUS            /* Source cannot enable     */
```

```
                            /* without User Interface   */
                            /* so it enabled with the   */
                            /* User Interface.          */
TWRC_FAILURE
   TWCC_BADDEST             /* No such Source in-session */
                            /* with application          */
   TWCC_LOWMEMORY           /* Not enough memory to open */
                            /* the Source                */
   TWCC_OPERATIONERROR      /* Internal Source error;    */
                            /* handled by the Source     */
   TWCC_SEQERROR            /* Operation invoked in      */
                            /* invalid state             */
   TWCC_NOMEDIA       /* Source has nothing to capture */
```

See Also

DG_CONTROL / DAT_NULL / MSG_CLOSEDSREQ (from Source to Application)
DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS

Capability - CAP_INDICATORS

Event loop information (in Chapter 3, "Application Implementation")

DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_USERINTERFACE,
MSG_ENABLEDSUIONLY, pUserInterface);
```

`pUserInterface` = A pointer to a `TW_USERINTERFACE` structure.

Valid States

4 only (transitions to State 5, if successful)

Description

This operation is very similar to `DG_CONTROL/ DAT_USERINTERFACE/ MSG_ENABLEDS` operation except that no image transfer will take place. This operation is used by applications that wish to display the source user interface to allow the user to manipulate the sources current settings for DPI, paper size, etc. but not acquire an image. The ShowUI member of the `TW_USERINTERFACE` structure is ignored since this operations only purpose is to display the source UI. The other members of the `TW_USERINTERFACE` structure have the same meaning as in the `DG_CONTROL/ DAT_USERINTERFACE/ MSG_ENABLEDS` operation.

This operation has the following effects.

- The source transitions from state 4 to state 5. The source will display its user interface dialog but will not have a scan button (unless its only purpose is to preview the image).
- The application must begin sending the Source every event that enters the applications main event loop. This mechanism is the same as in the `MSG_ENABLEDS` operation.
- When the user hits OK or cancel from the source user interface dialog the source will send either `MSG_CLOSEDSOK` or `MSG_CLOSEDSREQ` Message.
- To close the source the application will respond back by sending a `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`. This source closes the dialog and then transitions from state 5 back to state 4 .

DG_CONTROL / DAT_XFERGROUP / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_CONTROL, DAT_XFERGROUP, MSG_GET,
pXferGroup);
```

`pXferGroup` = A pointer to a `TW_UINT32` value.

Valid States

4 through 6

Description

Returns the Data Group (the type of data) for the upcoming transfer. The Source is required to only supply one of the DGs specified in the SupportedGroups field of `pOrigin`.

Application

Should have previously (during a `DG_CONTROL` / `DAT_PARENT` / `MSG_OPENDSM`) set `pOrigin`. SupportedGroups to reflect the DGs the application is interested in receiving from a Source. Since `DG_xxxx` identifiers are bit flags, the application can perform a bitwise OR of `DG_xxxx` constants of interest to build the `SupportedGroups` field (this is appropriate when more kinds of data than `DG_IMAGE` are available).

Note: Version 1.x of the Toolkit defines `DG_IMAGE` and `DG_AUDIO` as the sole Data Groups (`DG_CONTROL` is masked from any processing of `SupportedGroups`). Future versions of TWAIN may define support for other DGs.

Source

Set `pXferGroup` to the `DG_xxxx` constant that identifies the type of data that is ready for transfer from the Source (`DG_IMAGE` is the only non-custom Data Group defined in TWAIN version 1.x).

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST        /* No such Source in-session with */
                       /* application                    */

   TWCC_SEQERROR       /* Operation invoked in invalid   */
                       /* state                          */
```

See Also

`DG_CONTROL / DAT_XFERGROUP / MSG_SET`

DG_CONTROL / DAT_XFERGROUP / MSG_SET

Call

```
DSM_Entry (pOrigin, pDest, DG_CONTROL, DAT_XFERGROUP,
   MSG_SET, pXFerGroup);
```

`pXferGroup` = A pointer to a `TW_UINT32` value.

Valid States

6 only

Description

The transfer group determines the kind of data being passed from the Source to the Application. By default a TWAIN Source must default to `DG_IMAGE`. Currently the only other data group supported is `DG_AUDIO`, which is a feature supported by some digital cameras.

An Application changes the data group in State 6 to indicate that it wants to transfer any audio data associated with the current image. The transfers follow the typical TWAIN State 6 – State 7 – State 6 pattern for each audio snippet transferred. When the application is done transferring audio data it must change back to `DG_IMAGE` in order to move on to the next image or to end the transfers and return to State5.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST – no such Source in session with application.

   TWCC_BADPROTOCOL - capability not supported.

   TWCC_SEQERROR - not state 6.
```

See Also

`DG_CONTROL / DAT_XFERGROUP / MSG_GET`

DG_IMAGE / DAT_CIECOLOR / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_CIECOLOR, MSG_GET, pCIEColor);
```

pCIEColor = A pointer to a `TW_CIECOLOR` structure.

Valid States

4 through 6

Description

Background - The `DAT_CIECOLOR` data argument type is used to communicate the parametrics for performing a transformation from any arbitrary set of tri-stimulus values into CIE XYZ color space. Color data stored in this format is more readily manipulated mathematically than some other spaces. Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

This operation causes the Source to report the currently active parameters to be used in converting acquired color data into CIE XYZ.

Application

Prior to invoking this operation, the application should establish that the Source can provide data in CIE XYZ form. This can be determined by invoking a `MSG_GET` on `ICAP_PIXELTYPE`. If `TWPT_CIEXYZ` is one of the supported types, then these operations are valid. The application can specify that transfers should use the CIE XYZ space by invoking a `MSG_SET` operation on `ICAP_PIXELTYPE` using a `TW_ONEVALUE` container structure whose value is `TWPT_CIEXYZ`.

No special set up is required. Invoking this operation following the transfer (after the Source is back in State 6) will guarantee that the exact parameters used to convert the image are reported.

Source

Fill pCIEColor with the current values applied in any conversion of image data to CIE XYZ. If no values have been set by the application, fill the structure with either the values calculated for this image or the Source’s default values, whichever most accurately reflect the state of the Source.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL       /* Source does not support the  */
                          /* CIE descriptors                */

   TWCC_SEQERROR          /* Operation invoked in invalid */
                          /* state                           */
```

See Also

Capability - `ICAP_PIXELTYPE`
Chapter A, "TWAIN Articles"

DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_EXTIMAGEINFO, MSG_GET,
pExtImageInfo);
```

pExtImageInfo = A pointer to a `TW_EXTIMAGEINFO` structure.

Valid States

7 only, after receiving `TWRC_XFERDONE`

Description

This operation is used by the application to query the data source for extended image attributes, .e.g. bar codes found on a page. The extended image information will be returned in a `TW_EXTIMAGEINFO` structure.

Application

To query extended image information, set the `pExtImageInfo` fields as follows:

The Application will allocate memory for the necessary container structure, the source will fill the values, and then application will free it up.

```
pExtImageInfo->NumInfos = Desired number of information;
pExtImageInfo->Info[0].InfoID = TWEI_xxxx;
pExtImageInfo->Info[1].InfoID = TWEI_xxxx;
```

Source

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_DATANOTAVAILABLE` in the ReturnCode field of `TW_INFO` structure.

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

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

```
pExtImageInfo->Info[0].ReturnCode = TWRC_SUCCESS;
pExtImageInfo->Info[0].ItemType = TWTY_UINT32;
pExtImageInfo->Info[0].NumItems = 1;
pExtImageInfo->Info[0].Item = 20;
```

For TWEI_FORMTEMPLATEMATCH:

```
pExtImageInfo->Info[0].RetCode = TWRC_SUCCESS;
pExtImageInfo->Info[0].ItemType = TWTY_STR255;
pExtImageInfo->Info[0].NumItems = 1;
pExtImageInfo->Info[0].Item = GlobalAlloc( GHND, sizeof(TW_STR255) );
```

Return Codes

```
TWRC_SUCCESS
TWRC_FAILURE
    TWCC_BADPROTOCOL /* Source does not support extended image  */
                     /* information                              */
      TWCC_SEQERROR  /* Not State 7, or in State 7 but TWRC_XFERDONE */
                     /* has not been received yet                   */
   TWCC_NOMEDIA      /* Source has nothing to capture */
```

See Also

Capability ICAP_EXTIMAGEINFO, ICAP_SUPPORTEDEXTIMAGEINFO

DG_IMAGE / DAT_FILTER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_FILTER, MSG_GET, pFilter);
```

`pFilter` = A pointer to a `TW_FILTER` structure.

Valid States

4 through 6

Description

Causes the Source to return the filter parameters that will be used during the next image acquisition.

`TW_FILTER` describes the color characteristic of the subtractive filter applied to the image data. Multiple filters may be applied to a single acquisition.

Application

The Application allocates the `TW_FILTER` structure. The Source will allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. `TW_FILTER/Descriptors` field specifies the number of elements in the array returned in `hDescriptors`. The size of the `TW_FILTER_DESCRIPTOR` structure may vary across the versions, so use the `TW_FILTER_DESCRIPTOR/Size` filed to step through the array. The Application has to deallocate `hDescriptors` after it is not needed anymore.

Source

Fill `pFilter` with the filter parameters that will be applied during the next acquisition.

The Source must allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The Source must check the `TW_FILTER/Size` field to see which of the structure fields it can fill.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL      /* Source does not support it */

   TWCC_SEQERROR         /* Operation invoked in invalid */
                         /* state                        */
```

See Also

`DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT`
`DG_IMAGE / DAT_FILTER / MSG_SET`
`DG_IMAGE / DAT_FILTER / MSG_RESET`

Capability - `ICAP_FILTER`

DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_FILTER, MSG_GET, pFilter);
```

`pFilter` = A pointer to a `TW_FILTER` structure.

Valid States

4 through 6

Description

Causes the Source to return the power-on default values applied to the filter.

Source will fill `TW_FILTER` structure fields `Descriptors` and `hDescriptors` with `0`. This means no filter will be applied.

`TW_FILTER` describes the color characteristic of the subtractive filter applied to the image data. Multiple filters may apply to a single acquisition.

Application

The Application allocates the `TW_FILTER` structure. The Source will allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The `TW_FILTER/Descriptors` field specifies the number of elements in the array returned in `hDescriptors`. The size of the `TW_FILTER_DESCRIPTOR` structure may vary across the versions, so use the `TW_FILTER_DESCRIPTOR/Size` filed to step through the array. The Application has to deallocate `hDescriptors` after it is not needed anymore.

Source

Fill `pFilter` with the filter parameters that will be applied during the next acquisition.

The Source must allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The Source must check the `TW_FILTER/Size` field to see which of the structure fields it can fill.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

```
   TWCC_BADPROTOCOL      /* Source does not support it */
   TWCC_SEQERROR         /* Operation invoked in invalid */
                         /* state                        */
```

See Also

`DG_IMAGE / DAT_FILTER / MSG_GET`
`DG_IMAGE / DAT_FILTER / MSG_SET`
`DG_IMAGE / DAT_FILTER / MSG_RESET`

Capability - `ICAP_FILTER`

DG_IMAGE / DAT_FILTER / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_FILTER, MSG_SET, pFilter);
```

`pFilter` = A pointer to a `TW_FILTER` structure.

Valid States

4 only

Description

Allows the Application to configure the filter parameters that will be used during the next image acquisition.

`TW_FILTER` describes the color characteristic of the subtractive filter applied to the image data. Multiple filters may be applied to a single acquisition.

If the Source supports `DAT_FILTER` as well, then it will apply the filter set by the last `SET` operation invoked by the Application. Setting/Resetting `ICAP_FILTER` will clear the filter associated with `DAT_FILTER`. Setting/Resetting `DAT_FILTER` will clear the filter associated with `ICAP_FILTER`.

Application

The Application allocates the `TW_FILTER` structure. The Application also has to allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The `TW_FILTER/Descriptors` field specifies the number of elements in the array in `hDescriptors`. If this number exceeds the `TW_FILTER/MaxDescriptors` returned by any `GET` operation, then the Source will accept only the allowed number of descriptors and it will return `TWRC_CHECKSTATUS`.

Source

Adopt the requested filter parameters that will be applied during the next acquisition. If a value does not exactly match an available value, match the value as closely as possible and return `TWRC_CHECKSTATUS`. If the value is beyond the range of available values, clip to the nearest value and return `TWRC_FAILURE/TWCC_BADVALUE`.

The Source must check the `TW_FILTER/Size` field to see which of the structure fields are valid. The size of the `TW_FILTER_DESCRIPTOR` structure may vary across the versions, so use the `TW_FILTER_DESCRIPTOR/Size` filed to step through the array.

The Source must discard all previously set filter parameters.

Return Codes

```
TWRC_SUCCESS

TWRC_CHECKSTATUS           /* value(s) could not be matched exactly */

TWRC_FAILURE
```

```
TWCC_BADPROTOCOL      /* Source does not support it */
TWCC_BADVALUE         /* illegal value(s) */
TWCC_SEQERROR         /* Operation invoked in invalid */
                      /* state                        */
```

See Also

DG_IMAGE / DAT_FILTER / MSG_GET
DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT
DG_IMAGE / DAT_FILTER / MSG_RESET

Capability - ICAP_FILTER

DG_IMAGE / DAT_FILTER / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_FILTER, MSG_SET, pFilter);
```

`pFilter` = A pointer to a `TW_FILTER` structure.

Valid States

4 only

Description

Return the Source to using its power-on default values when it is applying the filter.

Source will fill `TW_FILTER` structure fields `Descriptors` and `hDescriptors` with `0`. This means no filter will be applied.

`TW_FILTER` describes the color characteristic of the subtractive filter applied to the image data. Multiple filters may be applied to a single acquisition.

If the Source supports `DAT_FILTER` as well, then it will apply the filter set by the last `SET` operation invoked by the Application. Setting/Resetting `ICAP_FILTER` will clear the filter associated with `DAT_FILTER`. Setting/Resetting `DAT_FILTER` will clear filter associated with `ICAP_FILTER`.

Application

The application allocates the `TW_FILTER` structure. The Source will allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The `TW_FILTER/Descriptors` field specifies the number of elements in the array returned in `hDescriptors`. The size of the `TW_FILTER_DESCRIPTOR` structure may vary across the versions, so use the `TW_FILTER_DESCRIPTOR/Size` filed to step through the array. The Application has to deallocate `hDescriptors` after it is not needed anymore.

Source

Fill `pFilter` with the filter parameters that will be applied during the next acquisition.

The Source must allocate memory for the `TW_FILTER_DESCRIPTOR` array if any. The Source must check `TW_FILTER/Size` field to see which of the structure fields it can fill.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL      /* Source does not support it   */

   TWCC_SEQERROR         /* Operation invoked in invalid */
                         /* state                        */
```

See Also

`DG_IMAGE / DAT_FILTER / MSG_GET`
`DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT`
`DG_IMAGE / DAT_FILTER / MSG_SET`
Capability - `ICAP_FILTER`

DG_IMAGE / DAT_GRAYRESPONSE / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_GRAYRESPONSE, MSG_RESET,
pResponse);
```

`pResponse` = A pointer to a `TW_GRAYRESPONSE` structure.

Valid States

4 only

Description

Background - The two `DAT_GRAYRESPONSE` operations allow the application to specify a transfer curve that the Source should apply to the grayscale it acquires. This curve should be applied to the data prior to transfer. The Source should maintain an "identity response curve" to be used when it is `MSG_RESET`.

The `MSG_RESET` operation causes the Source to use its "identity response curve." The identity curve causes no change in the values of the captured data when it is applied.

Application

No special action.

Source

Apply the identity response curve to all future grayscale transfers. This means that the Source will transfer the grayscale data exactly as acquired.

Return Codes

TWRC_SUCCESS

TWRC_FAILURE

```
TWCC_BADPROTOCOL      /* Source does not support      */
                      /* grayscale response curves    */
TWCC_SEQERROR         /* Operation invoked in invalid */
                      /* state                        */
```

See Also

DG_IMAGE / DAT_GRAYRESPONSE / MSG_SET

Capability - ICAP_PIXELTYPE

DG_IMAGE / DAT_GRAYRESPONSE / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_GRAYRESPONSE, MSG_SET,
pResponse);
```

pResponse = A pointer to a `TW_GRAYRESPONSE` structure.

Valid States

4 only

Description

Background - The two `DAT_GRAYRESPONSE` operations allow the application to specify a transfer curve that the Source should apply to the grayscale it acquires. This curve should be applied to the data prior to transfer. The Source should maintain an "identity response curve" to be used when it is `MSG_RESET`. This identity curve should cause no change in the values of the data it is applied to.

This operation causes the Source to transform any grayscale data according to the response curve specified.

Application

All three elements of the response curve for any given index should hold the same value (the curve is stored in a `TW_ELEMENT8` which contains three "channels" of data). The Source may not support this operation. The application should be diligent to examine the return code from this operation.

Source

Apply the specified response curve to all future grayscale transfers. The transformation should be applied before the data is transferred.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL      /* Source does not support      */
                         /* grayscale response curves    */

   TWCC_SEQERROR         /* Operation invoked in invalid */
                         /* state                        */
```

See Also

`DG_IMAGE / DAT_GRAYRESPONSE / MSG_RESET`

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_ICCPROFILE / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_ICCPROFILE, MSG_GET,
pICCProfile);
```

`pICCProfile` = A pointer to a `TW_MEMORY` structure.

Valid States

6 or 7

Description

This operation provides the application with the ICC profile associated with the image which is about to be transferred (state 6) or is being transferred (state 7).

Application

The application can use the operation to retrieve an ICC profile associated with image data. This profile could then be used to transform the image to sRGB or to embed into a JPEG or TIFF file that the application is writing. If the application is having the source write the file (`ICAP_XFERMECH` of `TWSX_FILE`), then there is no need to call this triplet and the capability `ICAP_ICCPROFILE` should be used. It is important that the application **not** allocate the memory itself. Although a `TW_MEMORY` structure is used, the memory is **always** allocated by the source. The application should set the entire structure to zero. Note that not all sources will have profiles and some might have profiles for color data but not for grayscale data.

The profile returned always applies to the current data being transferred and not the image being currently scanned. This distinction is important for scanners that buffer pages since the data being transferred is most likely not the image being currently scanned.

For optimization, it is recommended that applications attempt to only call this on an as needed basis. In general, the application calls this once for each batch. However, it is important to note any changes in the pixeltype during a batch because changes in pixeltype will mandata a change in profile. While most scanners will not change the pixeltype int eh middle of a batch, those with job control sheets may do so.

Source

Allocates the TheMem member and sets the Flags member to have `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.

See Also

Capability - `ICAP_ICCPROFILE`

The new `ICAP_PIXELTYPE` values are:

`TWPT_CIELAB`	
`TWPT_SRGB`	Specifies that the data coming back has been calibrated to sRGB

If a source supports `TWPT_SRGB`, it must also support `TWPT_RGB` for backwards compatibility. If it only has `sRGB` data, then it should still support `TWPT_RGB` and pass back its sRBG data in that mode.

DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGEFILEXFER, MSG_GET, NULL);
```

This operation acts on `NULL` data. File information can be set with the `DG_CONTROL` / **`DAT_SETUPFILEXFER`** / `MSG_SET` operation.

Valid States

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

Description

This operation is used to initiate the transfer of an image from the Source to the application via the disk-file transfer mechanism. It causes the transfer to begin.

Application

No special set up or action required. Application should have already invoked the `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation unless the Source's default transfer format and file name (typically, `TWAIN.TMP`) are acceptable to the application. The application need only invoke this operation once per image transferred.

Note: **I**f the application is planning to receive multiple images from the Source while using the Source's default file name, the application should plan to pause between transfers and copy the file just written. The Source will overwrite the file unless it is instructed to write to a different file.

Note: Applications can specify a unique file for each transfer using `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation in State 6 or 5 (and 4, of course).

Source

Acquire the image data, format it, create any appropriate header information, and write everything into the file specified by the previous `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation, and close the file.

Handling Possible File Conditions:

- If the application did not set conditions up using the **`DAT_SETUPFILEXFER`** / `MSG_SET` operation during this session, use your own default file name, file format, and location for the created file.
- If the specified file already exists, overwrite the file in place.
- If the specified file does not exist, create the file.
- If the specified file exists and cannot be accessed, or a system error occurs while writing the file, report the error to the user and return `TWRC_FAILURE` with `TWCC_OPERATIONERROR`. Stay in

State 6. The file contents are invalid. The image whose transfer failed is still a pending transfer so do not decrement `TW_PENDINGXFERS.Count`.

- If the file is written successfully, return `TWRC_XFERDONE`.
- If the user cancels the transfer, return `TWRC_CANCEL`.

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 */
```

See Also

`DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET`
`DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGEINFO / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`

Capabilities - `ICAP_XFERMECH`, `ICAP_IMAGEFILEFORMAT`

DG_IMAGE / DAT_IMAGEINFO / MSG_GET

Call

`DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGEINFO, MSG_GET,pImageInfo);`

`pImageInfo` = A pointer to a `TW_IMAGEINFO` structure.

Valid States

6 and 7 (State 7 only after receiving `TWRC_XFERDONE`)

Description

When called in State 6, this operation provides to the application general image description information about the image about to be transferred.

When called in State 7, this operation provides the Application with specific image description information about the current image that has just been transferred. It is important during a Memory transfer to call this triplet only after `TWRC_XFERDONE` is received, since that is the only time the Source will know all the final image information.

The same data structure type is used regardless of the mechanism used to transfer the image (Native, Disk File, or Buffered Memory transfer).

Application

The Application can use this operation to check the parameters of the image before initiating the transfer during State 6, or to clarify image parameters during State 7 after the transfer is complete.

Applications may inform Sources that they accept -1 value for `ImageHeight/ImageWidth` by setting the `ICAP_UNDEFINEDIMAGESIZE` capability to `TRUE`.

Should the Application decide to invoke any Source features that allow the image description information to change during scanning (such as `ICAP_UNDEFINEDIMAGESIZE`) and still wish to transfer in Buffered memory mode, a `DG_CONTROL/DAT_IMAGEINFO/MSG_GET` call must be made in State 7 after receiving `TWRC_XFERDONE` to properly interpret the image data. This is not the default behavior of the Source.

Note that the speed at which the Application supplies buffers may determine the scanning speed.

Source

For maximum compatibility with applications, Data Source writers are strongly encouraged to report back finished image values in State 6. In other words, calls to `DAT_IMAGEINFO` should return the same identical values in State 6, and in State 7 after `TWRC_XFERDONE` has been issued to the application.

During State 6 - Fills in all fields in pImageInfo. All fields are filled in as you would expect with the following exceptions:

XResolution or YResolution

Set to -1 if the device creates data with no inherent resolution (such as a digital camera).

ImageWidth

Set to -1 if the image width to be acquired is unknown (such as when using a hand-held scanner and dragging left-to-right) , and the Application has set `ICAP_UNDEFINEDIMAGESIZE` to `TRUE`. In this case the Source must transfer the image in tiles.

ImageLength

`ImageLength`—Set to -1 if the image length to be acquired is unknown (such as when using a hand-held scanner and dragging top-to-bottom), and the Application has set `ICAP_UNDEFINEDIMAGESIZE` to `TRUE`.

During State 7 - Fills in all fields in pImageInfo. All fields are filled in as during State 6, except ImageWidth and ImageLength MUST be valid. Source shall return `TWRC_SEQERROR` if call is made before `TWRC_XFERDONE` is sent.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE
   TWCC_BADDEST     /* No such Source in-session with */
                      /* application                    */

   TWCC_SEQERROR    /* Operation invoked in invalid   */
                      /* state                          */
```

See Also

`DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

Capabilities - `ICAP_BITDEPTH`, `ICAP_COMPRESSION`, `ICAP_PIXELTYPE`, `ICAP_PLANARCHUNKY`, `ICAP_XRESOLUTION`, `ICAP_YRESOLUTION`

DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGELAYOUT, MSG_GET,
pImageLayout);
```

`pImageLayout` = A pointer to a `TW_IMAGELAYOUT` structure.

Valid States

4 through 6

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 scanner and the extents of the image are expressed in the unit of measure 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`.

Application

No special set up or action required, unless the current units of measure are unacceptable. In that case, the application must re-negotiate `ICAP_UNITS` prior to invoking this operation. Remember to do this in State 4—the only state wherein capabilities can be set or reset.

Beyond supplying possibly interesting position information on the image to be transferred, the application can use this structure to constrain the final size of the image and to relate the image within a series of pages or documents (see the `DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET` operation).

Source

Fill all fields of `pImageLayout`. Most Sources will set `FrameNumber`, `PageNumber`, and `DocumentNumber` to `1`.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST        /* No such Source in-session    */
                         /* with application              */

   TWCC_SEQERROR       /* Operation invoked in invalid */
                         /* state                         */
```

See Also

`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET`

Capabilities - Many such as `ICAP_FRAMES`, `ICAP_MAXFRAMES`, `ICAP_UNITS`

DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGELAYOUT, MSG_GETDEFAULT,
pImageLayout);
```

`pImageLayout` = A pointer to a `TW_IMAGELAYOUT` structure.

Valid States

4 through 6

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.).

This operation returns the default information on the layout of an image. This is the size and position of the image that will be acquired from the Source if the acquisition is started with the Source (and the device it is controlling) in its power-on state (for instance, most flatbed scanners will capture the entire bed).

Application

No special set up or action required.

Source

Fill in all fields of `pImageLayout` with the device's power-on origin and extents. Most Sources will set `FrameNumber`, `PageNumber`, and `DocumentNumber` to `1`.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST      /* No such Source in-session    */
                       /* with application              */

   TWCC_SEQERROR     /* Operation invoked in invalid */
                       /* state                         */
```

See Also

`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET`

Capabilities - `ICAP_FRAMES`, `ICAP_MAXFRAMES`, `ICAP_UNITS`

DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGELAYOUT, MSG_RESET,
pImageLayout);
```

`pImageLayout` = A pointer to a `TW_IMAGELAYOUT` structure.

Valid States

4 only

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.).

This operation sets the image layout information for the next transfer to its default settings.

Application

No special set up or action required. Ascertain the current settings of `ICAP_ORIENTATION`, `ICAP_PHYSICALWIDTH`, and `ICAP_PHYSICALHEIGHT` if you don't already know this device's power-on default values.

Source

Reset all the fields of the structure pointed at by pImageLayout to the device's power-on origin and extents. There is an implied resetting of `ICAP_ORIENTATION`, `ICAP_PHYSICALWIDTH`, and `ICAP_PHYSICALHEIGHT` to the device's power-on default values.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADDEST      /* No such Source in-session    */
                       /* with application              */

   TWCC_SEQERROR     /* Operation invoked in invalid */
                        /* state                         */
```

See Also

`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET`

Capabilities - `ICAP_FRAMES`, `ICAP_MAXFRAMES`, `ICAP_UNITS`

DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGELAYOUT, MSG_SET,
pImageLayout);
```

`pImageLayout` = A pointer to a `TW_IMAGELAYOUT` structure.

Valid States

4 only

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.).

This operation sets the layout for the next image transfer. This allows the application to specify the physical area to be acquired during the next image transfer (for instance, a frame-based application would pass to the Source the size of the frame the user selected within the application—the helpful Source would present a selection region already sized to match the layout frame size).

If the application and Source have negotiated one or more frames through `ICAP_FRAMES`, the frame set with this operation will only persist until the transfer following this one. Otherwise, the frame will persist as the current frame for the remainder of the session (unless superseded by negotiation on `ICAP_FRAMES` or another operation on `DAT_IMAGELAYOUT` overrides it).

The application writer should note that setting these values is a *request*. The Source should first try to match the requested values exactly. Failing that, it should approximate the requested values as closely as it can—extents of the approximated frame should at least equal the requested extents unless the device cannot comply. The Source should return `TWRC_CHECKSTATUS` if the actual values set in `pImageLayout->Frame` are greater than or equal to the requested values in both extents. If one or both of the requested values exceed the Source's available values, the Source should return `TWRC_FAILURE` with `TWCC_BADVALUE`. The application should check for these return codes and perform a `MSG_GET` to verify that the values set by the Source are acceptable. The application may choose to cancel the transfer if Source could not set the layout information closely enough to the requested values.

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`.

Return Codes

```
TWRC_SUCCESS

TWRC_CHECKSTATUS     /* Source approximated the requested*/
                     /* values                           */

TWRC_FAILURE

   TWCC_BADDEST      /* No such Source in-session       */
                     /* with application                */

   TWCC_BADVALUE     /* Specified Layout values illegal */
                     /* for Source                      */

   TWCC_SEQERROR     /* Operation invoked in invalid    */
                     /* state                           */
```

See Also

`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET`

Capabilities - `ICAP_FRAMES`, `ICAP_MAXFRAMES`, `ICAP_UNITS`

DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGEMEMFILEXFER, MSG_GET,
pImageMemXfer);
```

`pImageMemXfer` = A pointer to a `TW_IMAGEMEMXFER` structure.

File format information can be set with the `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation.

Valid States

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

Description

This operation is used to initiate the transfer of an image from the Source to the application via the Memory-File transfer mechanism.

This operation supports the transfer of successive blocks of an image file from the Source into one or more main memory transfer buffers. These buffers are allocated and owned by the application. The application should repeatedly invoke this operation while `TWRC_SUCCESS` is returned by the Source.

Application

No special set up is required. The application should have already invoked the `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation unless the Source's default file format is acceptable to the application (the filename is not used, since this transfer is being done in memory). The `DG_CONTROL` / `DAT_SETUPMEMXFER` / `MSG_GET` operation should be used to determine the valid range of sizes for transferring the image. The application only needs to invoke both of these operations once per image transferred.

The application will allocate one or more memory buffers to contain the data being transferred from the Source. The application may allocate enough buffer space to contain the entire image being transferred or, more commonly, use the transfer buffer(s) as a temporary holding area while the complete image is assembled elsewhere (on disk, for instance).

If the application sets up buffers that are either too small or too large, the Source will fail the operation returning `TWRC_FAILURE/TWCC_BADVALUE`.

Once the buffers have been set up, the application should fill `pImageMemXfer->Memory.Length` with the actual size (in bytes) of each memory buffer (which are, of course, all the same size).

Notes: Applications can specify a unique file format for each transfer using `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` in State 6 or 5 (and 4 also). Also note that although the images are being transferred in complete image formats, they are memory transfers, and will be chunked just like a `DG_IMAGE` / `DAT_IMAGEMEMXFER` / `MSG_GET` operation.

The size of the allocated buffer(s) should be homogeneous (don't change buffer sizes during transfer). The size the application selects should be based on the information returned by the Source from the DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation. The application should do its best to allocate transfer buffers of the size "preferred" by the Source. This will enhance the chances for superior transfer performance. The buffer size must be between MinBufSize and MaxBufSize as reported by the Source.

There is no concept of striping or tiling when using this operation. Data is transferred in generic chucks, which, depending on the file format, may result in partial header or footer information being sent in any given transfer. Applications are advised to avoid parsing the image format data until all of the blocks have been transferred

Source

If the application did not set up the conditions via the DAT_SETUPFILEXFER / MSG_SET operation during this session, use the Source's default file format for the transfer.

Prior to writing the first buffer, check pImageMemXfer->Memory.Length for the size of the buffer(s) the application has allocated. If the size lies outside the maximum or minimum buffer size communicated to the application during the DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation, return TWRC_FAILURE/TWCC_BADVALUE and remain in State 6.

If the buffer is of an acceptable size, fill in all fields of pImageMemXfer except *pImageMemXfer->Memory*. The Source must write the data block into the buffer referenced by pImageMemXfer->Memory.TheMem and store the actual number of bytes written into the buffer in pImageMemXfer->BytesWritten. Compressed and tiled data effects how the Source fills in these values.

Return TWRC_SUCCESS after successfully writing each buffer. Return TWRC_CANCEL if the Source needs to terminate the transfer before the last buffer is written (as when the user aborts the transfer from the Source's user interface). Return TWRC_XFERDONE to signal that the last buffer has been written. Following completion of the transfer, either after all the data has been written or the transfer has been canceled, remain in State 7 until explicitly transitioned back to State 6 by the application (DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER).

If TWRC_FAILURE occurred on the first buffer, the session remains in State 6. If failing on a subsequent buffer, the session remains in State 7. The strip whose transfer failed is still pending.

Notes on Memory Usage: Following a canceled transfer, the Source should dispose of the image that was being transferred and assure that any temporary variable and local buffer allocations are eliminated. The Source should be wary of allocating large temporary buffers or variables. Doing so may disrupt or even disable the transfer process. The application should be aware of the possible needs of the Source to allocate such space, however, and consider allocating all large blocks of RAM needed to support the transfer prior to invoking this operation. This may be especially important for devices that create image transfers of indeterminate size—such as hand-held scanners.

Return Codes

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

```
                         /* the specified image               */
TWRC_CANCEL              /* User aborted the transfer from     */
                          /* the Source                       */
TWRC_FAILURE
   TWCC_BADDEST          /* No such Source in-session          */
                         /* with application                  */
   TWCC_BADVALUE         /* Size of buffer did not             */
                         /* match TW_SETUPMEMXFER              */
   TWCC_OPERATIONERROR    /* Failure in the Source --           */
                          /* transfer invalid                  */
   TWCC_SEQERROR  /* Operation invoked in              */
                   /* invalid state                     */
   /* The following introduced for 2.0 or higher */
   TWCC_INTERLOCK         /* Cover or door is open     */
   TWCC_DAMAGEDCORNER     /* Document has a damaged corner */
   TWCC_FOCUSERROR        /* Focusing error during document capture */
   TWCC_DOCTOOLIGHT       /* Document is too light     */
   TWCC_DOCTOODARK        /* Document is too dark      */
   TWCC_NOMEDIA           /* Source has nothing to capture */
```

See Also

DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_CONTROL / DAT_SETUPFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGEINFO / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET

Capabilities - ICAP_COMPRESSION, ICAP_IMAGEFILEFORMAT, ICAP_XFERMECH

DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGEMEMXFER, MSG_GET,
pImageMemXfer);
```

`pImageMemXfer` = A pointer to a `TW_IMAGEMEMXFER` structure.

Valid States

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

Description

This operation is used to initiate the transfer of an image from the Source to the application via the Buffered Memory transfer mechanism.

This operation supports the transfer of successive blocks of image data (in strips or, optionally, tiles) from the Source into one or more main memory transfer buffers. These buffers (for strips) are allocated and owned by the application. For tiled transfers, the source allocates the buffers. The application should repeatedly invoke this operation while `TWRC_SUCCESS` is returned by the Source.

Application

The application will allocate one or more memory buffers to contain the data being transferred from the Source. The application may allocate enough buffer space to contain the entire image being transferred or, more commonly, use the transfer buffer(s) as a temporary holding area while the complete image is assembled elsewhere (on disk, for instance).

The size of the allocated buffer(s) should be homogeneous (don't change buffer sizes during transfer). The size the application selects should be based on the information returned by the Source from the `DG_CONTROL` / **`DAT_SETUPMEMXFER`** / `MSG_GET` operation. The application should do its best to allocate transfer buffers of the size "preferred" by the Source. This will enhance the chances for superior transfer performance. The buffer size must be between MinBufSize and MaxBufSize as reported by the Source. Further, the buffers must contain an even number of bytes. Memory buffers must be double-word aligned and should be padded with zeros at the end of each raster line.

If the application sets up buffers that are either too small or too large, the Source will fail the operation returning `TWRC_FAILURE/TWCC_BADVALUE`.

Once the buffers have been set up, the application should fill pImageMemXfer->Memory.Length with the actual size (in bytes) of each memory buffer (which are, of course, all the same size).

Windows only—The buffers should be allocated in global memory.

Source

Prior to writing the first buffer, check pImageMemXfer->Memory.Length for the size of the buffer(s) the application has allocated. If the size lies outside the maximum or minimum buffer size communicated to the application during the DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET operation, return TWRC_FAILURE/TWCC_BADVALUE and remain in State 6.

If the buffer is of an acceptable size, fill in all fields of pImageMemXfer except pImageMemXfer->Memory. The Source must write the data block into the buffer referenced by pImageMemXfer->Memory.TheMem. Store the actual number of bytes written into the buffer in pImageMemXfer->BytesWritten. Compressed and tiled data effects how the Source fills in these values.

Return TWRC_SUCCESS after successfully writing each buffer. Return TWRC_CANCEL if the Source needs to terminate the transfer before the last buffer is written (as when the user aborts the transfer from the Source's user interface). Return TWRC_XFERDONE to signal that the last buffer has been written. Following completion of the transfer, either after all the data has been written or the transfer has been canceled, remain in State 7 until explicitly transitioned back to State 6 by the application (DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER).

If TWRC_FAILURE occurred on the first buffer, the session remains in State 6. If failing on a subsequent buffer, the session remains in State 7. The strip whose transfer failed is still pending.

Notes on Memory Usage: Following a canceled transfer, the Source should dispose of the image that was being transferred and assure that any temporary variable and local buffer allocations are eliminated. The Source should be wary of allocating large temporary buffers or variables. Doing so may disrupt or even disable the transfer process. The application should be aware of the possible needs of the Source to allocate such space, however, and consider allocating all large blocks of RAM needed to support the transfer prior to invoking this operation. This may be especially important for devices that create image transfers of indeterminate size—such as hand-held scanners.

Return Codes

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

TWRC_XFERDONE             /* Source done transferring       */
                            /* the specified image            */

TWRC_CANCEL               /* User aborted the transfer from */
                            /* the Source                     */

TWRC_FAILURE

   TWCC_BADDEST           /* No such Source in-session */
                            /* with application          */

   TWCC_BADVALUE          /* Size of buffer did not    */
                            /* match TW_SETUPMEMXFER     */

   TWCC_OPERATIONERROR   /* Failure in the Source--   */
                            /* transfer invalid          */

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

                         /* The following introduced for 2.0 or higher */

   TWCC_INTERLOCK          /* Cover or door is open      */

   TWCC_DAMAGEDCORNER      /* Document has a damaged corner */
```

```
    TWCC_FOCUSERROR         /* Focusing error during document capture */
    TWCC_DOCTOOLIGHT        /* Document is too light      */
    TWCC_DOCTOODARK         /* Document is too dark       */
```

See Also

DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGEINFO / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET

Capabilities - ICAP_COMPRESSION, ICAP_TILES, ICAP_XFERMECH

DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_IMAGENATIVEXFER, MSG_GET,
pHandle);
```

`pHandle` = A pointer to a variable of the Operating Systems Native image format.

Windows: Pointer to a handle to a DIB (Device Independent Bitmap) located in memory.

Macintosh: The Pointer to a handle to a TIFF image. It is a TIFF file located in memory if both the application and the data source are TWAIN 2.4 and later. Pointer to a handle to a Picture (a PicHandle, QuickDraw picture) located in memory if either the application or the data source is TWAIN 2.3 and earlier..

Linux: : Pointer to a handle to a TIFF image. It is a TIFF file located in memory.

Refer to Chapter 12, "Operating System Dependencies" for more information on Native Transfer.

Valid States

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

Description

Causes the transfer of an image's data from the Source to the application, via the Native transfer mechanism, to begin. The resulting data is stored in main memory in a single block. The data is stored in the Operating Systems native image format. The size of the image that can be transferred is limited to the size of the memory block that can be allocated by the Source. If the image is too large to fit, the Source may resize or crop the image.

Note: This is the default transfer mechanism. All Source's support this mechanism. The Source will use this mechanism unless the application explicitly negotiates a different transfer mechanism with `ICAP_XFERMECH`.

Application

The application need only invoke this operation once per image. The Source allocates up to the largest block of available memory and transfers the image into it.

Read the image header to determine if the source has modified the image size to fit memory available. The application is responsible for deallocating the memory block holding the Native-format image.

Set `pHandle` pointing to a handle.

The Source will allocate the image buffer and return the handle to the address specified.

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

Source

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

Windows: Set `pHandle` pointing to a handle to a device-independent bit map (DIB) in memory.

Macintosh: Set `pHandle` pointing to a handle to a TIFF file in memory if both application and data source are version 2.4 or later. Set `pHandle` pointing to a handle to a Picture in memory if either the application or the data source is TWAIN 2.3 and earlier.

Linux: Set `pHandle` pointing to a handle to a TIFF file in memory.

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

Return Codes

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

TWRC_CANCEL              /* User aborted the transfer    */
                         /* within the Source            */

TWRC_FAILURE

   TWCC_BADDEST          /* No such Source in session */
                         /* with application          */

   TWCC_LOWMEMORY        /* Not enough memory for     */
                         /* image--cannot crop to fit */

   TWCC_OPERATIONERROR   /* Failure in the Source--   */
                         /* transfer invalid          */

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

   /* The following introduced for 2.0 or higher */

   TWCC_INTERLOCK        /* Cover or door is open     */

   TWCC_DAMAGEDCORNER    /* Document has a damaged corner */

   TWCC_FOCUSERROR       /* Focusing error during document capture */

   TWCC_DOCTOOLIGHT      /* Document is too light     */

   TWCC_DOCTOODARK       /* Document is too dark      */

   TWCC_NOMEDIA          /* Source has nothing to capture */
```

See Also

`DG_IMAGE / DAT_IMAGEINFO / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`

Capability - `ICAP_XFERMECH`

DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_JPEGCOMPRESSION, MSG_GET,
pCompData);
```

`pCompData` = A pointer to a `TW_JPEGCOMPRESSION` structure.

Valid States

4 through 6

Description

Causes the Source to return the parameters that will be used during the compression of data using the JPEG algorithms.

All the information that is reported by the `MSG_GET` operation will be available in the header portion of the JPEG data. Transferring JPEG-compressed data through memory buffers is slightly different than other types of buffered transfers. The difference is that the JPEG-compressed image data will be prefaced by a block of uncompressed information—the JPEG header. This header information contains all the information that is returned from the `MSG_GET` operation. The compressed image information follows the header. The Source should return the header information in the first transfer. The compressed image data will then follow in the second through the final buffer. If the application is allocating the buffers, it should assure that the buffer size for transfer of the header is large enough to contain the complete header.

Application

The application allocates the `TW_JPEGCOMPRESSION` structure.

Source

Fill `pCompData` with the parameters that will be applied to the next JPEG-compression operation. The Source must allocate memory for the contents of the pointer fields pointed to within the structure (i.e. QuantTable, HuffmanDC, and HuffmanAC).

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL       /* Source does not support JPEG */
                          /* data compression            */

   TWCC_SEQERROR          /* Operation invoked in invalid */
                          /* state                        */
```

See Also

`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET`

Capability - `ICAP_COMPRESSION`

DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_JPEGCOMPRESSION, MSG_GETDEFAULT,
pCompData);
```

`pCompData` = A pointer to a `TW_JPEGCOMPRESSION` structure.

Valid States

4 through 6

Description

Causes the Source to return the power-on default values applied to JPEG-compressed data transfers.

Application

The application allocates the `TW_JPEGCOMPRESSION` structure.

Source

Fill in `pCompData` with the power-on default values. The Source must allocate memory for the contents of the pointer fields pointed to within the structure (i.e. `QuantTable`, `HuffmanDC` and `HuffmanAC`). The Source should maintain meaningful default values.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL        /* Source does not support JPEG */
                           /* data compression            */

   TWCC_SEQERROR           /* Operation invoked in invalid */
                           /* state                       */
```

See Also

`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET`

Capability - `ICAP_COMPRESSION, ICAP_JPEGQUALITY`

DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_JPEGCOMPRESSION, MSG_RESET,
pCompData);
```

`pCompData` = A pointer to a `TW_JPEGCOMPRESSION` structure.

Valid States

4 only

Description

Return the Source to using its power-on default values for JPEG-compressed transfers.

Application

No special action. May want to perform a `MSG_GETDEFAULT` if you're curious what the new values might be.

Source

Use your power-on default values for all future JPEG-compressed transfers. The Source should maintain meaningful default values for all parameters.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL        /* Source does not support JPEG */
                           /* data compression           */

   TWCC_SEQERROR           /* Operation invoked in invalid */
                           /* state                      */
```

See Also

`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET`

Capability - `ICAP_COMPRESSION`, `ICAP_JPEGQUALITY`

DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_JPEGCOMPRESSION, MSG_SET,
pCompData);
```

`pCompData` = A pointer to a `TW_JPEGCOMPRESSION` structure.

Valid States

4 only

Description

Allows the application to configure the compression parameters to be used on all future JPEG-compressed transfers during the current session. The application should have already established that the requested values are supported by the Source.

Application

Fill `pCompData`. Write `TWON_DONTCARE16` into the numeric fields that don't matter to the application. Write `NULL` into the table fields that should use the default tables as defined by the JPEG specification.

Source

Adopt the requested values for use with all future JPEG-compressed transfers. If a value does not exactly match an available value, match the value as closely as possible and return `TWRC_CHECKSTATUS`. If the value is beyond the range of available values, clip to the nearest value and return `TWRC_FAILURE`/`TWCC_BADVALUE`.

Return Codes

```
TWRC_SUCCESS

TWRC_CHECKSTATUS

TWRC_FAILURE

   TWCC_BADPROTOCOL        /* Source does not support JPEG */
                           /* data compression             */

   TWCC_BADVALUE           /* illegal value specified      */

   TWCC_SEQERROR           /* Operation invoked in invalid */
                           /* state                        */
```

See Also

`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET`

Capability - `ICAP_COMPRESSION, ICAP_JPEGQUALITY`

DG_IMAGE / DAT_PALETTE8 / MSG_GET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_PALETTE8, MSG_GET, pPalette);
```

`pPalette` = A pointer to a `TW_PALETTE8` structure.

Valid States

4 through 6

Description

This operation causes the Source to report its current palette information. The application should assure that the Source can provide palette information by invoking a `MSG_GET` operation on `ICAP_PIXELTYPE` and checking for `TWPT_PALETTE`. If this pixel type has not been established as the type to be used for future acquisitions, the Source should respond with its default palette.

To assure that the palette information is wholly accurate, the application should invoke this operation immediately after completion of the image transfer. The Source may perform palette optimization during acquisition of the data and the palette it reports before the transfer will differ from the one available afterwards.

(In general, the `DAT_PALETTE8` operations are specialized to deal with 8-bit data, whether grayscale or color (8-bit or 24-bit). Most current devices provide data with this bit depth. These operations allow the application to inquire a Source's support for palette color data and set up a palette color transfer. See Chapter 8, "Data Types and Data Structures" for the definitions and data structures used to describe palette color data within TWAIN.)

Application

The application should allocate the pPalette structure for the Source.

Source

Fill `pPalette` with the current palette. If no palette has been specified or calculated, use the Source's default palette (which may coincidentally be the current or default system palette).

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL          /* Source does not support      */
                             /* palette color transfers      */

   TWCC_SEQERROR             /* Operation invoked in invalid */
                             /* state                        */
```

See Also

```
DG_IMAGE / DAT_PALETTE8 / MSG_GETDEFAULT
DG_IMAGE / DAT_PALETTE8 / MSG_RESET
DG_IMAGE / DAT_PALETTE8 / MSG_SET
```

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_PALETTE8 / MSG_GETDEFAULT

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_PALETTE8, MSG_GETDEFAULT,
pPalette);
```

`pPalette` = A pointer to a `TW_PALETTE8` structure.

Valid States

4 through 6

Description

This operation causes the Source to report its power-on default palette.

Application

The application should allocate the `pPalette` structure for the Source.

Source

Fill `pPalette` with the default palette.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL        /* Source does not support      */
                           /* palette color transfers      */

   TWCC_SEQERROR           /* Operation invoked in invalid */
                           /* state                        */
```

See Also

```
DG_IMAGE / DAT_PALETTE8 / MSG_GET
DG_IMAGE / DAT_PALETTE8 / MSG_RESET
DG_IMAGE / DAT_PALETTE8 / MSG_SET
```

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_PALETTE8 / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_PALETTE8, MSG_RESET, pPalette);
```

`pPalette` = A pointer to a `TW_PALETTE8` structure.

Valid States

4 only

Description

This operation causes the Source to dispose of any current palette it has and to use its default palette for the next palette transfer. A Source that always performs palette optimization may not use the default palette for the next transfer, but should dispose of its current palette and adopt the default palette for the moment, anyway. The application can check the actual palette information by invoking a `MSG_GET` operation immediately following the image transfer.

Application

The application should allocate the pPalette structure for the Source.

Source

Fill `pPalette` with the default palette and use the default palette for the next palette transfer.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL          /* Source does not support      */
                             /* palette color transfers      */

   TWCC_SEQERROR             /* Operation invoked in invalid */
                             /* state                        */
```

See Also

`DG_IMAGE / DAT_PALETTE8 / MSG_GET`
`DG_IMAGE / DAT_PALETTE8 / MSG_GETDEFAULT`
`DG_IMAGE / DAT_PALETTE8 / MSG_SET`

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_PALETTE8 / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_PALETTE8, MSG_SET, pPalette);
```

`pPalette` = A pointer to a `TW_PALETTE8` structure.

Valid States

4 only

Description

This operation requests that the Source adopt the specified palette for use with all subsequent palette transfers. The application should be careful to supply a palette that matches the bit depth of the Source. The Source is not required to adopt this palette. The application should be careful to check the return value from this operation.

Application

Fill `pPalette` with the desired palette. If writing grayscale information, write the same data into the `Channel1`, `Channel2`, and `Channel3` fields of the `Colors` array. If `NumColors != 256`, fill the unused array elements with minimum ("black") values.

Source

The Source should not return `TWRC_SUCCESS` unless it will actually use the requested palette. The Source should not modify the palette in any way until the transfer is complete. The palette should be used for all remaining palette transfers for the duration of the session.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL          /* Source does not support      */
                             /* palette color transfers      */

   TWCC_SEQERROR             /* Operation invoked in invalid */
                             /* state                        */
```

See Also

`DG_IMAGE / DAT_PALETTE8 / MSG_GET`
`DG_IMAGE / DAT_PALETTE8 / MSG_GETDEFAULT`
`DG_IMAGE / DAT_PALETTE8 / MSG_RESET`

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_RGBRESPONSE / MSG_RESET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_RGBRESPONSE, MSG_RESET,
pResponse);
```

`pResponse` = A pointer to a `TW_RGBRESPONSE` structure.

Valid States

4 only

Description

Causes the Source to use its "identity" response curves for future RGB transfers. The identity curve causes no change in the values of the captured data when it is applied. (Note that resetting the curves for RGB data **does not** reset any `MSG_SET` curves for other pixel types).

Note: The `DAT_RGBRESPONSE` operations allow the application to specify the transfer curves that the Source should apply to the RGB data it acquires. The Source should not support these operations unless it can provide data of pixel type `TWPT_RGB`. The Source need not maintain actual "identity response curves" for use with the `MSG_RESET` operation—once reset, the Source should transfer the RGB data as acquired from the Source. The application should be sure that the Source supports these operations before invoking them. The operations should only be invoked when the active pixel type is RGB (`TWPT_RGB`). See Chapter 8, "Data Types and Data Structures" for information about the definitions and data structures used to describe the RGB response curve within TWAIN.

Application

No special action.

Source

Apply the identity response curve to all future RGB transfers. This means that the Source will transfer the RGB data exactly as acquired from the device.

Return Codes

`TWRC_SUCCESS`

`TWRC_FAILURE`

```
   TWCC_BADPROTOCOL        /* Source does not support RGB  */
                           /* response curves              */
   TWCC_BADVALUE           /* Current pixel type is not    */
                           /* TWPT_RGB                     */
   TWCC_SEQERROR           /* Operation invoked in invalid */
                           /* state                        */
```

See Also

`DG_IMAGE / DAT_RGBRESPONSE / MSG_SET`

Capability - `ICAP_PIXELTYPE`

DG_IMAGE / DAT_RGBRESPONSE / MSG_SET

Call

```
DSM_Entry(pOrigin, pDest, DG_IMAGE, DAT_RGBRESPONSE, MSG_SET,
pResponse);
```

`pResponse` = A pointer to a `TW_RGBRESPONSE` structure.

Valid States

4 only

Description

Causes the Source to transform any RGB data according to the response curves specified by the application.

Application

Fill all three elements of the response curve with the response curve data you want the Source to apply to future RGB transfers. The application should consider writing the same values into each element of the same index to minimize color shift problems.

The Source may not support this operation. The application should ensure that the current pixel type is `TWPT_RGB` and examine the return code from this operation.

Source

Apply the specified response curves to all future RGB transfers.

Return Codes

```
TWRC_SUCCESS

TWRC_FAILURE

   TWCC_BADPROTOCOL         /* Source does not support color */
                            /* response curves               */

   TWCC_BADVALUE            /* Current pixel type is not RGB */

   TWCC_SEQERROR            /* Operation invoked in invalid  */
                            /* state                         */
```

See Also

`DG_IMAGE / DAT_RGBRESPONSE / MSG_RESET`

Capability - `ICAP_PIXELTYPE`

8

Data Types and Data Structures

Chapter Contents

This section of the Specification is definitive and authoritative in its description of the TWAIN namespace and the numeric ids that go with each name in that space. If a discrepancy is found between this chapter and any C/C++ TWAIN.H definition file then the TWAIN.H file must be corrected.

A `TWAIN.H` definition file is provided with this toolkit, this file is specific to C/C++ solutions.

If a definition file for a previously unsupported language is submitted to the TWAIN Working Group, and if it passes review, then the salient points needed to recreate it will be added to this chapter. A definition file cannot be called TWAIN or said to support TWAIN unless it can be completely created following the information in this chapter.

Naming Conventions

Data Structures, Variables, Pointers and Handles

Data structures referenced by pData parameter in DSM_Entry calls

Are prefixed by `TW_` and followed by a descriptive name, in upper case. The name typically matches the call's `DAT` parameter.

Example: `TW_USERINTERFACE`

Fields in data structures (not containing pointers or handles)

Typically, begin with a capital letter followed by mixed upper and lower case letters.

Example: The `MinBufSize`, `MaxBufSize`, and `Preferred` fields in which are in the `TW_SETUPMEMXFER` structure.

Fields in data structures that contain pointers or handles

Name starts with lower case "p" or "h" for pointer or handle followed by a typical field name with initial capital then mixed case characters.

Example: `pData`, `hContainer`

Constants and Types

General-use constants

Are prefixed by `TWON_` followed by the description of the constant's meaning.

Example: `TWON_ARRAY`, `TWON_ONEVALUE`

Specific-use constants

Are prefixed with `TWxx_` where `xx` are two letters identifying the group to which the constant belongs.

Example: `TWTY_INT16`, `TWTY_STR32` are constants of the group `TW Types`

Common data types

Rather than use the `int`, `char`, `long`, and other. types with their variations between compilers, TWAIN defines a group of types that are used to cast each data item used by the protocol. Types are prefixed and named exactly the same as TWAIN data structures, `TW_` followed by a descriptive name, all in upper case characters.

Example: `TW_UINT32`, `TW_HANDLE`

TWAIN.H internal constants

Starting with TWAIN 2.0 internal constants that are of special interest to TWAIN.H itself are used to improve the readability and maintainability of the file. They are prefixed with TWH_.

Custom Constants

Applications and Sources may define their own private (custom) constant identifiers for any existing constant group by assigning the constant a value greater than or equal to 0x8000. They may also define any new desired custom constant group. The consuming entity should check the originating entity's `TW_IDENTITY.ProductName` when encountering a constant value greater than or equal to `0x8000` to see whether it can be recognized as a custom constant. Sources and applications should not assume that all entities will have such error checking built in, however.

The following are operation identifiers:

Data Groups	Prefixed with `DG_`
Data Argument Types	Prefixed with `DAT_`
Messages	Prefixed with `MSG_`
Return codes	Prefixed with `TWRC_`
Condition codes	Prefixed with `TWCC_`
General capabilities	Prefixed with `CAP_`

Image-specific capabilities	Prefixed with `ICAP_`
Audio-specific capabilities	Prefixed with `ACAP_`

As a general note, whenever the application or the Source allocates a TWAIN data structure, it should fill all the fields it is instructed to fill and write the default value (if one is specified) into any field it is not filling. If no default is specified, fill the field with the appropriate `TWON_DONTCAREXX` constant where `XX` describes the size of the field in bits (bytes, in the case of strings). The `TWON_` constants are described at the end of this chapter and defined in the `TWAIN.H` file.

Some fields return a value of -1 when the data to be returned is ambiguous or unknown. Applications and Sources must look for these special cases, especially when allocating memory. Examples of Fields with -1 values are found in `TW_PENDINGXFERS (Count)`, `TW_SETUPMEMXFER (MaxBufSize)` and `TW_IMAGEINFO (ImageWidth and ImageLength)`.

Platform Dependent Definitions and Typedefs

Single Compile

The TWAIN include file must only be referenced once for any compiled module. This is achieved by bracketing the contents of the entire file with the following:

```
#ifndef TWAIN
#define TWAIN
   …contents of the TWAIN include file…
#endif /* TWAIN */
```

Platform Identification Macros

TWAIN supports multiple operating system platforms; it also can run with multiple compilers. The following macros are intended to help organize these combinations. Note that they focus more on the compilers than the platforms (cf: for the purposes of TWAIN GNU works the same on all operating systems).

```
/* Microsoft C/C++ Compiler */
#if defined(WIN32) || defined(WIN64) || defined (_WINDOWS)
    #define TWH_CMP_MSC
    #if  defined(_WIN64) || defined(WIN64)
      #define TWH_64BIT
    #elif defined(WIN32) || defined(_WIN32)
      #define TWH_32BIT
    #endif

/* Apple Compiler (which is GNU now) */
#elif defined(__APPLE__)
    #define TWH_CMP_XCODE
    #define TWH_32BIT

/* GNU C/C++ Compiler */
#elif defined(__GNUC__)
    #define TWH_CMP_GNU
    #if defined(__alpha__)\
        ||defined(__ia64__)\
        ||defined(__ppc64__)\
        ||defined(__s390x__)\
        ||defined(__x86_64__)
      #define TWH_64BIT
    #else
      #define TWH_32BIT
    #endif

/* Borland C/C++ Compiler */
#elif defined(__BORLAND__)
    #define TWH_CMP_BORLAND
    #define TWH_32BIT
/* Unrecognized */
#else
    #error Unrecognized compiler
#endif
```

Platform Specific Typedefs

These definitions and typedefs are dependent on the compiler.

```
/* Win32 and Win64 systems */
#if defined(TWH_CMP_MSC) | defined(TWH_CMP_BORLAND)
    typedef HANDLE  TW_HANDLE;
    typedef LPVOID  TW_MEMREF;
    typedef UINT_PTR TW_UINTPTR;

/* MacOS/X... */
#elif defined(TWH_CMP_XCODE)
    #define PASCAL   pascal
    #define FAR
    typedef Handle   TW_HANDLE;
    typedef char    *TW_MEMREF;

    #ifdef TWH_32BIT
      //32 bit GNU
      typedef unsigned long      TW_UINTPTR;
    #else
      //64 bit GNU
      typedef unsigned long long TW_UINTPTR;
    #endif

/* Everything else... */
#else
    #define PASCAL
    #define FAR
    typedef void* TW_HANDLE;
    typedef void* TW_MEMREF;
    typedef unsigned char BYTE;

    #ifdef TWH_32BIT
      //32 bit GNU
      typedef unsigned long      TW_UINTPTR;
    #else
      //64 bit GNU
      typedef unsigned long long TW_UINTPTR;
    #endif
#endif
```

Platform Specific Byte Packing (Alignment)

In addition to the dependent definitions and typedefs TWAIN requires that the data alignment of all structures occurs on an agreed upon boundary. This prevents mismatches in the alignment of the data between the driver, the source manager and the application.

```
/* Set the packing: this occurs before any structures are defined */
#ifdef TWH_CMP_MSC
   #pragma pack (push, before_twain)
   #pragma pack (2)
#elif defined(TWH_CMP_GNU)
   #if defined(__APPLE__) /* cf: Mac version of TWAIN.h */
      #pragma options align = power
   #else
      #pragma pack (push, before_twain)
      #pragma pack (2)
   #endif
#elif defined(TWH_CMP_BORLAND)
   #pragma option -a2
#endif
/* Restore the previous packing alignment: this occurs after all
structures are defined */
#ifdef TWH_CMP_MSC
   #pragma pack (pop, before_twain)
```

```
#elif defined(TWH_CMP_GNU)
   #if defined(__APPLE__) /* cf: Mac version of TWAIN.h */
      #pragma options align = reset
   #else
      #pragma pack (pop, before_twain)
#endif
#elif defined(TWH_CMP_BORLAND)
   #pragma option –a.
#endif
```

Definitions of Common Types

String types

```
#if defined(__APPLE__)/* cf: Mac version of TWAIN.h */
   typedef unsigned char    TW_STR32[34],     FAR *pTW_STR32;
   typedef unsigned char    TW_STR64[66],     FAR *pTW_STR64;
   typedef unsigned char    TW_STR128[130],   FAR *pTW_STR128;
   typedef unsigned char    TW_STR255[256],   FAR *pTW_STR255;
#else
   typedef char    TW_STR32[34],     FAR *pTW_STR32;
   typedef char    TW_STR64[66],     FAR *pTW_STR64;
   typedef char    TW_STR128[130],   FAR *pTW_STR128;
   typedef char    TW_STR255[256],   FAR *pTW_STR255;
#endif
```

On Windows: These include room for the strings and a `NULL` character.

On Macintosh: These include room for a length byte followed by the string.

Note: The `TW_STR255` must hold less than 256 characters so the length fits in the first byte on Macintosh.

Numeric types

```
typedef char                  TW_INT8, FAR *pTW_INT8;
typedef short                 TW_INT16, FAR *pTW_INT16;
#if defined(__APPLE__)        /* cf: Mac version of TWAIN.h */
      typedef int             TW_INT32, FAR *pTW_INT32;
#else
      typedef long            TW_INT32, FAR *pTW_INT32;
#endif
typedef unsigned char         TW_UINT8, FAR *pTW_UINT8;
typedef unsigned short        TW_UINT16, FAR *pTW_UINT16;
#if defined(__APPLE__)        /* cf: Mac version of TWAIN.h */
      typedef unsigned int          TW_UINT32, FAR *pTW_UINT32;
#else
      typedef unsigned long      TW_UINT32, FAR *pTW_UINT32;
```

```
#endif
typedef unsigned short        TW_BOOL, FAR *pTW_BOOL;
```

Fixed point structure type

```
typedef struct {
            TW_INT16    Whole;
            TW_UINT16   Frac;
} TW_FIX32, FAR *_pTW_FIX32;
```

Note: In cases where the data type is smaller than `TW_UINT32`, the data resides in the lower word.

Data Structure Definitions

This section provides descriptions of the data structure definitions.

TW_ARRAY

```
typedef struct {
   TW_UINT16      ItemType;
   TW_UINT32      NumItems;
   TW_UINT8       ItemList[1];
} TW_ARRAY, FAR * pTW_ARRAY;
```

Used by

`TW_CAPABILITY` structure (when `ConType` field specifies `TWON_ARRAY`)

Description

This structure stores a group of associated individual values which, when taken as a whole, describes a single "value" for a capability. The values need have no relationship to one another aside from being used to describe the same "value" of the capability. Such an array of values is useful to describe the `CAP_SUPPORTEDCAPS` list. This structure is used as a member of `TW_CAPABILITY` structures. Since this structure does not, therefore, exist "stand-alone" it is identified by a `TWON_xxxx` constant rather than a `DAT_xxxx`. This structure is related in function and purpose to `TW_ENUMERATION`, `TW_ONEVALUE`, and `TW_RANGE`.

Field Descriptions

`ItemType`	The type of items in the array. The type is indicated by the constant held in this field. The constant is of the kind `TWTY_xxxx`. All items in the array have the same size.
`NumItems`	How many items are in the array.
`ItemList[1]`	This is the array. One value resides within each element of the array. Space for the array is not allocated inside this structure. The ItemList value is simply a placeholder for the start of the actual array, which must be allocated when the container is allocated. Remember to typecast the allocated array, as well as references to the elements of the array, to the type indicated by the constant in ItemType. Ex: To set an item in a CAP_SUPPORTEDCAPS array… ((TW_UINT16*)twarray.ItemList)[2] = ICAP_XFERMECH;

TW_AUDIOINFO

```
typedef struct {
   TW_STR255     Name;
   TW_UINT32     Reserved;
} TW_AUDIOINFO, FAR * pTW_AUDIOINFO;
```

Used by

The `DG_AUDIO / DAT_AUDIOINFO / MSG_GET` operation

Description

Field Descriptions

`Name`	Name of audio data
`Reserved`	Reserved space

TW_CALLBACK

```
typedef struct  {
      TW_MEMREF            CallBackProc;
      #if defined(__APPLE__)     /* cf: Mac version of TWAIN.h */
               TW_MEMREF   RefCon;
      #else
               TW_UINT32   RefCon;
      #endif
      TW_INT16             Message;
} TW_CALLBACK, FAR * pTW_CALLBACK;
```

Used by

```
DG_CONTROL / DAT_CALLBACK / MSG_REGISTER_CALLBACK
DG_CONTROL / DAT_CALLBACK / MSG_INVOKE_CALLBACK
```

Description

Used in Callback mechanism for sending messages from the Source to the Application. Applications version 2.2 or higher must use `TW_CALLBACK2`.

Field Descriptions

`CallBackProc`	The callback function's entry point, used by `MSG_REGISTER_CALLBACK`.
`RefCon`	An application defined reference constant.
`Message`	`Initialized` to any valid `DG_CONTROL / DAT_NULL` message.

TW_CALLBACK2

```
typedef struct  {
   TW_MEMREF   CallBackProc;
   TW_UINTPTR  RefCon;
   TW_INT16    Message;
} TW_CALLBACK2, FAR * pTW_CALLBACK2;
```

Used by

`DG_CONTROL / DAT_CALLBACK2 / MSG_REGISTER_CALLBACK`

Description

Used in the Callback mechanism for sending messages from the Source to the Application.

Field Descriptions

`CallBackProc`	The callback function's entry point, used by `MSG_REGISTER_CALLBACK`.
`RefCon`	An application defined reference constant. It has a different size on different platforms.
`Message`	`Initialized` to any valid `DG_CONTROL / DAT_NULL` message.

TW_CAPABILITY

```
typedef struct {
    TW_UINT16      Cap;
    TW_UINT16      ConType;
    TW_HANDLE      hContainer;
} TW_CAPABILITY, FAR * pTW_CAPABILITY;
```

Used by

```
DG_CONTROL / DAT_CAPABILITY / MSG_GET
DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT
DG_CONTROL / DAT_CAPABILITY / MSG_GETDEFAULT
DG_CONTROL / DAT_CAPABILITY / MSG_RESET
DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL
DG_CONTROL / DAT_CAPABILITY / MSG_SET
```

Description

Used by an application either to get information about, or control the setting of a capability. The first field identifies the capability being negotiated (e.g., `ICAP_BRIGHTNESS`). The second specifies the format of the container (e.g., `TWON_ONEVALUE`). The third is a handle (`HGLOBAL` under Microsoft Windows) to the container itself.

The application always sets the Cap field. On `MSG_SET`, the application also sets the `ConType` and hContainer fields. On `MSG_RESET`, `MSG_RESETALL`, `MSG_GET`, `MSG_GETCURRENT`, and `MSG_GETDEFAULT`, the source fills in the `ConType` and `hContainer` fields.

It is always the application's responsibility to free the container when it is no longer needed. On a `MSG_GET`, `MSG_GETCURRENT`, or `MSG_GETDEFAULT`, the source allocates the container but ownership passes to the application. On a `MSG_SET`, the application provides the container either by allocating it or by re-using a container created earlier.

On a `MSG_SET`, the Source must not modify the container and it must copy any data that it wishes to retain.

Field Descriptions

`Cap`	The numeric designator of the capability (of the form `CAP_xxxx`, `ICAP_xxxx`, or `ACAP_xxxx`). e.g. `ICAP_BRIGHTNESS`. A list of these can be found in Chapter 10, "Capabilities" and in the `TWAIN.H` file.
`ConType`	The type of the container referenced by hContainer. The container structure will be one of four types: `TWON_ARRAY`, `TWON_ENUMERATION`, `TWON_ONEVALUE`, or `TWON_RANGE`. One of these values, which types the container, should be entered into this field by whichever TWAIN entity fills in the container. When the application wants to set (`MSG_SET`) the Source's capability, the application must fill in this field. When the application wants to get (`MSG_GET`) capability information from the Source, the Source must fill in this field.
`hContainer`	References the container structure where detailed information about the capability is stored. When the application wants to set (`MSG_SET`) the Source's capability, the application must provide the hContainer. When the application wants to get (`MSG_GET`) the Source's capability information, the Source must allocate the space for the container. In either case, the application must release this space.

TW_CIECOLOR

```
typedef struct {
  TW_UINT16         ColorSpace
  TW_INT16          LowEndian;
  TW_INT16          DeviceDependent;
  TW_INT32          VersionNumber;
  TW_TRANSFORMSTAGE  StageABC;
  TW_TRANSFORMSTAGE  StageLMN;
  TW_CIEPOINT       WhitePoint;
  TW_CIEPOINT       BlackPoint;
  TW_CIEPOINT       WhitePaper;
  TW_CIEPOINT       BlackInk;
  TW_FIX32          Samples[1];
} TW_CIECOLOR, FAR * pTW_CIECOLOR;
```

Used by

DG_IMAGE / DAT_CIECOLOR / MSG_GET

Description

Defines the mapping from an RGB color space device into CIE 1931 (XYZ) color space. For more in-depth information, please reference the PostScript Language Reference Manual, Second Edition, pp. 173-193. Note that the field names do not follow the conventions used elsewhere within TWAIN.

If the Source can provide TWPT_CIEXYZ, it must support all operations on this structure.

Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

Field Descriptions

ColorSpace	Defines the original color space that was transformed into CIE XYZ. Use a constant of type TWPT_xxxx. This value is not set-able by the application. Application should write TWON_DONTCARE16 into this on a MSG_SET.
LowEndian	Used to indicate which data byte is taken first. If zero, then high byte is first. If non-zero, then low byte is first.
DeviceDependent	If non-zero then color data is device-dependent and only ColorSpace is valid in this structure.
VersionNumber	Version of the color space descriptor specification used to define the transform data. The current version is zero.
StageABC	Describes parametrics for the first stage transformation of the Postscript Level 2 CIE color space transform process.
StageLMN	Describes parametrics for the first stage transformation of the Postscript Level 2 CIE color space transform process.
WhitePoint	Values that specify the CIE 1931 (XYZ space) tri-stimulus value of the diffused white point.

`BlackPoint`	Values that specify the CIE 1931 (XYZ space) tri-stimulus value of the diffused black point.
`WhitePaper`	Values that specify the CIE 1931 (XYZ space) tri-stimulus value of ink-less "paper" from which the image was acquired.
`BlackInk`	Values that specify the CIE 1931 (XYZ space) tri-stimulus value of solid black ink on the "paper" from which the image was acquired.
`Samples[1]`	Optional table look-up values used by the decode function. Samples are ordered sequentially and end-to-end as A, B, C, L, M, and N.

TW_CIEPOINT

```
typedef struct {
  TW_FIX32      X;
  TW_FIX32      Y;
  TW_FIX32      Z;
} TW_CIEPOINT, FAR * pTW_CIEPOINT;
```

Used by

Embedded in the `TW_CIECOLOR` structure

Description

Defines a CIE XYZ space tri-stimulus value.

Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

Field Descriptions

X	First tri-stimulus value of the CIE space representation.
Y	Second tri-stimulus value of the CIE space representation.
Z	Third tri-stimulus value of the CIE space representation.

TW_CUSTOMDSDATA

```
typedef struct {
   TW_UINT32       InfoLength;/* Length (in bytes) of data */
   TW_HANDLE       hData;     /* Handle to data */
} TW_CUSTOMDSDATA, FAR *_pTW_CUSTOMDSDATA;
```

Used by

```
DG_CONTROL / DAT_CUSTOMDSDATA / MSG_GET
DG_CONTROL / DAT_CUSTOMDSDATA / MSG_SET
```

Description

Allows for a data source and application to pass custom data to each other.

The format of the data contained in `hData` will be data source specific and will not be defined by the TWAIN API. This structure will be used by an application to query the data source for its current settings, and to archive them to disk. Although the format for this custom data is not defined by TWAIN, source implementers are encouraged to use a ASCII representation for the custom data to be used for settings archival. A Windows INI style format would be easy to implement and allow for additional features to be added without breaking backwards compatibility.

It is also recommended that source vendors embed basic source revision and vendor ID information in the `hData` body so they can determine if the structure being passed to the data source is correct.

Note: 1.x versions of the specification have shown the following structure.

```
typedef struct {
   TW_UINT32       InfoLength;     /* Length (in bytes) of data */
   TW_UINT8        InfoData[1];    /* Array (Length) bytes long */
} TW_CUSTOMDSDATA, FAR *_pTW_CUSTOMDSDATA;
```

Starting with TWAIN 2.0 only the structure with `hData` is considered correct. If both a driver and an application are reporting version 2 by examining the `TW_IDENTITY`.`SupportedGroups` for `DF_APP2` and `DF_DS2`, then both may assume that `hData` is in use. It is not sufficient to check the `TW_IDENTITY`. `ProtocolMajor` field for a value greater than or equal to 2.

For older drivers and applications it's expected that most developers have followed the use of `hData` from the `TWAIN.H` file, however, good defensive programming recommends at least attempting to anticipate both forms. On Window systems the developer can use `GlobalSize()` to test if the `TW_CUSTOMDSDATA` structure is greater than sizeof (`TW_CUSTOMDSDATA`), which would suggest that `hData` isn't a pointer, but is the actual beginning of the data. Other calls like `GlobalLock()` and `IsBadReadPtr()` can be used to check the validity of the pointer in `hData`. No system is perfect, but it should be possible to cover most drivers and applications this way.

Field Descriptions

`InfoLength`	`Length`, in bytes, of data
`hData`	Handle to memory containing `InfoLength` bytes of data

TW_DECODEFUNCTION

```
typedef struct {
  TW_FIX32      StartIn;
  TW_FIX32      BreakIn;
  TW_FIX32      EndIn;
  TW_FIX32      StartOut;
  TW_FIX32      BreakOut;
  TW_FIX32      EndOut;
  TW_FIX32      Gamma;
  TW_FIX32      SampleCount;
} TW_DECODEFUNCTION, FAR * pTW_DECODEFUNCTION;
```

Used by

Embedded in the `TW_TRANSFORMSTAGE` structure that is embedded in the `TW_CIECOLOR` structure

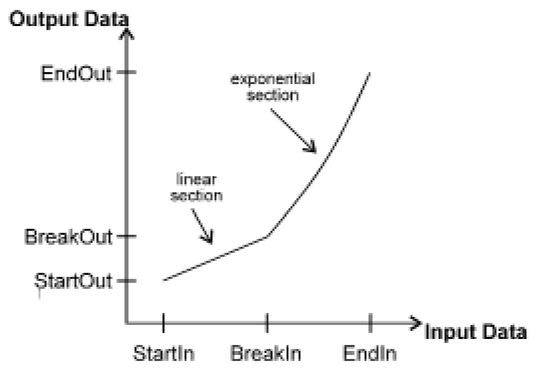
Description

Defines the parameters used for channel-specific transformation. The transform can be described either as an extended form of the gamma function or as a table look-up with linear interpolation.

Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

Field Descriptions

`StartIn`	Starting input value of the extended gamma function. Defines the minimum input value of channel data.
`BreakIn`	Ending input value of the extended gamma function. Defines the maximum input value of channel data.
`EndIn`	The input value at which the transform switches from linear transformation/ interpolation to gamma transformation.
`StartOut`	Starting output value of the extended gamma function. Defines the minimum output value of channel data.
`BreakOut`	Ending output value of the extended gamma function. Defines the maximum output value of channel data.
`EndOut`	The output value at which the transform switches from linear transformation/ interpolation to gamma transformation.
`Gamma`	Constant value. The exponential used in the gamma function.
`SampleCount`	The number of samples in the look-up table. Includes the values of StartIn and EndIn. Zero-based index (actually, number of samples - 1). If zero, use extended gamma, otherwise use table look-up.



Extended Gamma Parameters

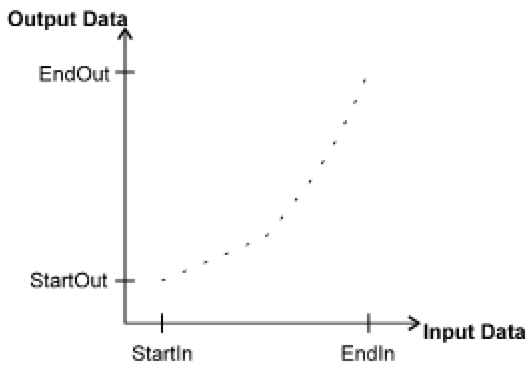


Table Look-up Parameters

TW_DEVICEEVENT

```
typedef struct {
  TW_UINT32     Event;
  TW_STR255     DeviceName;
  TW_UINT32     BatteryMinutes;        // Battery Minutes Remaining
  TW_INT16      BatteryPercentage;      // Battery Percentage Remaining
  TW_INT32      PowerSupply;           // Power Supply
  TW_FIX32      XResolution;           // Resolution
  TW_FIX32      YResolution;           // Resolution
  TW_UINT32     FlashUsed2;            // Flash Used2
  TW_UINT32     AutomaticCapture;      // Automatic Capture
  TW_UINT32     TimeBeforeFirstCapture; // Automatic Capture
  TW_UINT32     TimeBetweenCaptures;    // Automatic Capture
} TW_DEVICEEVENT, FAR * pTW_DEVICEEVENT;
```

Used by

DG_CONTROL / DAT_DEVICEEVENT / MSG_GET

Description

Provides information about the Event that was raised by the Source. The Source should only fill in those fields applicable to the Event. The Application must only read those fields applicable to the Event.

Field Descriptions

Event	One of the TWDE_xxxx values. Defines event that has taken place.
DeviceName	The name of the device that generated the event.
Valid for TWDE_BATTERYCHECK only	
BatteryMinutes	Minutes of battery power remaining.
BatteryPercentage	Percentage of battery power remaining.
Valid for TWDE_POWERSUPPLY only	
PowerSupply	Current power supply in use.
Valid for TWDE_RESOLUTION only	
XResolution	Current X Resolution.
YResolution	Current Y Resolution.
Valid for TWDE_FLASHUSED2 only	
FlashUsed2	Current flash setting.
Valid for TWDE_AUTOMATICCAPTURE only	
AutomaticCapture	Number of images camera will capture.
TimeBeforeFirstCapture	Number of seconds before first capture.
TimeBetweenCaptures	Hundredths of a second between captures.

TW_ENTRYPOINT

```
typedef struct {
   TW_UINT32 Size;
   DSM_ENTRY DSM_Entry;
   DSM_MEMALLOCATE DSM_MemAllocate;
   DSM_MEMFREE DSM_MemFree;
   DSM_MEMLOCK DSM_MemLock;
   DSM_MEMUNLOCK DSM_MemUnlock;
} TW_ENTRYPOINT, FAR * pTW_ENTRYPOINT;
```

Used by

```
DG_CONTROL / DAT_ENTRYPOINT / MSG_GET
DG_CONTROL / DAT_ENTRYPOINT / MSG_SET
```

Description

Provides entry points required by TWAIN 2.0 Applications and Sources.

Field Descriptions

`Size`	Size of the structure in bytes. The application must set this before calling `MSG_GET`. The Source should examine this when processing a `MSG_SET`.
`DSM_Entry`	A pointer to the `DSM_Entry` function. TWAIN 2.0 Sources must use this value instead of getting it themselves.
`DSM_MemAllocate`	A pointer to the memory allocation function, taking the form

`TW_HANDLE PASCAL DSM_MemAllocate (TW_UINT32).`

`DSM_MemFree`	A pointer to the memory free function, taking the form

`void PASCAL DSM_MemAllocate (TW_HANDLE)`

`DSM_MemLock`	A pointer to the memory lock function, taking the form

`TW_MEMREF PASCAL DSM_MemAllocate (TW_HANDLE)`

`DSM_MemUnlock`	A pointer to the memory unlock function, taking the form `void PASCAL DSM_MemUnlock (TW_HANDLE)`

TW_ELEMENT8

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

Used by

Embedded in the TW_GRAYRESPONSE, TW_PALETTE8 and TW_RGBRESPONSE structures

Description

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

Field Descriptions

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

TW_ENUMERATION

```
typedef struct {
  TW_UINT16     ItemType;
  TW_UINT32     NumItems;
  TW_UINT32     CurrentIndex;
  TW_UINT32     DefaultIndex;
  TW_UINT8      ItemList[1];
} TW_ENUMERATION, FAR * pTW_ENUMERATION;
```

Used by

TW_CAPABILITY structure (when ConType field specifies TWON_ENUMERATION)

Description

An enumeration stores a list of individual values, with one of the items designated as the current value.

There is no required order to the values in the list. However, it is recommended that the data source's GUI show the values in the order that they have been negotiated by the application.

It is also recommended, but not required, that a MSG_GET operation reflects the same order as the last MSG_SET operation for that capability.

Data sources may opt to always order some enumerated lists, like ICAP_XRESOLUTION, so that the values are presented on the GUI in numerical order.

This structure is related in function and purpose to TW_ARRAY, TW_ONEVALUE, and TW_RANGE.

Field Descriptions

ItemType	The type of items in the enumerated list. The type is indicated by the constant held in this field. The constant is of the kind TWTY_xxxx. All items in the array have the same size.
NumItems	How many items are in the enumeration.
CurrentIndex	The item number, or index (zero-based) into ItemList[], of the "current" value for the capability.
DefaultIndex	The item number, or index (zero-based) into ItemList[], of the "power-on" value for the capability.
ItemList[1]	The enumerated list: one value resides within each array element. Space for the list is not allocated inside this structure. The ItemList value is simply a placeholder for the start of the actual array, which must be allocated when the container is allocated. Remember to typecast the allocation to ItemType, as well as references to the elements of the array. Ex: Second element of ICAP_XFERMECH (assuming >= 2 NumItems) value = ((TW_UINT16*)twenum.ItemList)[1];

TW_EVENT

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

Used by

`DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT`

Description

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

Field Descriptions

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

TW_EXTIMAGEINFO

```
typedef struct {
     TW_UINT32       NumInfos;
     TW_INFO        Info[1];
} TW_EXTIMAGEINFO, FAR * pTW_EXTIMAGEINFO;
```

Used by

DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET

Description

This structure is used to pass extended image information from the Data Source to the Application at the end of State 7. The Application creates this structure at the end of State 7 when it receives XFERDONE. The Application fills NumInfos with the Number information it needs, and the array of extended information attributes in Info[ ]array. The Application then sends it down to the source using the above operation triplet. The Data Source then examines each Info, and fills the rest of data with information allocating memory when necessary.

The design of extended image information allows for two methods of passing multiple InfoID types. For instance, assume it is possible for a Source to generate more than one barcode off an image. An Application can request to acquire the data in one of two ways. The first way is as follows:

Applications asks for:

```
TW_EXTIMAGEINFO
  NumInfos = 4
  TW_INFO[0]
       InfoID = TWEI_BARCODECOUNT
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
         Item = 0
    TW_INFO[1]
       InfoID = TWEI_BARCODETYPE
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
         Item = 0
   TW_INFO[2]
       InfoID = TWEI_BARCODETEXTLENGTH
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
         Item = 0
   TW_INFO[3]
       InfoID = TWEI_BARCODETEXT
       ItemType = 0
       NumItems = 0
```

```
          ReturnCode = 0
            Item = 0
```

The Source returns…

```
TW_EXTIMAGEINFO
     NumInfos = 4
     TW_INFO[0]
          InfoID = TWEI_BARCODECOUNT
          ItemType = TW_UINT32
          NumItems = 1
          ReturnCode = TWCC_SUCCESS
          Item = 2
     TW_INFO[1]
          InfoID = TWEI_BARCODETYPE
          ItemType = TW_UINT32
          NumItems = 2
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-0
     TW_INFO[2]
          InfoID = TWEI_BARCODETEXTLENGTH
          ItemType = TW_UINT32
          NumItems = 2
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-1
     TW_INFO[3]
          InfoID = TWEI_BARCODETEXT
          ItemType = TW_HANDLE
          NumItems = 2
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-2
   ((TW_UINT32*)TW_HANDLE-0)[0]     TWBT_3OF9
((TW_UINT32*)TW_HANDLE-0)[1]     TWBT_2OF5INTERLEAVED
   ((TW_UINT32*)TW_HANDLE-1)[0]     16
((TW_UINT32*)TW_HANDLE-1)[1]     32
   ((TW_UINT8*)TW_HANDLE-2)[0]      Barcode Text 0
((TW_UINT8*)TW_HANDLE-2)[ ((TW_UINT32*)TW_HANDLE-1)[0]]
                             Barcode Text 1
```

Note that Item is a pointer to the first datum for this `TW_INFO`. The Item field must be a `TW_HANDLE` to the data if the value if the following is true:

```
     (SizeOfSpecifiedItem * NumItems) > sizeof(TW_HANDLE)
```

It is the responsibility of the Application to free both the `TW_EXTIMAGEINFO` structure and any Item values that are `TW_HANDLE`, based on this calculation.

The reason for this design is so that the Source and Application can easily index through the `TW_INFO` structures (ex: `TW_EXTIMAGEINFO->Item[0]`)

Note that the above structure could also be requested by the Application as follows:

```
TW_EXTIMAGEINFO
  NumInfos = 5
  TW_INFO[0]
       InfoID = TWEI_BARCODECOUNT
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
       Item = 0
  TW_INFO[1]
       InfoID = TWEI_BARCODETYPE
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
       Item = 0
 TW_INFO[2]
       InfoID = TWEI_BARCODETEXTLENGTH
       ItemType = TW_UNINT32
       NumItems = 0
       ReturnCode = 0
       Item = 0
 TW_INFO[3]
       InfoID = TWEI_BARCODETEXT
       ItemType = 0
       NumItems = 0
       ReturnCode = 0
       Item = 0
 TW_INFO[4]
       InfoID = TWEI_BARCODETEXT
       ItemType = 0
       NumItems = 0
       ReturnCode = 0
       Item = 0
```

If the Source detects multiple occurrences of a tag, then it must distribute the data as best it can across the applicable `TW_INFO` fields. `NumItems` must be equal to one, and if there are not enough `TW_INFOs` supplied for the specified `InfoID`, then any remaining data is discarded by the Source. In this instance the return structure is big enough, and would look like the following…

```
TW_EXTIMAGEINFO
  NumInfos = 5
  TW_INFO[0]
       InfoID = TWEI_BARCODECOUNT
       ItemType = TW_UINT32
       NumItems = 1
       ReturnCode = TWCC_SUCCESS
       Item = 2
  TW_INFO[1]
```

```
          InfoID = TWEI_BARCODETYPE
          ItemType = TW_UINT32
          NumItems = 2
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-0
     TW_INFO[2]
          InfoID = TWEI_BARCODETEXTLENGTH
          ItemType = TW_UINT32
          NumItems = 2
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-1
     TW_INFO[3]
          InfoID = TWEI_BARCODETEXT
          ItemType = TW_HANDLE
          NumItems = 1
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-2
     TW_INFO[4]
          InfoID = TWEI_BARCODETEXT
          ItemType = TW_HANDLE
          NumItems = 1
          ReturnCode = TWCC_SUCCESS
          Item = TW_HANDLE-3
   ((TW_UINT32*)TW_HANDLE-0)[0]     TWBT_3OF9
((TW_UINT32*)TW_HANDLE-0)[1]     TWBT_2OF5INTERLEAVED
   ((TW_UINT32*)TW_HANDLE-1)[0]     16
((TW_UINT32*)TW_HANDLE-1)[1]     32
   ((TW_UINT8*)TW_HANDLE-2)[0]      Barcode Text 0
   ((TW_UINT8*)TW_HANDLE-3)[ 0]     Barcode Text 1
```

Field Descriptions

`NumInfos`	The number of `INFO` structures must be greater than 0. The application should allocate memory and fill in the attribute tag for image information.
`Info[1]`	Array of information. See `TW_INFO` structure.

TW_FILESYSTEM

```
typedef struct {
  // DG_CONTROL / DAT_FILESYSTEM / MSG_xxxx fields…
  TW_STR255   InputName;
  TW_STR255   OutputName;
  TW_MEMREF   Context;
  // DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
  // DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE field…
  union {
        int         Recursive;
        TW_BOOL     Subdirectories;
  };
  // DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO fields…
  union {
        TW_INT32    FileType;
        TW_UINT32   FileSystemType;
  };
  TW_UINT32   Size;
  TW_STR32    CreateTimeDate;
  TW_STR32    ModifiedTimeDate;
  TW_UINT32   FreeSpace;
  TW_INT32    NewImageSize;
  TW_UINT32   NumberOfFiles;
  TW_UINT32   NumberOfSnippets;
  TW_UINT32   DeviceGroupMask;
  TW_INT8     Reserved[508];
} TW_FILESYSTEM, FAR * pTW_FILESYSTEM;
```

Used by

```
DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_COPY
DG_CONTROL / DAT_FILESYSTEM / MSG_CREATEDIRECTORY
DG_CONTROL / DAT_FILESYSTEM / MSG_DELETE
DG_CONTROL / DAT_FILESYSTEM / MSG_FORMATMEDIA
DG_CONTROL / DAT_FILESYSTEM / MSG_GETCLOSE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETFIRSTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_GETINFO
DG_CONTROL / DAT_FILESYSTEM / MSG_GETNEXTFILE
DG_CONTROL / DAT_FILESYSTEM / MSG_RENAME
```

Description

Provides information about the currently selected device.

Field Descriptions

`InputName`	The name of the input or source file.

`OutputName`	The result of an operation or the name of a destination file.
`Context`	A pointer to Source specific data used to remember state information, such as the current directory.
`MSG_GETINFO / MSG_GETFILEFIRST / MSG_DELETE`	
`Recursive`	When set to `TRUE` recursively apply the operation. (ex: deletes all subdirectories in the directory being deleted; or copies all sub-directories in the directory being copied.
`MSG_GETINFO / MSG_GETFILEFIRST / MSG_GETFILENEXT`	
`FileType`	One of the `TWFY_xxxx` values.
`Size`	`TWFY_DIRECTORY`- Total size of media in bytes. `TWFY_IMAGE`- Size of image in bytes. `TWFY_xxxx`- All other file types return a value of 0.
`CreateTimeDate`	The create date of the file, in the form "YYYY/MM/DD HH:mm:SS:sss" where YYYY is the year, MM is the numerical month, DD is the numerical day, HH is the hour, mm is the minute, SS is the second, and sss is the millisecond.
`ModifyTimeDate`	Last date the file was modified. Same format as *CreateTimeDate*.
`FreeSpace`	The bytes of free space left on the current device.
`NewImageSize`	An estimate of the amount of space a new image would take up, based on image layout, resolution and compression. Dividing this value into the *FreeSpace* will yield the approximate number of images that the Device has room for.
`NumberOfFiles`	`TWFY_IMAGE`- Return 0 `TWFY_xxxx`- Return number of `TWFY_IMAGE` files on the file system including those in all sub-directories.
`NumberOfSnippets`	The number of audio snippets associated with a file of type `TWFY_IMAGE`.
`DeviceGroupMask`	A set of bits, with each bit uniquely identifying a device of type `TWFY_CAMERA` and any associated `TWFY_CAMERATOP` and/or `TWFY_CAMERABOTTOM` devices. This field is intended to be used to group devices together. For example: • /Camera_ADF_Top – DeviceGroupMask = 1 • /Camera_ ADF_Bottom – DeviceGroupMask = 1 • /Camera_Flatbed_Top – DeviceGroupMask = 2 Possible masks values are bit fields – possible values are: • 1, 2, 4, 8, 16, 32, 64, 128 (0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80) See "File System" on page A-11. of this specification for more information.
`Reserved`	Space reserved for future expansion of this structure.

TW_FILTER

```
typedef struct {
  TW_UINT32 Size;
  TW_UINT32 DescriptorCount;
  TW_UINT32 MaxDescriptorCount;
  TW_UINT32 Condition;
  TW_HANDLE hDescriptors
} TW_FILTER, *pTW_FILTER
```

Used by

```
DG_IMAGE / DAT_FILTER / MSG_GET
DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT
DG_IMAGE / DAT_FILTER / MSG_RESET
DG_IMAGE / DAT_FILTER / MSG_SET
```

Description

Specifies the filter to be applied during image acquisition. More than one descriptor can be specified. All descriptors are applied with an OR statement. Filter will check if the current pixel color is inside (or outside) the area specified by the descriptors, and if it is, it will replace its color with value specified in `TW_FILTER_DESCRIPTOR/Replacement`. Note that the resulting image can be grayscale or bitonal, but not color.

Field Descriptions

`Size`	Size of this structure in bytes.
`Descriptors`	Number of descriptors in `hDescriptors` array.
`MaxDescriptorCount`	Maximum possible descriptors. Valid only for `MSG_GET` and `MSG_GETDEFAULT` operations.
`Condition`	If the value is 0 filter will check if current pixel color is inside the area specified by the descriptor. If the value is 1 it will check if it is outside of this area.
`hDescriptors`	Handle to array of `TW_FILTER_DESCRIPTOR`. See `TW_FILTER_DESCRIPTOR`.

TW_FILTER_DESCRIPTOR

```
typedef struct {
  TW_UINT32 Size;
  TW_UINT32 HueStart;
  TW_UINT32 HueEnd;
  TW_UINT32 SaturationStart;
  TW_UINT32 SaturationEnd;
  TW_UINT32 ValueStart;
  TW_UINT32 ValueEnd;
  TW_UINT32 Replacement;
} TW_FILTER_DESCRIPTOR, *pTW_FILTER_DESCRIPTOR;
```

Used by

```
DG_IMAGE / DAT_FILTER / MSG_GET
DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT
DG_IMAGE / DAT_FILTER / MSG_RESET
DG_IMAGE / DAT_FILTER / MSG_SET
```

Description

The range of colors specified by this structure is replaced with Replacement grayscale value in the binary image. The color is specified in HSV color space.

Field Descriptions

`Size`	Size of this structure in bytes.
`HueStart`	Hue starting number. Valid values 0 to 3600 (0° to 360°)
`HueEnd`	Hue ending number. Valid values 0 to 3600 (0° to 360°)
`SaturationStart`	Saturation starting number. Valid values 0 to 1000 (0% to 100%)
`SaturationEnd`	Saturation ending number. Valid values 0 to 1000 (0% to 100%)
`ValueStart`	Luminosity starting number. Valid values 0 to 1000 (0% to 100%)
`ValueEnd`	Luminosity ending number. Valid values 0 to 1000 (0% to 100%)
`Replacement`	Replacement grayscale value. Valid values 0 to $(2^{32})–1$ (Maximum value depends on grayscale bit depth)

TW_FIX32

```
typedef struct {
   TW_INT16      Whole;
   TW_UINT16     Frac;
} TW_FIX32, FAR * pTW_FIX32;
```

Used by

Embedded in the TW_CIECOLOR, TW_CIEPOINT, TW_DECODEFUNCTION, TW_FRAME, TW_IMAGEINFO, and TW_TRANSFORMSTAGE structures.

Used in TW_ARRAY, TW_ENUMERATION, TW_ONEVALUE, and TW_RANGE structures when ItemType is TWTY_FIX32.

Description

Stores a Fixed point number in two parts, a whole and a fractional part. The Whole part carries the sign for the number. The Fractional part is unsigned.

Field Descriptions

Whole	The Whole part of the floating point number. This number is signed.
Frac	The Fractional part of the floating point number. This number is unsigned.

The following functions convert TW_FIX32 to float and float to TW_FIX32:

```
   /***********************************************************
* FloatToFix32
* Convert a floating point value into a FIX32.
   ***********************************************************/
   TW_FIX32 FloatToFix32 (float floater)
   {
      TW_FIX32 Fix32_value;
      TW_INT32 value = (TW_INT32) (floater * 65536.0 + 0.5);
      Fix32_value.Whole = value >> 16;
      Fix32_value.Frac = value & 0x0000ffffL;
      return (Fix32_value);
   }
   /***********************************************************
   * Fix32ToFloat
   * Convert a FIX32 value into a floating point value.
   ***********************************************************/
   float FIX32ToFloat (TW_FIX32 fix32)
   {
      float     floater;
      floater = (float) fix32.Whole + (float) fix32.Frac / 65536.0;

      return floater;
   }
```

TW_FRAME

```
typedef struct {
   TW_FIX32      Left;
   TW_FIX32      Top;
   TW_FIX32      Right;
   TW_FIX32      Bottom;
} TW_FRAME, FAR * pTW_FRAME;
```

Used by

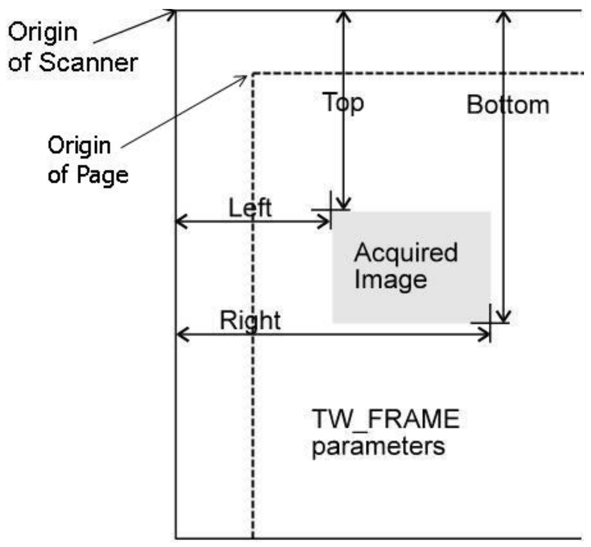
Embedded in the `TW_IMAGELAYOUT` structure

Description

Defines a frame rectangle in `ICAP_UNITS` coordinates.

Field Descriptions

`Left`	Value of the left-most edge of the rectangle (in `ICAP_UNITS`).
`Top`	Value of the top-most edge of the rectangle (in `ICAP_UNITS`).
`Right`	Value of the right-most edge of the rectangle (in `ICAP_UNITS`).
`Bottom`	Value of the bottom-most edge of the rectangle (in `ICAP_UNITS`).



Frame Parameters

TW_GRAYRESPONSE

```
typedef struct {
   TW_ELEMENT8      Response[1];
} TW_GRAYRESPONSE, FAR * pTW_GRAYRESPONSE;
```

Used by

```
DG_IMAGE / DAT_GRAYRESPONSE / MSG_RESET
DG_IMAGE / DAT_GRAYRESPONSE / MSG_SET
```

Description

This structure is used by the application to specify a set of mapping values to be applied to grayscale data. Use this structure for grayscale data whose bit depth is up to and including 8-bits. This structure can only be used if `TW_IMAGEINFO. PixelType` is `TWPT_GRAY`. The number of elements in the array is determined by `TW_IMAGEINFO.BitsPerPixel`—the number of elements is 2 raised to the power of `TW_IMAGEINFO.BitsPerPixel`.

This structure is primarily intended for use by applications that bypass the Source's built-in user interface.

Field Descriptions

`Response[1]`	Transfer curve descriptors. All three channels (`Channel1`, `Channel2` and `Channel3`) must contain the same value for every entry.

TW_HANDLE

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_IDENTITY

```
typedef struct {
        #if defined(__APPLE__)   /* cf: Mac version of TWAIN.h */
              TW_MEMREF   Id;
        #else
              TW_UINT32   Id;
        #endif
        TW_VERSION  Version;
        TW_UINT16   ProtocolMajor;
        TW_UINT16   ProtocolMinor;
        TW_UINT32   SupportedGroups;
        TW_STR32    Manufacturer;
        TW_STR32    ProductFamily;
        TW_STR32    ProductName;
} TW_IDENTITY, FAR * pTW_IDENTITY;
```

Used by

A large number of the operations because it identifies the application and the Source

Description

Provides identification information about a TWAIN entity. Used to maintain consistent communication between entities.

Field Descriptions

`Id`	A unique, internal identifier for the TWAIN entity. This field is only filled by the Source Manager. Neither an application nor a Source should fill this field. The Source uses the contents of this field to "identify" which application is invoking the operation sent to the Source.
`Version`	A `TW_VERSION` structure identifying the TWAIN entity.
`ProtocolMajor`	Major number of latest TWAIN version that this element supports (see `TWON_PROTOCOLMAJOR`).
`ProtocolMinor`	Minor number of latest TWAIN version that this element supports (see `TWON_PROTOCOLMINOR`).
`Manufacturer`	String identifying the manufacturer of the application or Source. e.g. "Aldus".
`ProductFamily`	Tells an application that performs device-specific operations which product family the Source supports. This is useful when a new Source has been released and the application doesn't know about the particular Source but still wants to perform Custom operations with it. e.g. "ScanMan".
`ProductName`	A string uniquely identifying the Source. This is the string that will be displayed to the user at Source select-time. This string must uniquely identify your Source for the user, and should identify the application unambiguously for Sources that care. e.g. "ScanJet IIc".

`SupportedGroups`	• The application will normally set this field to specify which Data Group(s) it wants the Source Manager to sort Sources by when presenting the Select Source dialog, or returning a list of available Sources. The application sets this prior to invoking a `MSG_USERSELECT` operation. • The application may also set this field to specify which Data Group(s) it wants the Source to be able to acquire and transfer. The application must do this prior to sending the Source its `MSG_ENABLEDS` operation. • The Source must set this field to specify which Data Group(s) it can acquire. It will do this in response to a `MSG_OPENDS`. • Beginning with TWAIN 2.0 the Source Manager reserves the most significant two byes in the SupportedGroups for the Data Flags (`0x0001000` to `0xFFFF0000`). `DF_DSM2` – identifies the Source Manager as TWAIN 2.0 compliant `DF_APP2` – is set by an Application that is TWAIN 2.0 compliant `DF_DS2` – is set by a Source that is TWAIN 2.0 compliant

TW_IMAGEINFO

```
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;
```

Used by

The `DG_IMAGE` / `DAT_IMAGEINFO` / `MSG_GET` operation

Description

Describes the "real" image data, that is, the complete image being transferred between the Source and application. The Source may transfer the data in a different format--the information may be transferred in "strips" or "tiles" in either compressed or uncompressed form. See the `TW_IMAGEMEMXFER` structure for more information.

The term "sample" is referred to a number of times in this structure. It holds the same meaning as in the TIFF specification. A sample is a contiguous body of image data that can be categorized by the channel or "ink color" it was captured to describe. In an R-G-B (Red-Green-Blue) image, such as on your TV or computer's CRT, each color channel is composed of a specific color. There are 3 samples in an R-G-B; Red, Green, and Blue. A C-Y-M-K image has 4 samples. A Grayscale or Black and White image has a single sample.

Note: The value -1 in `ImageWidth` and `ImageLength` are special cases. It is possible for a Source to not know either its `Width` or `Length`. Applications need to consider this when allocating memory or otherwise dealing with the size of the Image.

Field Descriptions

`XResolution`	The number of pixels per `ICAP_UNITS` in the horizontal direction. The current unit is assumed to be "inches" unless it has been otherwise negotiated between the application and Source.
`YResolution`	The number of pixels per `ICAP_UNITS` in the vertical direction.
`ImageWidth`	How wide, in pixels, the entire image to be transferred is. If the Source doesn't know, set this field to -1 (hand scanners may do this). --1 can only be used if the application has set `ICAP_UNDEFINEDIMAGESIZE` to `TRUE`.

`ImageLength`	How tall/long, *in pixels*, the image to be transferred is. If the Source doesn't know, set this field to -1 (hand scanners may do this). -1 can only be used if the application has set ICAP_UNDEFINEDIMAGESIZE to `TRUE`.
`SamplesPerPixel`	The number of samples being returned. For R-G-B, this field would be set to 3. For C-M-Y-K, 4. For Grayscale or Black and White, 1.
`BitsPerSample[8]`	For each sample, the number of bits of information. 24-bit R-G-B will typically have 8 bits of information in each sample (8+8+8). Some 8-bit color is sampled at 3 bits Red, 3 bits Green, and 2 bits Blue. Such a scheme would put 3, 3, and 2 into the first 3 elements of this array. The supplied array allows up to 8 samples. Samples are not limited to 8 bits. However, both the application and Source must simultaneously support sample sizes greater than 8 bits per color.
`BitsPerPixel`	The number of bits in each image pixel (or bit depth). This value is invariant across the image. 24-bit R-G-B has BitsPerPixel = 24. 40-bit C-M-Y-K has BitsPerPixel=40. 8-bit Grayscale has BitsPerPixel = 8. Black and White has BitsPerPixel = 1.
`Planar`	If SamplesPerPixel > 1, indicates whether the samples follow one another on a pixel-by-pixel basis (R-G-B-R-G-B-R-G-B...) as is common with a one-pass scanner or all the pixels for each sample are grouped together (complete group of R, complete group of G, complete group of B) as is common with a three-pass scanner. If the pixel-by-pixel method (also known as "chunky") is used, the Source should set Planar = `FALSE`. If the grouped method (also called "planar") is used, the Source should set Planar = `TRUE`.
`PixelType`	This is the highest categorization for how the data being transferred should be interpreted by the application. This is how the application can tell if the data is Black and White, Grayscale, or Color. Currently, the only color type defined is "tri-stimulus", or color described by three characteristics. Most popular color description methods use tri-stimulus descriptors. For simplicity, the constant used to identify tri-stimulus color is called TWPT_RBG, for R-G-B color. There is no default for this value. Fill this field with the appropriate `TWPT_xxxx` constant.
`Compression`	The compression method used to process the data being transferred. Default is no compression. Fill this field with the appropriate `TWCP_xxxx` constant.

TW_IMAGELAYOUT

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

Used by

```
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT
DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET
```

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.

Another attribute of this structure is the included frame, page, and document indexing information. Most Sources and applications, at least at first, will likely set all these fields to one. For Sources that can acquire more than one frame from a page in a single acquisition, the FrameNumber field will be handy. Sources that can acquire more than one page from a document feeder will use `PageNumber` and `DocumentNumber`. These fields will be especially useful for forms-processing applications and other applications with similar document tracking requirements.

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`.
`DocumentNumber`	The document number, assigned by the Source, that the acquired data originated on. Useful for grouping pages together. Usually a physical representation, this could just as well be a logical construct. Initial value is 1. Increment when a new document is placed into the document feeder (usually tell this has happened when the feeder empties). Reset when no longer acquiring from the feeder.
`PageNumber`	The page which the acquired data was captured from. Useful for grouping Frames together that are in some way related, usually Source. Usually a physical representation, this could just as well be a logical construct. Initial value is 1. Increment for each page fed from a page feeder. Reset when a new document is placed into the feeder.
`FrameNumber`	Usually a chronological index of the acquired frame. These frames are related to one another in some way; usually they were acquired from the same page. The Source assigns these values. Initial value is 1. Reset when a new page is acquired.

TW_IMAGEMEMXFER

```
typedef struct {
     TW_UINT16      Compression;
     TW_UINT32      BytesPerRow;
     TW_UINT32      Columns;
     TW_UINT32      Rows;
     TW_UINT32      XOffset;
     TW_UINT32      YOffset;
     TW_UINT32      BytesWritten;
     TW_MEMORY      Memory;
} TW_IMAGEMEMXFER, FAR * pTW_IMAGEMEMXFER;
```

Used by

DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET

Description

Describes the form of the acquired data being passed from the Source to the application. When used in combination with a TW_IMAGEINFO structure, the application can correctly interpret the image.

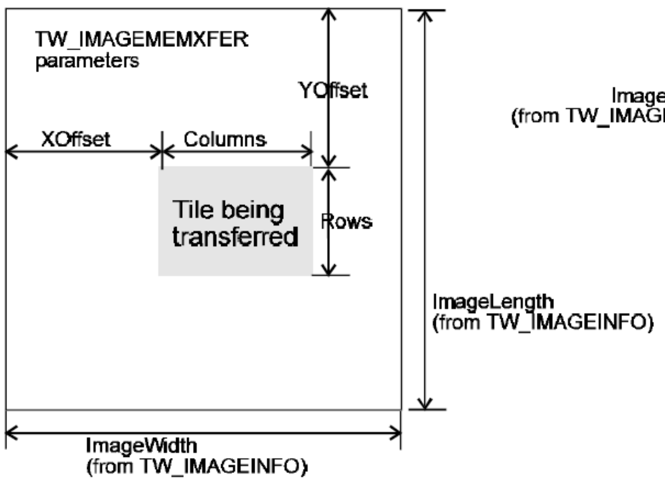
This structure allows transfer of "chunks" from the acquired data. These portions may be either "strips" or "tiles." Strips are tiles whose width matches that of the full image. Strips are always passed sequentially, from "top" to "bottom". A tile's position does not necessarily follow that of the previously passed tile. Most Sources will transfer strips.

Note: When transferring tiles, the application should remember what corner was contained in the first tile of a plane. When the opposite corner is delivered, the plane is complete. The dimensions of the memory transfers may vary.

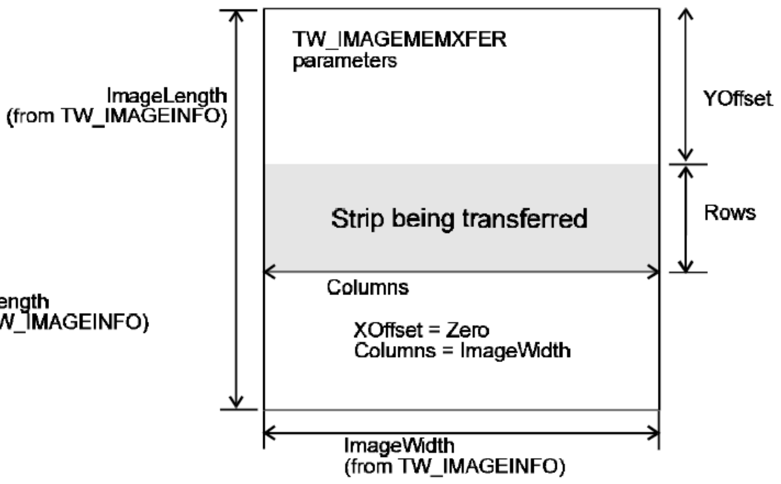
Data may be passed either compressed or uncompressed. All Sources must support uncompressed Data. Sources are not required to support compressed data transfers. Compressed data transfers, and how the values are entered into the fields of this structure, are described in Chapter 4, "Advanced Application Implementation".

Following is a picture of some of the fields from a TW_IMAGEMEMXFER structure. **The large outline shows the entire image which was selected to be transferred.** The smaller rectangle shows the particular portion being described by this TW_IMAGEMEMXFER structure.

Note: Remember that for a "strip" transfer `XOffset = 0`, and `Columns = TW_IMAGEINFO.ImageWidth`.


TW_IMAGEMEMXFER parameters
YOffset
XOffset
Columns
Tile being transferred
Rows
ImageLength (from TW_IMAGEINFO)
ImageWidth (from TW_IMAGEINFO)

Tile Positioning


TW_IMAGEMEMXFER parameters
ImageLength (from TW_IMAGEINFO)
YOffset
Rows
Strip being transferred
Columns
XOffset = Zero
Columns = ImageWidth
ImageWidth (from TW_IMAGEINFO)

Strip Positioning

Field Descriptions

`Compression`	The compression method used to process the data being transferred. Write the constant (`TWCP_xxxx`) that precisely describes the type of compression used for the buffer. This may be different from the method reported in the `TW_IMAGEINFO` structure (if the user selected a different method before the actual transfer began, for instance). This is unlikely, but possible. The application can optionally abort the acquisition if the value in this field differs from the `TW_IMAGEINFO` value. Default is no compression (`TWCP_NONE`) and most transfers will probably be uncompressed. See the list of constants in the TWAIN.H file.
`BytesPerRow`	The number of uncompressed bytes in each row of the piece of the image being described in this buffer.
`Columns`	The number of uncompressed columns (in pixels) in this buffer.
`Rows`	The number or uncompressed rows (in pixels) in this buffer.
`XOffset`	How far, in pixels, the left edge of the piece of the image being described by this structure is inset from the “left” side of the original image. If the Source is transferring in “strips”, this value will equal zero. If the Source is transferring in “tiles”, this value will often be non-zero.
`YOffset`	Same idea as `XOffset`, but the measure is in pixels from the “top” of the original image to the upper edge of this piece.
`BytesWritten`	The number of bytes written into the transfer buffer. This field must always be filled in correctly, whether compressed or uncompressed data is being transferred.
`Memory`	A structure of type `TW_MEMORY` describing who must dispose of the buffer, the actual size of the buffer, in bytes, and where the buffer is located in memory.

TW_INFO

```
typedef struct {
     TW_UINT16       InfoID;
     TW_UINT16       ItemType;
     TW_UINT16       NumItems;
     TW_UINT16       ReturnCode;
     TW_UINTPTR      Item;
  } TW_INFO, FAR * pTW_INFO;
```

Used by

Within TW_EXTIMAGEINFO structure.

Description

This structure is used to pass specific information between the data source and the application.

Field Descriptions

InfoID	Tag identifying an information. For TW_EXTIMAGEINFO, the information ID is defined as IACAP_xxxx. (Please refer to Extended Image capabilities).
ItemType	Item data type. It is one of TWTY_xxxx value as listed in the TW_INFO.ITEMTYPE below.
NumItems	Number of items for this field.
ReturnCode	This is the return code of availability of data for extended image attribute requested. Following is the list of possible condition codes: TWRC_INFONOTSUPPORTED TWRC_DATANOTAVAILABLE TWRC_SUCCESS
Item	The TW_INFO.Item field contains either data or a handle to data. The field contains data if the total amount of data is less than or equal to four bytes. The field contains a handle if the total amount of data is more than four bytes. The amount of data is determined by multiplying TW_INFO.NumItems times the byte size of the data type specified by TW_INFO.ItemType. If the TW_INFO.Item field contains a handle to data, then the Application is responsible for freeing that memory.

TW_INFO.ITEMTYPE

ItemType	NumItems	Data/Pointer	Reason
TW_STR32	1	handle to data	(sizeof (TW_STR32) * 1) > 4
TW_INT32	1	data	(sizeof (TW_INT32) * 1) == 4
TW_INT8	3	data	(sizeof (TW_INT8) * 3) < 4
TW_INT8	5	handle to data	(sizeof (TW_INT8) * 5) > 4

TW_INFO Return Codes

Following is the list of added return codes.

`TWRC_INFONOTSUPPORTED`	Requested information is not supported.
`TWRC_DATANOTAVAILABLE`	Requested information is supported, but some unknown reason, information is not available.

TW_JPEGCOMPRESSION

```
typedef struct {
  TW_UINT16      ColorSpace;
  TW_UINT32      SubSampling;
  TW_UINT16      NumComponents;
  TW_UINT16      RestartFrequency;
  TW_UINT16      QuantMap[4];
  TW_MEMORY      QuantTable[4];
  TW_UINT16      HuffmanMap[4];
  TW_MEMORY      HuffmanDC[2];
  TW_MEMORY      HuffmanAC[2];
} TW_JPEGCOMPRESSION, FAR * pTW_JPEGCOMPRESSION;
```

Used by

```
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET
```

Description

Describes the information necessary to transfer a JPEG-compressed image during a buffered transfer. Images compressed in this fashion will be compatible with the JPEG File Interchange Format, version 1.1. For more information on JPEG and TWAIN, see Chapter 4, "Advanced Application Implementation". The TWAIN JPEG implementation is based on the JPEG Draft International Standard, version 10918-1. The sample tables found in Section K of the JPEG Draft International Standard, version 10918-1 are used as the default tables for QuantTable, HuffmanDC, and HuffmanAC.

Field Descriptions

`ColorSpace`	One of the `TWPT_xxxx` values. Defines the color space in which the compressed components are stored. Only spaces supported by the Source for `ICAP_JPEGPIXELTYPE` are valid.
`SubSampling`	Encodes the horizontal and vertical subsampling in the form ABCDEFGH, where ABCD are the high-order four nibbles which represent the horizontal subsampling and EFGH are the low-order four nibbles which represent the vertical subsampling. Each nibble may have a value of 0, 1, 2, 3, or 4. However, max(A,B,C,D) * max(E,F,G,H) must be less than or equal to 10. Subsampling is irrelevant for single component images. Therefore, the corresponding nibbles should be set to 1. e.g. To indicate subsampling two Y for each U and V in a YUV space image, where the same subsampling occurs in both horizontal and vertical axes, this field would hold 0x21102110. For a grayscale image, this field would hold 0x10001000. A CMYK image could hold 0x11111111.
`NumComponents`	Number of color components in the image to be compressed.
`RestartFrequency`	Number of MDUs (Minimum Data Units) between restart markers. Default is 0, indicating that no restart markers are used. An MDU is defined for interleaved data (i.e. R-G-B, Y-U-V, etc.) as a minimum complete set of 8x8 component blocks.
`QuantMap[4]`	Mapping of components to Quantization tables.

`QuantTable[4]`	Quantization tables.
`HuffmanMap[4]`	Mapping of components to Huffman tables. Null entries signify selection of the default tables.
`HuffmanDC[2]`	DC Huffman tables. Null entries signify selection of the default tables.
`HuffmanAC[2]`	AC Huffman tables. Null entries signify selection of the default tables.

TW_MEMORY

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

Used by

Embedded in the `TW_IMAGEMEMXFER` and `TW_JPEGCOMPRESSION` structures

Description

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

Field Descriptions

`Flags`	Encodes which entity releases the buffer and how the buffer is referenced. The ownership flags must be used: • when transferring Buffered Memory data as tiles • when transferring Buffered Memory that is compressed • in the `TW_JPEGCOMPRESSION` structure When transferring Buffered Memory data as uncompressed strips, the application allocates the buffers and is responsible for setting the ownership flags. This field is used to identify how the memory is to be referenced. The memory is always referenced by a Handle on the Macintosh and a Pointer under UNIX. It is referenced by a Handle or a pointer under Microsoft Windows. Use `TWMF_xxxx` constants, bit-wise OR'd together to fill this field. Flag Constants: `TWMF_APPOWNS` 0x1 `TWMF_DSMOWNS` 0x2 `TWMF_DSOWNS` 0x4 `TWMF_POINTER` 0x8 `TWMF_HANDLE` 0x10
`Length`	The size of the buffer in bytes. Should always be an even number and word-aligned.
`TheMem`	Reference to the buffer. May be a Pointer or a Handle (see Flags field to make this determination). You must typecast this field before referencing it in your code.

TW_MEMREF

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

Used by

Embedded in the `TW_EVENT` and `TW_MEMORY` structures

Description

Memory references are specific to each operating system. TWAIN defines `TW_MEMREF` to be the memory reference type supported by the operating system.

Field Descriptions

See definitions above.

TW_METRICS

```
typedef struct {
   TW_UINT32      SizeOf;
   TW_UINT32      ImageCount;
   TW_UINT32      SheetCount;
} TW_METRICS, FAR * pTW_METRICS;
```

Used by

The `DG_CONTROL / DAT_METRICS / MSG_*` calls

Description

Provides metrics since the last call to `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

Field Descriptions

`SizeOf`	Set by the application. Specifies the number of bytes in the structure (i.e., `SizeOf(TW_METRICS)`)
`ImageCount`	The number of images made available for transfer by the driver. This is not necessarily the same as the number of images actually transferred, since the application may opt to skip transfers or to end without transferring all images. This value starts as 0 when the driver is opened with `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`, and resets to zero any time `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` is received.
`SheetCount`	The number of sheets of paper processed by the scanner. This value starts as 0 when the driver is opened with `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`, and resets to zero any time `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` is received.

TW_UINTPTR

On Windows:

```
typedef UINT_PTR        TW_UINTPTR;
```

On Macintosh and Unix:

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

Used by

Embedded in the `TW_INFO` structure.

Description

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

Field Descriptions

See definitions above

TW_ONEVALUE

```
typedef struct {
   TW_UINT16     ItemType;
   TW_UINT32     Item;
} TW_ONEVALUE, FAR * pTW_ONEVALUE;
```

Used by

`TW_CAPABILITY` structure (when `ConType` field specifies `TWON_ONEVALUE`)

Description

Stores a single value (item) which describes a capability. This structure is currently used only in a `TW_CAPABILITY` structure. Such a value would be useful to describe the current value of the device's contrast, or to set a specific contrast value. This structure is related in function and purpose to `TW_ARRAY`, `TW_ENUMERATION`, and `TW_RANGE`.

Note that in cases where the data type is `TW_UINT16`, the data should reside in the lower word.

Field Descriptions

`ItemType`	The type of the item. The type is indicated by the constant held in this field. The constant is of the kind `TWTY_xxxx`.
`Item`	The value.

TW_PALETTE8

```
typedef struct {
  TW_UINT16     NumColors;
  TW_UINT16     PaletteType;
  TW_ELEMENT8    Colors[256];
} TW_PALETTE8, FAR * pTW_PALETTE8;
```

Used by

```
DG_IMAGE / DAT_PALETTE8 / MSG_GET
DG_IMAGE / DAT_PALETTE8 / MSG_GETDEFAULT
DG_IMAGE / DAT_PALETTE8 / MSG_RESET
DG_IMAGE / DAT_PALETTE8 / MSG_SET
```

Description

This structure holds the color palette information for buffered memory transfers of type `ICAP_PIXELTYPE` = `TWPT_PALETTE`.

Field Descriptions

`NumColors`	Number of colors in the color table; maximum index into the color table should be one less than this (since color table indexes are zero-based).
`PaletteType`	`TWPA_xxxx` constant specifying the type of palette.
`Colors[256]`	Array of palette values.

TW_PASSTHRU

```
typedef struct {
   TW_MEMREF     pCommand;
   TW_UINT32     CommandBytes;
   TW_INT32      Direction;
   TW_MEMREF     pData;
   TW_UINT32     DataBytes;
   TW_UINT32     DataBytesXfered;
} TW_PASSTHRU, FAR * pTW_PASSTHRU;
```

Used by

`DG_CONTROL / DAT_PASSTHRU / MSG_PASSTHRU`

Description

Used to bypass the TWAIN protocol when communicating with a device. All memory must be allocated and freed by the Application. Use of this feature is limited to Source writers who require a standard entry point for specialized Applications, such as diagnostics.

Field Descriptions

`pCommand`	Pointer to Command buffer.
`CommandBytes`	Number of bytes in Command buffer.
`Direction`	One of the `TWDR_xxxx` values. Defines the direction of data flow.
`pData`	Pointer to Data buffer.
`DataBytes`	Number of bytes in Data buffer.
`DataBytesXfered`	Number of bytes successfully transferred.

TW_PENDINGXFERS

```
typedef struct {
   TW_UINT16 Count;
   union {
      TW_UINT32 EOJ;
      TW_UINT32 Reserved;
      #if defined(__APPLE__) /* cf: Mac version of TWAIN.h */
         union {
                  TW_UINT32 EOJ;
                  TW_UINT32 Reserved;
                  } TW_JOBCONTROL;
      #endif
   };
} TW_PENDINGXFERS, FAR *pTW_PENDINGXFERS;
```

Used by

DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET

Description

This structure tells the application how many more complete transfers the Source currently has available. The application should MSG_GET this structure at the conclusion of a transfer to confirm the Source's current state. If the Source has more transfers pending it will remain in State 6 awaiting initiation of the next transfer by the application.

If it has no more image transfers pending, it will place zero into the Count and will have automatically transitioned to State 5 (audio transfers will remain in State 6, even when the Count goes to zero).

If the Source knows there are more transfers pending but is unsure of the actual number, it should place -1 into Count (for example, with document feeders or continuous video sources). Otherwise, the Source should place the actual number of pending transfers into Count.

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.

TW_RANGE

```
typedef struct {
   TW_UINT16     ItemType;
   TW_UINT32     MinValue;
   TW_UINT32     MaxValue;
   TW_UINT32     StepSize;
   TW_UINT32     DefaultValue;
   TW_UINT32     CurrentValue;
} TW_RANGE, FAR * pTW_RANGE;
```

Used by

`TW_CAPABILITY` structure (when `ConType` field specifies `TWON_RANGE`)

Description

Stores a range of individual values describing a capability. The values are uniformly distributed between a minimum and a maximum value. The step size between each value is constant. Such a value is useful when describing such capabilities as the resolutions of a device which supports discreet, uniform steps between each value, such as 50 through 300 dots per inch in steps of 2 dots per inch (50, 52, 54, ..., 296, 298, 300). This structure is related in function and purpose to `TW_ARRAY`, `TW_ENUMERATION`, and `TW_ONEVALUE`.

Field Descriptions

`ItemType`	The type of items in the list. The type is indicated by the constant held in this field. The constant is of the kind `TWTY_xxxx`. All items in the list have the same size/type.
`MinValue`	The least positive/most negative value of the range.
`MaxValue`	The most positive/least negative value of the range.
`StepSize`	The delta between two adjacent values of the range. e.g. Item2 - Item1 = StepSize;
`DefaultValue`	The device's "power-on" value for the capability. If the application is performing a `MSG_SET` operation and isn't sure what the default value is, set this field to `TWON_DONTCARE32`.
`CurrentValue`	The value to which the device (or its user interface) is currently set to for the capability.

TW_RGBRESPONSE

```
typedef struct {
     ELEMENT8     Response[1];
   } TW_RGBRESPONSE, FAR * pTW_RGBRESPONSE;
```

Used by

`DG_IMAGE` / `DAT_RGBRESPONSE` / `MSG_RESET`
`DG_IMAGE` / `DAT_RGBRESPONSE` / `MSG_SET`

Description

This structure is used by the application to specify a set of mapping values to be applied to RGB color data. Use this structure for RGB data whose bit depth is up to, and including, 8-bits. The number of elements in the array is determined by `TW_IMAGEINFO`.BitsPerPixel—the number of elements is 2 raised to the power of `TW_IMAGEINFO`.BitsPerPixel.

This structure is primarily intended for use by applications that bypass the Source's built-in user interface.

Field Descriptions

`Response[1]`	Transfer curve descriptors. To minimize color shift problems, writing the same values into each channel is desirable.

TW_SETUPFILEXFER

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

Used by

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

Description

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

Field Descriptions

FileName	A complete file specifier to the target file. On Windows, be sure to include the complete pathname.
Format	The format of the file the Source is to fill. Fill with the correct constant—as negotiated with the Source—of type TWFF_xxxx.
VRefNum	The volume reference number for the file. This applies to Macintosh only. On Windows and Linux, fill the field with TWON_DONTCARE16.

TW_SETUPMEMXFER

```
typedef struct {
   TW_UINT32     MinBufSize;
   TW_UINT32     MaxBufSize;
   TW_UINT32     Preferred;
} TW_SETUPMEMXFER, FAR * pTW_SETUPMEMXFER;
```

Used by

DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET

Description

Provides the application information about the Source's requirements and preferences regarding allocation of transfer buffer(s). The best applications will allocate buffers of the Preferred size. An application should never allocate a buffer smaller than MinBufSize. Some Sources may not be able to fill a buffer larger than MaxBufSize so a larger allocation is a waste of RAM (digital cameras or frame grabbers fit this category).

Sources should fill out all three fields as accurately as possible. If a Source can fill an indeterminately large buffer (hand scanners might do this), put a -1 in `MaxBufSize`.

Field Descriptions

`MinBufSize`	The size of the smallest transfer buffer, in bytes, that a Source can be successful with. This will typically be the number of bytes in an uncompressed row in the block to be transferred. An application should never allocate a buffer smaller than this.
`MaxBufSize`	The size of the largest transfer buffer, in bytes, that a Source can fill. If a Source can fill an arbitrarily large buffer, it might set this field to negative 1 to indicate this (a hand-held scanner might do this, depending on how long its cord is). Other Sources, such as frame grabbers, cannot fill a buffer larger than a certain size. Allocation of a transfer buffer larger than this value is wasteful.
`Preferred`	The size of the optimum transfer buffer, in bytes. A smart application will allocate transfer buffers of this size, if possible. Buffers of this size will optimize the Source's performance. Sources should be careful to put reasonable values in this field. Buffers that are 10's of kbytes will be easier for applications to allocate than buffers that are 100's or 1000's of kbytes.

TW_STATUS

```
typedef struct {
   TW_UINT16      ConditionCode;    // output
   union {
      TW_UINT16   Data;             // output (TWAIN 2.1 and newer)
      TW_UINT16   Reserved;         // output (TWAIN 2.0 and older)
   };
} TW_STATUS, FAR * pTW_STATUS;
```

Used by

`DG_CONTROL / DAT_STATUS / MSG_GET`
`DG_CONTROL / DAT_STATUSUTF8 / MSG_GET` (as part of `TW_STATUSUTF8`)

Description

Describes the status of a source.

Field Descriptions

`ConditionCode`	`TWCC_xxxx` Condition Code describing the status.
`Data`	Valid for TWAIN 2.1 and later. This field contains additional scanner-specific data. If there is no data, then this value must be zero.
`Reserved`	Only option for TWAIN 2.0 and earlier. If used it must be zero.

TW_STATUSUTF8

```
typedef struct {
    TW_STATUS   Status;// input
    TW_UINT32   Size;// output
    TW_HANDLE   UTF8string;// output
} TW_STATUSUTF8, FAR * pTW_STATUSUTF8;
```

Used by

`DG_CONTROL / DAT_STATUSUTF8 / MSG_GET`

Description

Translates the contents of `Status` into a localized `UTF8string`, with the total number of bytes in the string.

Field Descriptions

`Status`	`TW_STATUS` data received from a previous call to `DG_CONTROL / DAT_STATUS / MSG_GET`.
`Size`	Total number of bytes in the `UTF8string`, plus the terminating NULL byte. This is not the same as the total number of characters in the string.
`UTF8string`	`TW_HANDLE` to a UTF-8 encoded localized string (based on `TW_IDENTITY`.Language or `CAP_LANGUAGE`). The `Source` allocates it, the `Application` frees it.

TW_TRANSFORMSTAGE

```
typedef struct {
   TW_DECODEFUNCTION      Decode[3];
   TW_FIX32              Mix[3][3];
} TW_TRANSFORMSTAGE, FAR * pTW_TRANSFORMSTAGE;
```

Used by

Embedded in the `TW_CIECOLOR` structure

Description

Specifies the parametrics used for either the ABC or LMN transform stages.

Go to http://www.cie.co.at/ for more information about CIE XYZ Color Space.

Field Descriptions

`Decode[3]`	Channel-specific transform parameters.
`Mix[3][3]`	3x3 matrix that specifies how channels are mixed in

TW_TWAINDIRECT

```
typedef struct {
        TW_UINT32     SizeOf;
        TW_UINT16     CommunicationManager;
        TW_HANDLE     Send;
        TW_UINT32     SendSize;
        TW_HANDLE     Receive;
        TW_UINT32     ReceiveSize;
} TW_TWAINDIRECT, FAR * pTW_TWAINDIRECT;
```

Used by

The DG_CONTROL / DAT_TWAINDIRECT / MSG_* calls

Description

Provides identification information about a TWAIN entity. Used to maintain consistent communication between entities.

Field Descriptions

Size of	Set by the application. Specifies the number of bytes in the structure (i.e., sizeof(TW_TWAINDIRECT)).
Communication Manager	The interpretation of the Send data may be influenced by the communication manager that is helping the application connect to the scanner. For instance, a task action to "scan" may need to be ignored under some circumstances. Use TWCOMMGR_* values.
Send	A handle to data to be sent from the application to the driver. The application owns this handle. If there is no data, this field is set to NULL.
SendSize	The number of bytes in the Send buffer. If there is no data this field is set to 0.
Receive	A handle to data sent from the driver to the application. The driver creates this handle, the application must free it. If there is no data this field is set to NULL.
ReceiveSize	The number of bytes in the Receive buffer, set by the driver. If there is no data this field is set to 0.

TW_USERINTERFACE

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

Used by

`DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`
`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

Description

This structure is used to handle the user interface coordination between an application and a Source.

Field Descriptions

`ShowUI`	Set to `TRUE` by the application if the Source should activate its built-in user interface. Otherwise, set to `FALSE`. Note that not all sources support ShowUI = `FALSE`. See the description of `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` for more information.
`ModalUI`	If ShowUI is `TRUE`, then an application setting this to `TRUE` requests the Source to run Modal (no user access to the application's windows while the Source is running).
`hParent`	Microsoft Windows only: Application's window handle. The Source designates the hWnd as its parent when creating the Source dialog. **NOTE:** Window handle allows Source's user interface to be a proper child of the parent application.

TW_VERSION

```
typedef struct {
  TW_UINT16     MajorNum;
  TW_UINT16     MinorNum;
  TW_UINT16     Language;
  TW_UINT16     Country;
  TW_STR32      Info;
} TW_VERSION, FAR * pTW_VERSION;
```

Used by

This is embedded in the `TW_IDENTITY` data structure

Description

A general way to describe the version of software that is running.

Field Descriptions

`MajorNum`	This refers to your application or Source's major revision number. e.g. The "2" in "version 2.1".
`MinorNum`	The incremental revision number of your application or Source. e.g. The "1" in "version 2.1".
`Language`	The primary language for your Source or application. e.g. TWLG_GER.
`Country`	The primary country where your Source or application is intended to be distributed. e.g. Germany.
`Info`	General information string - fill in as needed. e.g. "1.0b3 Beta release".

Data Argument Types that Don't Have Associated TW_Structures

Most of the `DAT_xxxx` components of the TWAIN operation triplets have a corresponding data structure whose name begins with TW_ and then uses the same suffix as the DAT_name. However, the following do not use that pattern.

DAT_IMAGEFILEXFER

Acts on `NULL` data.

DAT_IMAGENATIVEXFER

Uses a `TW_HANDLE` variable.

On Windows: A handle variable to a DIB (Device Independent Bitmap) located in memory.

On Macintosh: A handle to a TIFF image located in memory if both application and data source are version 2.4 or later. A handle to a QuickDraw picture located in memory if either the application or the data source is TWAIN 2.3 and earlier.

On Linux: A handle to a TIFF. It is a TIFF image located in memory.

DAT_NULL

Used by the Source to signal the need for an event to announce `MSG_XFERREADY`, `MSG_CLOSEDSOK`, `MSG_CLOSEDSREQ`, or `MSG_DEVICEEVENT`.

Mac OS X using TWAIN DSM 1.9 will use `MSG_INVOKECALLBACK`.

DAT_PARENT

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

On Windows: They act on a variable of type `TW_HANDLE`. Prior to the operation, the application must write, a window handle to the application's window that acts as the "parent" for the Source's user interface. This must be done whether or not the Source's user interface will be used. The Source Manager uses this window handle to signal the application when data is ready for transfer (`MSG_XFERREADY`) or the Source needs to be closed (`MSG_CLOSEDSREQ`).

On Macintosh: These act on `NULL` data.

On Linux: These act on `NULL` data.

DAT_XFERGROUP

Used by the `DG_CONTROL` / `DAT_XFERGROUP` / `MSG_GET` operation. The data acted on by this operation is a variable of type `TW_UINT32`. (The same as a `DG_xxxx` designator.) The value of this variable is indeterminate prior to the operation. Following the operation, a single bit is set indicating the Data Group of the transfer.

Constants

Generic Constants

Constants

Version	Constant ID	Value
1.0	TWON_PROTOCOLMAJOR	2
1.0	TWON_PROTOCOLMINOR	1
1.0	TWON_ARRAY	3
1.0	TWON_ENUMERATION	4
1.0	TWON_ONEVALUE	5
1.0	TWON_RANGE	6
1.0	TWON_ICONID	962
1.0	TWON_DSMID	461
1.0	TWON_DSMCODEID	63
1.0	TWON_DONTCARE8	0xff
1.0	TWON_DONTCARE16	0xffff
1.0	TWON_DONTCARE32	0xffffffff

Flags used in TW_MEMORY

Version	Constant ID	Value
1.0	TWMF_APPOWNS	0x0001
1.0	TWMF_DSMOWNS	0x0002
1.0	TWMF_DSOWNS	0x0004
1.0	TWMF_POINTER	0x0008
1.0	TWMF_HANDLE	0x0010

Palette types for TW_PALETTE8

Version	Constant ID	Value
1.0	TWPA_RGB	0
1.0	TWPA_GRAY	1
1.0	TWPA_CMY	2

Events for TW_DEVICEEVENT

See “CAP_DEVICEEVENT” on page 8-81.

File Types for TW_FILESYSTEM

Version	Constant ID	Value
1.0	TWFY_CAMERA	0
1.0	TWFY_CAMERATOP	1
1.0	TWFY_CAMERABOTTOM	2
1.0	TWFY_CAMERAPREVIEW	3
1.0	TWFY_DOMAIN	4
1.0	TWFY_HOST	5
1.0	TWFY_DIRECTORY	6
1.0	TWFY_IMAGE	7
1.0	TWFY_UNKNOWN	8

Query Support Bits

Note: These are bits in a mask.

Version	Constant ID	Value
1.6	TWQC_GET	0x0001
1.6	TWQC_SET	0x0002
1.6	TWQC_GETDEFAULT	0x0004
1.6	TWQC_GETCURRENT	0x0008
1.6	TWQC_RESET	0x0010
2.2	TWQC_SETCONSTRAINT	0x0020
2.2	TWQC_GETHELP	0x0100
	TWQC_GETLABEL	0x0200
	TWQC_GETLABELENUM	0x0400

ConType for Capability Container structures

Version	Constant ID	Value
1.0	TWON_ARRAY	3
1.0	TWON_ENUMERATION	4
1.0	TWON_ONEVALUE	5
1.0	TWON_RANGE	6

ItemType for Capability Container structures

Version	Constant ID	Value
1.0	TWTY_INT8	0x0000
1.0	TWTY_INT16	0x0001
1.0	TWTY_INT32	0x0002
1.0	TWTY_UINT8	0x0003
1.0	TWTY_UINT16	0x0004
1.0	TWTY_UINT32	0x0005
1.0	TWTY_BOOL	0x0006

1.0	TWTY_FIX32	0x0007
1.0	TWTY_FRAME	0x0008
1.0	TWTY_STR32	0x0009
1.0	TWTY_STR64	0x000A
1.0	TWTY_STR128	0x000B
1.0	TWTY_STR255	0x000C
1.0	TWTY_HANDLE	0x000F // Item is a TW_HANDLE

Direction for TW_PASSTHRU

Version	Constant ID	Value
1.8	TWDR_GET	1
1.8	TWDR_SET	2

Patch Codes for TW_PENDINGXFERS

Version	Constant ID	Value
1.0	TWEJ_NONE	0x0000
1.0	TWEJ_MIDSEPERATOR	0x0001
1.0	TWEJ_PATCH1	0x0002
1.0	TWEJ_PATCH2	0x0003
1.0	TWEJ_PATCH3	0x0004
1.0	TWEJ_PATCH4	0x0005
1.0	TWEJ_PATCH6	0x0006
1.0	TWEJ_PATCHT	0x0007

Triplet Constants

Data Groups (DG_)

Note: These are bits in a mask.

Version	Data Group (DG_)	Numeric ID
1.0	DG_CONTROL	0x0001L
1.0	DG_IMAGE	0x0002L
1.8	DG_AUDIO	0x0004L
2.1	DG_MASK	0xFFFFL

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

Data Argument Types (DAT_)

Version	Data Argument Types (DAT_)	Numeric ID
1.0	DAT_NULL	0x0000
1.0	DAT_CUSTOMBASE	0x8000
1.0	DAT_CAPABILITY	0x0001
1.0	DAT_EVENT	0x0002
1.0	DAT_IDENTITY	0x0003
1.0	DAT_PARENT	0x0004
1.0	DAT_PENDINGXFERS	0x0005
1.0	DAT_SETUPMEMXFER	0x0006
1.0	DAT_SETUPFILEXFER	0x0007
1.0	DAT_STATUS	0x0008
1.0	DAT_USERINTERFACE	0x0009
1.0	DAT_XFERGROUP	0x000A
1.0	DAT_CUSTOMDSDATA	0x000C
1.8	DAT_DEVICEEVENT	0x000D
1.8	DAT_FILESYSTEM	0x000E
1.8	DAT_PASSTHRU	0x000F
2.0	DAT_CALLBACK	0x0010
2.1	DAT_STATUSUTF8	0x0011
2.2	DAT_CALLBACK2	0x0012
2.4	DAT_METRICS	0x0013
2.4	DAT_TWAINDIRECT	0x0014
2.0	DAT_ENTRYPOINT	0x0403
1.0	DAT_IMAGEINFO	0x0101
1.0	DAT_IMAGELAYOUT	0x0102
1.0	DAT_IMAGEMEMXFER	0x0103
1.0	DAT_IMAGENATIVEXFER	0x0104
1.0	DAT_IMAGEFILEXFER	0x0105
1.0	DAT_CIECOLOR	0x0106
1.0	DAT_GRAYRESPONSE	0x0107
1.0	DAT_RGBRESPONSE	0x0108
1.0	DAT_JPEGCOMPRESSION	0x0109
1.0	DAT_PALETTE8	0x010A
1.7	DAT_EXTIMAGEINFO	0x010B
2.2	DAT_FILTER	0x010C
1.8	DAT_AUDIOFILEXFER	0x0201
1.8	DAT_AUDIOINFO	0x0202
1.8	DAT_AUDIONATIVEXFER	0x0203
1.91	DAT_ICCPROFILE	0x0401
1.91	DAT_IMAGEMEMFILEXFER	0x0402

Message (MSG_)

Version	Message (MSG_)	Numeric ID
1.0	MSG_NULL	0x0000
1.0	MSG_CUSTOMBASE	0x8000
1.0	MSG_GET	0x0001
1.0	MSG_GETCURRENT	0x0002
1.0	MSG_GETDEFAULT	0x0003
1.0	MSG_GETFIRST	0x0004
1.0	MSG_GETNEXT	0x0005
1.0	MSG_SET	0x0006
1.0	MSG_RESET	0x0007
1.0	MSG_QUERYSUPPORT	0x0008
2.2	MSG_SETCONSTRAINT	0x000c
2.1	MSG_GETHELP	0x0009
2.1	MSG_GETLABEL	0x000A
2.1	MSG_GETLABELENUM	0x000B
1.0	MSG_XFERREADY	0x0101
1.0	MSG_CLOSEDSREQ	0x0102
1.0	MSG_CLOSEDSOK	0x0103
1.0	MSG_DEVICEEVENT	0x0104
1.0	MSG_OPENDSM	0x0301
1.0	MSG_CLOSEDSM	0x0302
1.0	MSG_OPENDS	0x0401
1.0	MSG_CLOSEDS	0x0402
1.0	MSG_USERSELECT	0x0403
1.0	MSG_DISABLEDS	0x0501
1.0	MSG_ENABLEDS	0x0502
1.0	MSG_ENABLEDSUIONLY	0x0503
1.0	MSG_PROCESSEVENT	0x0601
1.0	MSG_ENDXFER	0x0701
1.0	MSG_STOPFEEDER	0x0702
1.8	MSG_CHANGEDIRECTORY	0x0801
1.8	MSG_CREATEDIRECTORY	0x0802
1.8	MSG_DELETE	0x0803
1.8	MSG_FORMATMEDIA	0x0804
1.8	MSG_GETCLOSE	0x0805
1.8	MSG_GETFIRSTFILE	0x0806
1.8	MSG_GETINFO	0x0807
1.8	MSG_GETNEXTFILE	0x0808
1.8	MSG_RENAME	0x0809
1.8	MSG_COPY	0x080A
1.8	MSG_AUTOMATICCAPTUREDIRECTORY	0x080B
1.0	MSG_PASSTHRU	0x0901
1.0	MSG_REGISTER_CALLBACK	0x0902
1.91	MSG_RESETALL	0x0A01

Return Code and Condition Code Constants

Return Codes (TWRC_)

Version	Constant ID	Numeric ID
1.0	TWRC_CUSTOMBASE	0x8000
1.0	TWRC_SUCCESS	0
1.0	TWRC_FAILURE	1
1.0	TWRC_CHECKSTATUS	2
1.0	TWRC_CANCEL	3
1.0	TWRC_DSEVENT	4
1.0	TWRC_NOTDSEVENT	5
1.0	TWRC_XFERDONE	6
1.0	TWRC_ENDOFLIST	7
1.0	TWRC_INFONOTSUPPORTED	8
1.0	TWRC_DATANOTAVAILABLE	9
2.2	TWRC_BUSY	10
2.2	TWRC_SCANNERLOCKED	11

Condition Codes (TWCC_)

Version	Constant ID	Numeric ID
1.0	TWCC_CUSTOMBASE	0x8000
1.0	TWCC_SUCCESS	0
1.0	TWCC_BUMMER	1
1.0	TWCC_LOWMEMORY	2
1.0	TWCC_NODS	3
1.0	TWCC_MAXCONNECTIONS	4
1.0	TWCC_OPERATIONERROR	5
1.0	TWCC_BADCAP	6
1.0	TWCC_BADPROTOCOL	9
1.0	TWCC_BADVALUE	10
1.0	TWCC_SEQERROR	11
1.0	TWCC_BADDEST	12
1.0	TWCC_CAPUNSUPPORTED	13
1.0	TWCC_CAPBADOPERATION	14
1.0	TWCC_CAPSEQERROR	15
1.8	TWCC_DENIED	16
1.8	TWCC_FILEEXISTS	17
1.8	TWCC_FILENOTFOUND	18
1.8	TWCC_NOTEMPTY	19
1.8	TWCC_PAPERJAM	20
1.8	TWCC_PAPERDOUBLEFEED	21
1.8	TWCC_FILEWRITEERROR	22
1.8	TWCC_CHECKDEVICEONLINE	23

Version	Constant ID	Numeric ID
2.0	TWCC_INTERLOCK	24
2.0	TWCC_DAMAGEDCORNER	25
2.0	TWCC_FOCUSERROR	26
2.0	TWCC_DOCTOOLIGHT	27
2.0	TWCC_DOCTOODARK	28
2.1	TWCC_NOMEDIA	29

Extended Image Information Constants

TWEI_Codes

Version	Constant ID	Numeric ID
1.7	TWEI_BARCODEX	0x1200
1.7	TWEI_BARCODEY	0x1201
1.7	TWEI_BARCODETEXT	0x1202
1.7	TWEI_BARCODETYPE	0x1203
1.7	TWEI_DESHADETOP	0x1204
1.7	TWEI_DESHADELEFT	0x1205
1.7	TWEI_DESHADEHEIGHT	0x1206
1.7	TWEI_DESHADEWIDTH	0x1207
1.7	TWEI_DESHADESIZE	0x1208
1.7	TWEI_SPECKLESREMOVED	0x1209
1.7	TWEI_HORZLINEXCOORD	0x120A
1.7	TWEI_HORZLINEYCOORD	0x120B
1.7	TWEI_HORZLINELENGTH	0x120C
1.7	TWEI_HORZLINETHICKNESS	0x120D
1.7	TWEI_VERTLINEXCOORD	0x120E
1.7	TWEI_VERTLINEYCOORD	0x120F
1.7	TWEI_VERTLINELENGTH	0x1210
1.7	TWEI_VERTLINETHICKNESS	0x1211
1.7	TWEI_PATCHCODE	0x1212
1.7	TWEI_ENDORSEDTEXT	0x1213
1.7	TWEI_FORMCONFIDENCE	0x1214
1.7	TWEI_FORMTEMPLATEMATCH	0x1215
1.7	TWEI_FORMTEMPLATEPAGEMATCH	0x1216
1.7	TWEI_FORMHORZDOCOFFSET	0x1217
1.7	TWEI_FORMVERTDOCOFFSET	0x1218
1.7	TWEI_BARCODECOUNT	0x1219
1.7	TWEI_BARCODECONFIDENCE	0x121A
1.7	TWEI_BARCODEROTATION	0x121B
1.7	TWEI_BARCODETEXTLENGTH	0x121C
1.7	TWEI_DESHADECOUNT	0x121D
1.7	TWEI_DESHADEBLACKCOUNTOLD	0x121E
1.7	TWEI_DESHADEBLACKCOUNTNEW	0x121F
1.7	TWEI_DESHADEBLACKRLMIN	0x1220
1.7	TWEI_DESHADEBLACKRLMAX	0x1221

Version	Constant ID	Numeric ID
1.7	TWEI_DESHADEWHITECOUNTOLD	0x1222
1.7	TWEI_DESHADEWHITECOUNTNEW	0x1223
1.7	TWEI_DESHADEWHITERLMIN	0x1224
1.7	TWEI_DESHADEWHITERLAVE	0x1225
1.7	TWEI_DESHADEWHITERLMAX	0x1226
1.7	TWEI_BLACKSPECKLESREMOVED	0x1227
1.7	TWEI_WHITESPECKLESREMOVED	0x1228
1.7	TWEI_HORZLINECOUNT	0x1229
1.7	TWEI_VERTLINECOUNT	0x122A
1.7	TWEI_DESKEWSTATUS	0x122B
1.7	TWEI_SKEWORIGINALANGLE	0x122C
1.7	TWEI_SKEWFINALANGLE	0x122D
1.7	TWEI_SKEWCONFIDENCE	0x122E
1.7	TWEI_SKEWWINDOWX1	0x122F
1.7	TWEI_SKEWWINDOWY1	0x1230
1.7	TWEI_SKEWWINDOWX2	0x1231
1.7	TWEI_SKEWWINDOWY2	0x1232
1.7	TWEI_SKEWWINDOWX3	0x1233
1.7	TWEI_SKEWWINDOWY3	0x1234
1.7	TWEI_SKEWWINDOWX4	0x1235
1.7	TWEI_SKEWWINDOWY4	0x1236
1.9	TWEI_BOOKNAME	0x1238
1.9	TWEI_CHAPTERNUMBER	0x1239
1.9	TWEI_DOCUMENTNUMBER	0x123A
1.9	TWEI_PAGENUMBER	0x123B
1.9	TWEI_CAMERA	0x123C
1.9	TWEI_FRAMENUMBER	0x123D
1.9	TWEI_FRAME	0x123E
1.9	TWEI_PIXELFLAVOR	0x123F
1.91	TWEI_ICCPROFILE	0x1240
1.91	TWEI_LASTSEGMENT	0x1241
1.91	TWEI_SEGMENTNUMBER	0x1242
2.0	TWEI_MAGDATA	0x1243
2.0	TWEI_MAGTYPE	0x1244
2.0	TWEI_PAGESIDE	0x1245
2.0	TWEI_FILESYSTEMSOURCE	0x1246
2.1	TWEI_IMAGEMERGED	0x1247
2.1	TWEI_MAGDATALENGTH	0x1248
2.2	TWEI_PAPERCOUNT	0x1249
2.3	TWEI_PRINTERTEXT	0x124A
2.4	TWEI_TWAINDIRECTMETADATA	0x124B

TWEI_BARCODEROTATION

Version	Constant ID	Numeric ID
1.7	TWBCOR_ROT0	0
1.7	TWBCOR_ROT90	1
1.7	TWBCOR_ROT180	2
1.7	TWBCOR_ROT270	3
1.7	TWBCOR_ROTX	4

TWEI_DESKEWSTATUS

Version	Constant ID	Numeric ID
1.7	TWDSK_SUCCESS	0
1.7	TWDSK_REPORTONLY	1
1.7	TWDSK_FAIL	2
1.7	TWDSK_DISABLED	3

TWEI_MAGTYPE

Version	Constant ID	Numeric ID
2.0	TWMD_MICR	0
2.1	TWMD_RAW	1
2.1	TWMD_INVALID	2

TWEI_PATCHCODE

Version	Constant ID	Numeric ID
1.7	TWPCH_PATCH1	0
1.7	TWPCH_PATCH2	1
1.7	TWPCH_PATCH3	2
1.7	TWPCH_PATCH4	3
1.7	TWPCH_PATCH6	4
1.7	TWPCH_PATCHT	5

Capability Constants

Version	Constant ID	Numeric ID
1.0	CAP_CUSTOMBASE	0x8000
1.0	CAP_XFERCOUNT	0x0001
1.0	ICAP_COMPRESSION	0x0100
1.0	ICAP_PIXELTYPE	0x0101
1.0	ICAP_UNITS	0x0102
1.0	ICAP_XFERMECH	0x0103
1.0	CAP_AUTHOR	0x1000

Version	Constant ID	Numeric ID
1.0	CAP_CAPTION	0x1001
1.0	CAP_FEEDERENABLED	0x1002
1.0	CAP_FEEDERLOADED	0x1003
1.0	CAP_TIMEDATE	0x1004
1.0	CAP_SUPPORTEDCAPS	0x1005
1.0	CAP_EXTENDEDCAPS	0x1006
1.0	CAP_AUTOFEED	0x1007
1.0	CAP_CLEARPAGE	0x1008
1.0	CAP_FEEDPAGE	0x1009
1.0	CAP_REWINDPAGE	0x100A
1.1	CAP_INDICATORS	0x100B
1.6	CAP_PAPERDETECTABLE	0x100D
1.6	CAP_UICONTROLLABLE	0x100E
1.6	CAP_DEVICEONLINE	0x100F
1.6	CAP_AUTOSCAN	0x1010
1.7	CAP_THUMBNAILSENABLED	0x1011
1.7	CAP_DUPLEX	0x1012
1.7	CAP_DUPLEXENABLED	0x1013
1.7	CAP_ENABLEDSUIONLY	0x1014
1.7	CAP_CUSTOMDSDATA	0x1015
1.7	CAP_ENDORSER	0x1016
1.7	CAP_JOBCONTROL	0x1017
1.8	CAP_ALARMS	0x1018
1.8	CAP_ALARMVOLUME	0x1019
1.8	CAP_AUTOMATICCAPTURE	0x101A
1.8	CAP_TIMEBEFOREFIRSTCAPTURE	0x101B
1.8	CAP_TIMEBETWEENCAPTURES	0x101C
1.8	CAP_MAXBATCHBUFFERS	0x101E
1.8	CAP_DEVICETIMEDATE	0x101F
1.8	CAP_POWERSUPPLY	0x1020
1.8	CAP_CAMERAPREVIEWUI	0x1021
1.8	CAP_DEVICEEVENT	0x1022
1.8	CAP_SERIALNUMBER	0x1024
1.8	CAP_PRINTER	0x1026

Version	Constant ID	Numeric ID
1.8	CAP_PRINTERENABLED	0x1027
1.8	CAP_PRINTERINDEX	0x1028
1.8	CAP_PRINTERMODE	0x1029
1.8	CAP_PRINTERSTRING	0x102A
1.8	CAP_PRINTERSUFFIX	0x102B
1.8	CAP_LANGUAGE	0x102C
1.8	CAP_FEEDERALIGNMENT	0x102D
1.8	CAP_FEEDERORDER	0x102E
1.8	CAP_REACQUIREALLOWED	0x1030
1.8	CAP_BATTERYMINUTES	0x1032
1.8	CAP_BATTERYPERCENTAGE	0x1033
1.91	CAP_CAMERASIDE	0x1034
1.91	CAP_SEGMENTED	0x1035
2.0	CAP_CAMERAENABLED	0x1036
2.0	CAP_CAMERAORDER	0x1037
2.0	CAP_MICRENABLED	0x1038
2.0	CAP_FEEDERPREP	0x1039
2.0	CAP_FEEDERPOCKET	0x103A
2.1	CAP_AUTOMATICSENSEMEDIUM	0x103B
2.1	CAP_CUSTOMINTERFACEGUID	0x103C
2.2	CAP_SUPPORTEDCAPSSEGMENTUNIQUE	0x103D
2.2	CAP_SUPPORTEDDATS	0x103E
2.2	CAP_DOUBLEFEEDDETECTION	0x103F
2.2	CAP_DOUBLEFEEDDETECTIONLENGTH	0x1040
2.2	CAP_DOUBLEFEEDDETECTIONSENSITIVITY	0x1041
2.2	CAP_DOUBLEFEEDDETECTIONRESPONSE	0x1042
2.2	CAP_PAPERHANDLING	0x1043
2.2	CAP_INDICATORSMODE	0x1044
2.2	CAP_PRINTERVERTICALOFFSET	0x1045
1.8	CAP_POWERSAVETIME	0x1046
2.3	CAP_PRINTERCHARROTATION	0x1047
2.3	CAP_PRINTERFONTSTYLE	0x1048
2.3	CAP_PRINTERINDEXLEADCHAR	0x1049
2.3	CAP_PRINTERINDEXMAXVALUE	0x104A

Version	Constant ID	Numeric ID
2.3	CAP_PRINTERINDEXNUMDIGITS	0x104B
2.3	CAP_PRINTERINDEXSTEP	0x104C
2.3	CAP_PRINTERINDEXTRIGGER	0x104D
2.3	CAP_PRINTERSTRINGPREVIEW	0x104E
2.4	CAP_SHEETCOUNT	0x104F
1.0	ICAP_AUTOBRIGHT	0x1100
1.0	ICAP_BRIGHTNESS	0x1101
1.0	ICAP_CONTRAST	0x1103
1.0	ICAP_CUSTHALFTONE	0x1104
1.0	ICAP_EXPOSURETIME	0x1105
1.0	ICAP_FILTER	0x1106
1.0	ICAP_FLASHUSED	0x1107
1.0	ICAP_GAMMA	0x1108
1.0	ICAP_HALFTONES	0x1109
1.0	ICAP_HIGHLIGHT	0x110A
1.0	ICAP_IMAGEFILEFORMAT	0x110C
1.0	ICAP_LAMPSTATE	0x110D
1.0	ICAP_LIGHTSOURCE	0x110E
1.0	ICAP_ORIENTATION	0x1110
1.0	ICAP_PHYSICALWIDTH	0x1111
1.0	ICAP_PHYSICALHEIGHT	0x1112
1.0	ICAP_SHADOW	0x1113
1.0	ICAP_FRAMES	0x1114
1.0	ICAP_XNATIVERESOLUTION	0x1116
1.0	ICAP_YNATIVERESOLUTION	0x1117
1.0	ICAP_XRESOLUTION	0x1118
1.0	ICAP_YRESOLUTION	0x1119
1.0	ICAP_MAXFRAMES	0x111A
1.0	ICAP_TILES	0x111B
1.0	ICAP_BITORDER	0x111C
1.0	ICAP_CCITTKFACTOR	0x111D
1.0	ICAP_LIGHTPATH	0x111E
1.0	ICAP_PIXELFLAVOR	0x111F
1.0	ICAP_PLANARCHUNKY	0x1120

Version	Constant ID	Numeric ID
1.0	ICAP_ROTATION	0x1121
1.0	ICAP_SUPPORTEDSIZES	0x1122
1.0	ICAP_THRESHOLD	0x1123
1.0	ICAP_XSCALING	0x1124
1.0	ICAP_YSCALING	0x1125
1.0	ICAP_BITORDERCODES	0x1126
1.0	ICAP_PIXELFLAVORCODES	0x1127
1.0	ICAP_JPEGPIXELTYPE	0x1128
1.0	ICAP_TIMEFILL	0x112A
1.0	ICAP_BITDEPTH	0x112B
1.5	ICAP_BITDEPTHREDUCTION	0x112C
1.6	ICAP_UNDEFINEDIMAGESIZE	0x112D
1.7	ICAP_IMAGEDATASET	0x112E
1.7	ICAP_EXTIMAGEINFO	0x112F
1.7	ICAP_MINUMUMHEIGHT	0x1130
1.7	ICAP_MINIMUMWIDTH	0x1131
2.0	ICAP_AUTODISCARDBLANKPAGES	0x1134
1.8	ICAP_FLIPROTATION	0x1136
1.8	ICAP_BARCODEDETECTIONENABLED	0x1137
1.8	ICAP_SUPPORTEDBARCODETYPES	0x1138
1.8	ICAP_BARCODEMAXSEARCHPRIORITIES	0x1139
1.8	ICAP_BARCODESEARCHPRIORITIES	0x113A
1.8	ICAP_BARCODESEARCHMODE	0x113B
1.8	ICAP_BARCODEMAXRETRIES	0x113C
1.8	ICAP_BARCODETIMEOUT	0x113D
1.8	ICAP_ZOOMFACTOR	0x113E
1.8	ICAP_PATCHCODEDETECTIONENABLED	0x113F
1.8	ICAP_SUPPORTEDPATCHCODETYPES	0x1140
1.8	ICAP_PATCHCODEMAXSEARCHPRIORITIES	0x1141
1.8	ICAP_PATCHCODESEARCHPRIORITIES	0x1142
1.8	ICAP_PATCHCODESEARCHMODE	0x1143
1.8	ICAP_PATCHCODEMAXRETRIES	0x1144
1.8	ICAP_PATCHCODETIMEOUT	0x1145
1.8	ICAP_FLASHUSED2	0x1146

Version	Constant ID	Numeric ID
1.8	ICAP_IMAGEFILTER	0x1147
1.8	ICAP_NOISEFILTER	0x1148
1.8	ICAP_OVERSCAN	0x1149
1.8	ICAP_AUTOMATICBORDERDETECTION	0x1150
1.8	ICAP_AUTOMATICDESKEW	0x1151
1.8	ICAP_AUTOMATICROTATE	0x1152
1.9	ICAP_JPEGQUALITY	0x1153
1.91	ICAP_FEEDERTYPE	0x1154
1.91	ICAP_ICCPROFILE	0x1155
2.0	ICAP_AUTOSIZE	0x1156
2.1	ICAP_AUTOMATICCROPUSESFRAME	0x1157
2.1	ICAP_AUTOMATICLENGTHDETECTION	0x1158
2.1	ICAP_AUTOMATICCOLORENABLED	0x1159
2.1	ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE	0x115A
2.1	ICAP_COLORMANAGEMENTENABLED	0x115B
2.1	ICAP_IMAGEMERGE	0x115C
2.1	ICAP_IMAGEMERGEHEIGHTTHRESHOLD	0x115D
2.1	ICAP_SUPPORTEDEXTIMAGEINFO	0x115E
2.2	ICAP_FILMTYPE	0x115F
2.2	ICAP_MIRROR	0x1160
2.2	ICAP_JPEGSUBSAMPLING	0x1161
1.8	ACAP_XFERMECH	0x1202

CAP_ALARMS

Version	Constant ID	Numeric ID
1.8	TWAL_ALARM	0
1.8	TWAL_FEEDERERROR	1
1.8	TWAL_FEEDERWARNING	2
1.8	TWAL_BARCODE	3
1.8	TWAL_DOUBLEFEED	4
1.8	TWAL_JAM	5
1.8	TWAL_PATCHCODE	6
1.8	TWAL_POWER	7
1.8	TWAL_SKEW	8

CAP_CAMERASIDE

Version	Constant ID	Numeric ID
1.91	TWCS_BOTH	0
1.91	TWCS_TOP	1
1.91	TWCS_BOTTOM	2

CAP_DEVICEEVENT

Version	Constant ID	Numeric ID
1.8	TWDE_CUSTOMEVENTS	0x8000
1.8	TWDE_CHECKAUTOMATICCAPTURE	0
1.8	TWDE_CHECKBATTERY	1
1.8	TWDE_CHECKDEVICEONLINE	2
1.8	TWDE_CHECKFLASH	3
1.8	TWDE_CHECKPOWERSUPPLY	4
1.8	TWDE_CHECKRESOLUTION	5
1.8	TWDE_DEVICEADDED	6
1.8	TWDE_DEVICEOFFLINE	7
1.8	TWDE_DEVICEREADY	8
1.8	TWDE_DEVICEREMOVED	9
1.8	TWDE_IMAGECAPTURED	10
1.8	TWDE_IMAGEDELETED	11
1.8	TWDE_PAPERDOUBLEFEED	12
1.8	TWDE_PAPERJAM	13
1.8	TWDE_LAMPFAILURE	14
1.8	TWDE_POWERSAVE	15
1.8	TWDE_POWERSAVENOTIFY	16

CAP_DOUBLEFEEDDETECTION

Version	Constant ID	Numeric ID
2.2	TWDF_ULTRASONIC	0
2.2	TWDF_BYLENGTH	1
2.2	TWDF_INFRARED	2

CAP_DOUBLEFEEDDETECTIONRESPONSE

Version	Constant ID	Numeric ID
2.2	TWDP_STOP	0
2.2	TWDP_STOPANDWAIT	1
2.2	TWDP_SOUND	2
2.2	TWDP_DONOTIMPRINT	3

Refer to CAP_DOUBLEFEEDDETECTIONRESPONSE in Chapter 10.

CAP_DOUBLEFEEDDETECTIONSENSITIVITY

Version	Constant ID	Numeric ID
2.2	TWUS_LOW	0
2.2	TWUS_MEDIUM	1
2.2	TWUS_HIGH	2

CAP_DUPLEX

Version	Constant ID	Numeric ID
1.7	TWDX_NONE	0
1.7	TWDX_1PASSDUPLEX	1
1.7	TWDX_2PASSDUPLEX	2

CAP_FEEDERALIGNMENT

Version	Constant ID	Numeric ID
1.8	TWFA_NONE	0
1.8	TWFA_LEFT	1
1.8	TWFA_CENTER	2
1.8	TWFA_RIGHT	3

CAP_FEEDERORDER

Version	Constant ID	Numeric ID
1.8	TWFO_FIRSTPAGEFIRST	0
1.8	TWFO_LASTPAGEFIRST	1

CAP_FEEDERPOCKET

Version	Constant ID	Numeric ID
2.0	TWFP_POCKETERROR	0
2.0	TWFP_POCKET1	1
2.0	TWFP_POCKET2	2
2.0	TWFP_POCKET3	3
2.0	TWFP_POCKET4	4
2.0	TWFP_POCKET5	5
2.0	TWFP_POCKET6	6
2.0	TWFP_POCKET7	7
2.0	TWFP_POCKET8	8
2.0	TWFP_POCKET9	9
2.0	TWFP_POCKET10	10
2.0	TWFP_POCKET11	11
2.0	TWFP_POCKET12	12
2.0	TWFP_POCKET13	13
2.0	TWFP_POCKET14	14
2.0	TWFP_POCKET15	15
2.0	TWFP_POCKET16	16

CAP_INDICATORSMODE

Version	Constant ID	Numeric ID
2.0	TWCI_INFO	0
2.0	TWCI_WARNING	1
2.0	TWCI_ERROR	2
2.0	TWCI_WARMUP	3

CAP_JOBCONTROL

Version	Constant ID	Numeric ID
1.7	TWJC_NONE	0
1.7	TWJC_JSIC	1
1.7	TWJC_JSIS	2
1.7	TWJC_JSXC	3
1.7	TWJC_JSXS	4

CAP_LANGUAGE

See “Language Constants” on page 8-92.

CAP_PAPERHANDLING

Version	Constant ID	Numeric ID
2.0	TWPH_NORMAL	0
2.0	TWPH_FRAGILE	1
2.0	TWPH_THICK	2
2.0	TWPH_TRIFOLD	3
2.0	TWPH_PHOTOGRAPH	4

CAP_POWERSUPPLY

Version	Constant ID	Numeric ID
1.0	TWPS_EXTERNAL	0
1.0	TWPS_BATTERY	1

CAP_PRINTER

Version	Constant ID	Numeric ID
1.8	TWPR_IMPRINTERTOPBEFORE	0
1.8	TWPR_IMPRINTERTOPAFTER	1
1.8	TWPR_IMPRINTERBOTTOMBEFORE	2
1.8	TWPR_IMPRINTERBOTTOMAFTER	3
1.8	TWPR_ENDORSERTOPBEFORE	4
1.8	TWPR_ENDORSERTOPAFTER	5
1.8	TWPR_ENDORSERBOTTOMBEFORE	6
1.8	TWPR_ENDORSERBOTTOMAFTER	7

CAP_PRINTERMODE

Version	Constant ID	Numeric ID
1.8	TWPM_SINGLESTRING	0
1.8	TWPM_MULTISTRING	1
1.8	TWPM_COMPOUNDSTRING	2

CAP_PRINTERFONTSTYLE

Version	Constant ID	Numeric ID
2.3	TWPF_NORMAL	0
2.3	TWPF_BOLD	1
2.3	TWPF_ITALIC	2
2.3	TWPF_LARGESIZE	3
2.3	TWPF_SMALLSIZE	4

CAP_PRINTERINDEXTRIGGER

Version	Constant ID	Numeric ID
2.3	TWCT_PAGE	0
2.3	TWCT_PATCH1	1
2.3	TWCT_PATCH2	2
2.3	TWCT_PATCH3	3
2.3	TWCT_PATCH4	4
2.3	TWCT_PATCHT	5
2.3	TWCT_PATCH6	6

CAP_SEGMENTED

Version	Constant ID	Numeric ID
1.91	TWSG_NONE	0
1.91	TWSG_AUTO	1
2.2	TWSG_MANUAL	2

ICAP_AUTODISCARDBLANKPAGES

Version	Constant ID	Numeric ID
2.0	TWBP_DISABLE	-2
2.0	TWBP_AUTO	-1

ICAP_AUTOSIZE

Version	Constant ID	Numeric ID
2.0	TWAS_NONE	0
2.0	TWAS_AUTO	1
2.0	TWAS_CURRENT	2

ICAP_BARCODESEARCHMODE

Version	Constant ID	Numeric ID
1.8	TWBD_HORZ	0
1.8	TWBD_VERT	1
1.8	TWBD_HORZVERT	2
1.8	TWBD_VERTHORZ	3

ICAP_SUPPORTEDBARCODETYPES

Version	Constant ID	Numeric ID
1.7	TWBT_3OF9	0
1.7	TWBT_2OF5INTERLEAVED	1
1.7	TWBT_2OF5NONINTERLEAVED	2
1.7	TWBT_CODE93	3
1.7	TWBT_CODE128	4
1.7	TWBT_UCC128	5
1.7	TWBT_CODABAR	6
1.7	TWBT_UPCA	7
1.7	TWBT_UPCE	8
1.7	TWBT_EAN8	9
1.7	TWBT_EAN13	10
1.7	TWBT_POSTNET	11
1.7	TWBT_PDF417	12
1.8	TWBT_2OF5INDUSTRIAL	13
1.8	TWBT_2OF5MATRIX	14
1.8	TWBT_2OF5DATALOGIC	15
1.8	TWBT_2OF5IATA	16
1.8	TWBT_3OF9FULLASCII	17
1.8	TWBT_CODABARWITHSTARTSTOP	18
1.8	TWBT_MAXICODE	19
2.2	TWBT_QRCODE	20

ICAP_BITDEPTHREDUCTION

Version	Constant ID	Numeric ID
1.5	TWBR_THRESHOLD	0
1.5	TWBR_HALFTONE	1
1.5	TWBR_CUSTHALFTONE	2
1.5	TWBR_DIFFUSION	3
2.2	TWBR_DYNAMICTHRESHOLD	4

ICAP_BITORDER

Version	Constant ID	Numeric ID
1.0	TWBO_LSBFIRST	0
1.0	TWBO_MSBFIRST	1

ICAP_COMPRESSION

Version	Constant ID	Numeric ID
1.0	TWCP_NONE	0
1.0	TWCP_PACKBITS	1
1.0	TWCP_GROUP31D	2
1.0	TWCP_GROUP31DEOL	3
1.0	TWCP_GROUP32D	4
1.0	TWCP_GROUP4	5
1.0	TWCP_JPEG	6
1.0	TWCP_LZW	7
1.7	TWCP_JBIG	8
1.8	TWCP_PNG	9
1.8	TWCP_RLE4	10
1.8	TWCP_RLE8	11
1.8	TWCP_BITFIELDS	12
2.2	TWCP_ZIP	13
2.2	TWCP_JPEG2000	14

ICAP_FEEDERTYPE

Version	Constant ID	Numeric ID
1.91	TWFE_GENERAL	0
1.91	TWFE_PHOTO	1

ICAP_FILTER

Version	Constant ID	Numeric ID
1.0	TWFT_RED	0
1.0	TWFT_GREEN	1
1.0	TWFT_BLUE	2
1.0	TWFT_NONE	3
1.0	TWFT_WHITE	4
1.0	TWFT_CYAN	5
1.0	TWFT_MAGENTA	6
1.0	TWFT_YELLOW	7
1.0	TWFT_BLACK	8

ICAP_FLASHUSED2

Version	Constant ID	Numeric ID
1.0	TWFL_NONE	0
1.0	TWFL_OFF	1
1.0	TWFL_ON	2
1.0	TWFL_AUTO	3
1.0	TWFL_REDEYE	4

ICAP_FILMTYPE

Version	Constant ID	Numeric ID
2.2	TWFM_POSITIVE	0
2.2	TWFM_NEGATIVE	1

ICAP_FLIPROTATION

Version	Constant ID	Numeric ID
1.8	TWFR_BOOK	0
1.8	TWFR_FANFOLD	1

ICAP_ICCPROFILE

Version	Constant ID	Numeric ID
1.91	TWIC_NONE	0
1.91	TWIC_LINK	1
1.91	TWIC_EMBED	2

ICAP_IMAGEFILEFORMAT

Version	Constant ID	Numeric ID
1.0	TWFF_TIFF	0
1.0	TWFF_PICT	1
1.0	TWFF_BMP	2
1.0	TWFF_XBM	3
1.0	TWFF_JFIF	4
1.0	TWFF_FPX	5
1.0	TWFF_TIFFMULTI	6
1.0	TWFF_PNG	7
1.0	TWFF_SPIFF	8
1.0	TWFF_EXIF	9
1.91	TWFF_PDF	10
1.91	TWFF_JP2	11
	removed	12
1.91	TWFF_JPX	13
1.91	TWFF_DEJAVU	14
2.0	TWFF_PDFA	15
2.1	TWFF_PDFA2	16
2.4	TWFF_PDFRASTER	17

ICAP_IMAGEFILTER

Version	Constant ID	Numeric ID
1.8	TWIF_NONE	0
1.8	TWIF_AUTO	1
1.8	TWIF_LOWPASS	2
1.8	TWIF_BANDPASS	3
1.8	TWIF_HIGHPASS	4
1.8	TWIF_TEXT	TWIF_BANDPASS
1.8	TWIF_FINELINE	TWIF_HIGHPASS

ICAP_IMAGEMERGE

Version	Constant ID	Numeric ID
2.1	TWIM_NONE	0
2.1	TWIM_FRONTONTOP	1
2.1	TWIM_FRONTONBOTTOM	2
2.1	TWIM_FRONTONLEFT	3
2.1	TWIM_FRONTONRIGHT	4

ICAP_JPEGQUALITY

Version	Constant ID	Numeric ID
1.9	TWJQ_UNKNOWN	-4
1.9	TWJQ_LOW	-3
1.9	TWJQ_MEDIUM	-2
1.9	TWJQ_HIGH	-1

ICAP_JPEGSUBSAMPLING

Version	Constant ID	Numeric ID
2.2	TWJS_444YCBCR	0
2.2	TWJS_444RGB	1
2.2	TWJS_422	2
2.2	TWJS_421	3
2.2	TWJS_411	4
2.2	TWJS_420	5
2.2	TWJS_410	6
2.2	TWJS_311	7

ICAP_LIGHTPATH

Version	Constant ID	Numeric ID
1.0	TWLP_REFLECTIVE	0
1.0	TWLP_TRANSMISSIVE	1

ICAP_LIGHTSOURCE

Version	Constant ID	Numeric ID
1.0	TWLS_RED	0
1.0	TWLS_GREEN	1
1.0	TWLS_BLUE	2
1.0	TWLS_NONE	3
1.0	TWLS_WHITE	4
1.0	TWLS_UV	5
1.0	TWLS_IR	6

ICAP_MIRROR

Version	Constant ID	Numeric ID
2.2	TWMR_NONE	0
2.2	TWMR_VERTICAL	1
2.2	TWMR_HORIZONTAL	2

ICAP_NOISEFILTER

Version	Constant ID	Numeric ID
1.8	TWNF_NONE	0
1.8	TWNF_AUTO	1
1.8	TWNF_LONEPIXEL	2
1.8	TWNF_MAJORITYRULE	3

ICAP_ORIENTATION

Version	Constant ID	Numeric ID
1.0	TWOR_ROT0	0
1.0	TWOR_ROT90	1
1.0	TWOR_ROT180	2
1.0	TWOR_ROT270	3
1.0	TWOR_PORTRAIT	TWOR_ROT0
1.0	TWOR_LANDSCAPE	TWOR_ROT270
2.0	TWOR_AUTO	4
2.0	TWOR_AUTOTEXT	5
2.0	TWOR_AUTOPICTURE	6

ICAP_OVERSCAN

Version	Constant ID	Numeric ID
1.8	TWOV_NONE	0
1.8	TWOV_AUTO	1
1.8	TWOV_TOPBOTTOM	2
1.8	TWOV_LEFTRIGHT	3
1.8	TWOV_ALL	4

ICAP_PLANARCHUNKY

Version	Constant ID	Numeric ID
1.0	TWPC_CHUNKY	0
1.0	TWPC_PLANAR	1

ICAP_PIXELFLAVOR

Version	Constant ID	Numeric ID
1.0	TWPF_CHOCOLATE	0
1.0	TWPF_VANILLA	1

ICAP_PIXELTYPE

Version	Constant ID	Numeric ID
1.0	TWPT_BW	0
1.0	TWPT_GRAY	1
1.0	TWPT_RGB	2
1.0	TWPT_PALETTE	3
1.0	TWPT_CMY	4
1.0	TWPT_CMYK	5
1.0	TWPT_YUV	6
1.0	TWPT_YUVK	7
1.0	TWPT_CIEXYZ	8
1.0	TWPT_LAB	9
1.91	TWPT_SRGB	10
1.91	TWPT_SCRGB	11
2.0	TWPT_INFRARED	16

ICAP_SUPPORTEDSIZES

Version	Constant ID	Numeric ID
1.0	TWSS_NONE	0
1.0	TWSS_A4	1
1.0	TWSS_JISB5	2
1.0	TWSS_USLETTER	3
1.0	TWSS_USLEGAL	4
1.5	TWSS_A5	5
1.5	TWSS_ISOB4	6
1.5	TWSS_ISOB6	7
	// removed	
1.7	TWSS_USLEDGER	9
1.7	TWSS_USEXECUTIVE	10
1.7	TWSS_A3	11
1.7	TWSS_ISOB3	12
1.7	TWSS_A6	13
1.7	TWSS_C4	14
1.7	TWSS_C5	15
1.7	TWSS_C6	16

Version	Constant ID	Numeric ID
1.8	TWSS_4A0	17
1.8	TWSS_2A0	18
1.8	TWSS_A0	19
1.8	TWSS_A1	20
1.8	TWSS_A2	21
1.8	TWSS_A7	22
1.8	TWSS_A8	23
1.8	TWSS_A9	24
1.8	TWSS_A10	25
1.8	TWSS_ISOB0	26
1.8	TWSS_ISOB1	27
1.8	TWSS_ISOB2	28
1.8	TWSS_ISOB5	29
1.8	TWSS_ISOB7	30
1.8	TWSS_ISOB8	31
1.8	TWSS_ISOB9	32
1.8	TWSS_ISOB10	33
1.8	TWSS_JISB0	34
1.8	TWSS_JISB1	35
1.8	TWSS_JISB2	36
1.8	TWSS_JISB3	37
1.8	TWSS_JISB4	38
1.8	TWSS_JISB6	39
1.8	TWSS_JISB7	40
1.8	TWSS_JISB8	41
1.8	TWSS_JISB9	42
1.8	TWSS_JISB10	43
1.8	TWSS_C0	44
1.8	TWSS_C1	45
1.8	TWSS_C2	46
1.8	TWSS_C3	47
1.8	TWSS_C7	48
1.8	TWSS_C8	49
1.8	TWSS_C9	50
1.8	TWSS_C10	51
1.8	TWSS_USSTATEMENT	52
1.8	TWSS_BUSINESSCARD	53
2.1	TWSS_MAXSIZE	54

ICAP_XFERMECH

Version	Constant ID	Numeric ID
1.0	TWSX_NATIVE	0
1.0	TWSX_FILE	1
1.0	TWSX_MEMORY //removed	2
1.91	TWSX_MEMFILE	4

ICAP_UNITS

Version	Constant ID	Numeric ID
1.0	TWUN_INCHES	0
1.0	TWUN_CENTIMETERS	1
1.0	TWUN_PICAS	2
1.0	TWUN_POINTS	3
1.0	TWUN_TWIPS	4
1.0	TWUN_PIXELS	5
1.91	TWUN_MILLIMETERS	6

Language Constants

Language

Version	Constant ID	Numeric ID
1.8	TWLG_USERLOCALE	-1
1.0	TWLG_DAN	0
1.0	TWLG_DUT	1
1.0	TWLG_ENG	2
1.0	TWLG_FCF	3
1.0	TWLG_FIN	4
1.0	TWLG_FRN	5
1.0	TWLG_GER	6
1.0	TWLG_ICE	7
1.0	TWLG_ITN	8
1.0	TWLG_NOR	9
1.0	TWLG_POR	10
1.0	TWLG_SPA	11
1.0	TWLG_SWE	12
1.0	TWLG_USA	13
1.8	TWLG_AFRIKAANS	14
1.8	TWLG_ALBANIA	15
1.8	TWLG_ARABIC	16
1.8	TWLG_ARABIC_ALGERIA	17
1.8	TWLG_ARABIC_BAHRAIN	18
1.8	TWLG_ARABIC_EGYPT	19
1.8	TWLG_ARABIC_IRAQ	20
1.8	TWLG_ARABIC_JORDAN	21
1.8	TWLG_ARABIC_KUWAIT	22
1.8	TWLG_ARABIC_LEBANON	23
1.8	TWLG_ARABIC_LIBYA	24
1.8	TWLG_ARABIC_MOROCCO	25

Version	Constant ID	Numeric ID
1.8	TWLG_ARABIC_OMAN	26
1.8	TWLG_ARABIC_QATAR	27
1.8	TWLG_ARABIC_SAUDIARABIA	28
1.8	TWLG_ARABIC_SYRIA	29
1.8	TWLG_ARABIC_TUNISIA	30
1.8	TWLG_ARABIC_UAE	31
1.8	TWLG_ARABIC_YEMEN	32
1.8	TWLG_BASQUE	33
1.8	TWLG_BYELORUSSIAN	34
1.8	TWLG_BULGARIAN	35
1.8	TWLG_CATALAN	36
1.8	TWLG_CHINESE	37
1.8	TWLG_CHINESE_HONGKONG	38
1.8	TWLG_CHINESE_PRC	39
1.8	TWLG_CHINESE_SINGAPORE	40
1.8	TWLG_CHINESE_SIMPLIFIED	41
1.8	TWLG_CHINESE_TAIWAN	42
1.8	TWLG_CHINESE_TRADITIONAL	43
1.8	TWLG_CROATIA	44
1.8	TWLG_CZECH	45
1.8	TWLG_DANISH	TWLG_DAN
1.8	TWLG_DUTCH	TWLG_DUT
1.8	TWLG_DUTCH_BELGIAN	46
1.8	TWLG_ENGLISH	TWLG_ENG
1.8	TWLG_ENGLISH_AUSTRALIAN	47
1.8	TWLG_ENGLISH_CANADIAN	48
1.8	TWLG_ENGLISH_IRELAND	49
1.8	TWLG_ENGLISH_NEWZEALAND	50

Version	Constant ID	Numeric ID
1.8	TWLG_ENGLISH_SOUTHAFRICA	51
1.8	TWLG_ENGLISH_UK	52
1.8	TWLG_ENGLISH_USA	TWLG_USA
1.8	TWLG_ESTONIAN	53
1.8	TWLG_FAEROESE	54
1.8	TWLG_FARSI	55
1.8	TWLG_FINNISH	TWLG_FIN
1.8	TWLG_FRENCH	TWLG_FRN
1.8	TWLG_FRENCH_BELGIAN	56
1.8	TWLG_FRENCH_CANADIAN	TWLG_FCF
1.8	TWLG_FRENCH_LUXEMBOURG	57
1.8	TWLG_FRENCH_SWISS	58
1.8	TWLG_GERMAN	TWLG_GER
1.8	TWLG_GERMAN_AUSTRIAN	59
1.8	TWLG_GERMAN_LUXEMBOURG	60
1.8	TWLG_GERMAN_LIECHTENSTEIN	61
1.8	TWLG_GERMAN_SWISS	62
1.8	TWLG_GREEK	63
1.8	TWLG_HEBREW	64
1.8	TWLG_HUNGARIAN	65
1.8	TWLG_ICELANDIC	TWLG_ICE
1.8	TWLG_INDONESIAN	66
1.8	TWLG_ITALIAN	TWLG_ITN
1.8	TWLG_ITALIAN_SWISS	67
1.8	TWLG_JAPANESE	68
1.8	TWLG_KOREAN	69
1.8	TWLG_KOREAN_JOHAB	70
1.8	TWLG_LATVIAN	71
1.8	TWLG_LITHUANIAN	72
1.8	TWLG_NORWEGIAN	TWLG_NOR
1.8	TWLG_NORWEGIAN_BOKMAL	73
1.8	TWLG_NORWEGIAN_NYNORSK	74
1.8	TWLG_POLISH	75
1.8	TWLG_PORTUGUESE	TWLG_POR
1.8	TWLG_PORTUGUESE_BRAZIL	76
1.8	TWLG_ROMANIAN	77
1.8	TWLG_RUSSIAN	78
1.8	TWLG_SERBIAN_LATIN	79
1.8	TWLG_SLOVAK	80
1.8	TWLG_SLOVENIAN	81
1.8	TWLG_SPANISH	TWLG_SPA
1.8	TWLG_SPANISH_MEXICAN	82
1.8	TWLG_SPANISH_MODERN	83
1.8	TWLG_SWEDISH	TWLG_SWE
1.8	TWLG_THAI	84
1.8	TWLG_TURKISH	85

Version	Constant ID	Numeric ID
1.8	TWLG_UKRANIAN	86
1.8	TWLG_ASSAMESE	87
1.8	TWLG_BENGALI	88
1.8	TWLG_BIHARI	89
1.8	TWLG_BODO	90
1.8	TWLG_DOGRI	91
1.8	TWLG_GUJARATI	92
1.8	TWLG_HARYANVI	93
1.8	TWLG_HINDI	94
1.8	TWLG_KANNADA	95
1.8	TWLG_KASHMIRI	96
1.8	TWLG_MALAYALAM	97
1.8	TWLG_MARATHI	98
1.8	TWLG_MARWARI	99
1.8	TWLG_MEGHALAYAN	100
1.8	TWLG_MIZO	101
1.8	TWLG_NAGA	102
1.8	TWLG_ORISSI	103
1.8	TWLG_PUNJABI	104
1.8	TWLG_PUSHTU	105
1.8	TWLG_SERBIAN_CYRILLIC	106
1.8	TWLG_SIKKIMI	107
1.8	TWLG_SWEDISH_FINLAND	108
1.8	TWLG_TAMIL	109
1.8	TWLG_TELUGU	110
1.8	TWLG_TRIPURI	111
1.8	TWLG_URDU	112
1.8	TWLG_VIETNAMESE	113

Country [TWCY_]

Version	Constant ID	Numeric ID
1.0	TWCY_AFGHANISTAN	1001
1.0	TWCY_ALGERIA	213
1.0	TWCY_AMERICANSAMOA	684
1.0	TWCY_ANDORRA	33
1.0	TWCY_ANGOLA	1002
1.0	TWCY_ANGUILLA	8090
1.0	TWCY_ANTIGUA	8091
1.0	TWCY_ARGENTINA	54
1.0	TWCY_ARUBA	297
1.0	TWCY_ASCENSIONI	247
1.0	TWCY_AUSTRALIA	61

Version	Constant ID	Numeric ID
1.0	TWCY_AUSTRIA	43
1.0	TWCY_BAHAMAS	8092
1.0	TWCY_BAHRAIN	973
1.0	TWCY_BANGLADESH	880
1.0	TWCY_BARBADOS	8093
1.0	TWCY_BELGIUM	32
1.0	TWCY_BELIZE	501
1.0	TWCY_BENIN	229
1.0	TWCY_BERMUDA	8094
1.0	TWCY_BHUTAN	1003
1.0	TWCY_BOLIVIA	591
1.0	TWCY_BOTSWANA	267
1.0	TWCY_BRITAIN	6
1.0	TWCY_BRITVIRGINIS	8095
1.0	TWCY_BRAZIL	55
1.0	TWCY_BRUNEI	673
1.0	TWCY_BULGARIA	359
1.0	TWCY_BURKINAFASO	1004
1.0	TWCY_BURMA	1005
1.0	TWCY_BURUNDI	1006
1.0	TWCY_CAMAROON	237
1.0	TWCY_CANADA	2
1.0	TWCY_CAPEVERDEIS	238
1.0	TWCY_CAYMANIS	8096
1.0	TWCY_CENTRALAFREP	1007
1.0	TWCY_CHAD	1008
1.0	TWCY_CHILE	56
1.0	TWCY_CHINA	86
1.0	TWCY_CHRISTMASIS	1009
1.0	TWCY_COCOSIS	1009
1.0	TWCY_COLOMBIA	57
1.0	TWCY_COMOROS	1010
1.0	TWCY_CONGO	1011
1.0	TWCY_COOKIS	1012
1.0	TWCY_COSTARICA	506
1.0	TWCY_CUBA	5
1.0	TWCY_CYPRUS	357
1.0	TWCY_CZECHOSLOVAKIA	42
1.0	TWCY_DENMARK	45
1.0	TWCY_DJIBOUTI	1013
1.0	TWCY_DOMINICA	8097
1.0	TWCY_DOMINCANREP	8098
1.0	TWCY_EASTERIS	1014
1.0	TWCY_ECUADOR	593
1.0	TWCY_EGYPT	20
1.0	TWCY_ELSALVADOR	503
1.0	TWCY_EQGUINEA	1015
1.0	TWCY_ETHIOPIA	251

Version	Constant ID	Numeric ID
1.0	TWCY_FALKLANDIS	1016
1.0	TWCY_FAEROEIS	298
1.0	TWCY_FIJIISLANDS	679
1.0	TWCY_FINLAND	358
1.0	TWCY_FRANCE	33
1.0	TWCY_FRANTILLES	596
1.0	TWCY_FRGUIANA	594
1.0	TWCY_FRPOLYNEISA	689
1.0	TWCY_FUTANAIS	1043
1.0	TWCY_GABON	241
1.0	TWCY_GAMBIA	220
1.0	TWCY_GERMANY	49
1.0	TWCY_GHANA	233
1.0	TWCY_GIBRALTER	350
1.0	TWCY_GREECE	30
1.0	TWCY_GREENLAND	299
1.0	TWCY_GRENADA	8099
1.0	TWCY_GRENEDINES	8015
1.0	TWCY_GUADELOUPE	590
1.0	TWCY_GUAM	671
1.0	TWCY_GUANTANAMOBAY	5399
1.0	TWCY_GUATEMALA	502
1.0	TWCY_GUINEA	224
1.0	TWCY_GUINEABISSAU	1017
1.0	TWCY_GUYANA	592
1.0	TWCY_HAITI	509
1.0	TWCY_HONDURAS	504
1.0	TWCY_HONGKONG	852
1.0	TWCY_HUNGARY	36
1.0	TWCY_ICELAND	354
1.0	TWCY_INDIA	91
1.0	TWCY_INDONESIA	62
1.0	TWCY_IRAN	98
1.0	TWCY_IRAQ	964
1.0	TWCY_IRELAND	353
1.0	TWCY_ISRAEL	972
1.0	TWCY_ITALY	39
1.0	TWCY_IVORYCOAST	225
1.0	TWCY_JAMAICA	8010
1.0	TWCY_JAPAN	81
1.0	TWCY_JORDAN	962
1.0	TWCY_KENYA	254
1.0	TWCY_KIRIBATI	1018
1.0	TWCY_KOREA	82
1.0	TWCY_KUWAIT	965
1.0	TWCY_LAOS	1019
1.0	TWCY_LEBANON	1020
1.0	TWCY_LIBERIA	231

Version	Constant ID	Numeric ID
1.0	TWCY_LIBYA	218
1.0	TWCY_LIECHTENSTEIN	41
1.0	TWCY_LUXENBOURG	352
1.0	TWCY_MACAO	853
1.0	TWCY_MADAGASCAR	1021
1.0	TWCY_MALAWI	265
1.0	TWCY_MALAYSIA	60
1.0	TWCY_MALDIVES	960
1.0	TWCY_MALI	1022
1.0	TWCY_MALTA	356
1.0	TWCY_MARSHALLIS	692
1.0	TWCY_MAURITANIA	1023
1.0	TWCY_MAURITIUS	230
1.0	TWCY_MEXICO	3
1.0	TWCY_MICRONESIA	691
1.0	TWCY_MIQUELON	508
1.0	TWCY_MONACO	33
1.0	TWCY_MONGOLIA	1024
1.0	TWCY_MONTSERRAT	8011
1.0	TWCY_MOROCCO	212
1.0	TWCY_MOZAMBIQUE	1025
1.0	TWCY_NAMIBIA	264
1.0	TWCY_NAURU	1026
1.0	TWCY_NEPAL	977
1.0	TWCY_NETHERLANDS	31
1.0	TWCY_NETHANTILLES	599
1.0	TWCY_NEVIS	8012
1.0	TWCY_NEWCALEDONIA	687
1.0	TWCY_NEWZEALAND	64
1.0	TWCY_NICARAGUA	505
1.0	TWCY_NIGER	227
1.0	TWCY_NIGERIA	234
1.0	TWCY_NIUE	1027
1.0	TWCY_NORFOLKI	1028
1.0	TWCY_NORWAY	47
1.0	TWCY_OMAN	968
1.0	TWCY_PAKISTAN	92
1.0	TWCY_PALAU	1029
1.0	TWCY_PANAMA	507
1.0	TWCY_PARAGUAY	595
1.0	TWCY_PERU	51
1.0	TWCY_PHILLIPPINES	63
1.0	TWCY_PITCAIRNIS	1030
1.0	TWCY_PNEWGUINEA	675
1.0	TWCY_POLAND	48
1.0	TWCY_PORTUGAL	351
1.0	TWCY_QATAR	974

Version	Constant ID	Numeric ID
1.0	TWCY_REUNIONI	1031
1.0	TWCY_ROMANIA	40
1.0	TWCY_RWANDA	250
1.0	TWCY_SAIPAN	670
1.0	TWCY_SANMARINO	39
1.0	TWCY_SAOTOME	1033
1.0	TWCY_SAUDIARABIA	966
1.0	TWCY_SENEGAL	221
1.0	TWCY_SEYCHELLESIS	1034
1.0	TWCY_SIERRALEONE	1035
1.0	TWCY_SINGAPORE	65
1.0	TWCY_SOLOMONIS	1036
1.0	TWCY_SOMALI	1037
1.0	TWCY_SOUTHAFRICA	27
1.0	TWCY_SPAIN	34
1.0	TWCY_SRILANKA	94
1.0	TWCY_STHELENA	1032
1.0	TWCY_STKITTS	8013
1.0	TWCY_STLUCIA	8014
1.0	TWCY_STPIERRE	508
1.0	TWCY_STVINCENT	8015
1.0	TWCY_SUDAN	1038
1.0	TWCY_SURINAME	597
1.0	TWCY_SWAZILAND	268
1.0	TWCY_SWEDEN	46
1.0	TWCY_SWITZERLAND	41
1.0	TWCY_SYRIA	1039
1.0	TWCY_TAIWAN	886
1.0	TWCY_TANZANIA	255
1.0	TWCY_THAILAND	66
1.0	TWCY_TOBAGO	8016
1.0	TWCY_TOGO	228
1.0	TWCY_TONGAIS	676
1.0	TWCY_TRINIDAD	8016
1.0	TWCY_TUNISIA	216
1.0	TWCY_TURKEY	90
1.0	TWCY_TURKSCAICOS	8017
1.0	TWCY_TUVALU	1040
1.0	TWCY_UGANDA	256
1.0	TWCY_USSR	7
1.0	TWCY_UAEMIRATES	971
1.0	TWCY_UNITEDKINGDOM	44
1.0	TWCY_USA	1
1.0	TWCY_URUGUAY	598
1.0	TWCY_VANUATU	1041
1.0	TWCY_VATICANCITY	39
1.0	TWCY_VENEZUELA	58
1.0	TWCY_WAKE	1042

Version	Constant ID	Numeric ID
1.0	TWCY_WALLISIS	1043
1.0	TWCY_WESTERNSAHARA	1044
1.0	TWCY_WESTERNSAMOA	1045
1.0	TWCY_YEMEN	1046
1.0	TWCY_YUGOSLAVIA	38
1.0	TWCY_ZAIRE	243
1.0	TWCY_ZAMBIA	260
1.0	TWCY_ZIMBABWE	263
1.8	TWCY_ALBANIA	355
1.8	TWCY_ARMENIA	374
1.8	TWCY_AZERBAIJAN	994
1.8	TWCY_BELARUS	375
1.8	TWCY_BOSNIAHERZGO	387
1.8	TWCY_CAMBODIA	855
1.8	TWCY_CROATIA	385
1.8	TWCY_CZECHREPUBLIC	420
1.8	TWCY_DIEGOGARCIA	246
1.8	TWCY_ERITREA	291
1.8	TWCY_ESTONIA	372
1.8	TWCY_GEORGIA	995
1.8	TWCY_LATVIA	371
1.8	TWCY_LESOTHO	266
1.8	TWCY_LITHUANIA	370
1.8	TWCY_MACEDONIA	389
1.8	TWCY_MAYOTTEIS	269
1.8	TWCY_MOLDOVA	373
1.8	TWCY_MYANMAR	95
1.8	TWCY_NORTHKOREA	850
1.8	TWCY_PUERTORICO	787
1.8	TWCY_RUSSIA	7
1.8	TWCY_SERBIA	381
1.8	TWCY_SLOVAKIA	421
1.8	TWCY_SLOVENIA	386
1.8	TWCY_SOUTHKOREA	82
1.8	TWCY_UKRAINE	380
1.8	TWCY_USVIRGINIS	340
1.8	TWCY_VIETNAM	84

Deprecated Items

These items are maintained because after something is added to TWAIN both its name and numeric id (if any) cannot be reused for any other purpose. However, even though they should not be used in applications and drivers, they must be included in a deprecated section in any compliant TWAIN include file. This helps ensure compatibility among all versions of TWAIN drivers and applications, and prevents collisions as new names and numbers are added to the Specification.

At this time the most notable depreciation is the DAT_SETUPFILEXFER2 and all related items. These were added in 1.9 to help with the Macintosh. It was subsequently decided that it wasn't very useful (and no one was using it), so to cut down on confusion it's been removed.

Platform Dependent Definitions and Typedefs

```
#ifdef WIN32
   #define TW_HUGE
#else
   #define TW_HUGE    huge
#endif
typedef BYTE TW_HUGE * HPBYTE;
typedef void TW_HUGE * HPVOID;
```

String types

```
typedef unsigned char    TW_STR1024[1026],  FAR *pTW_STR1026,
                         FAR *pTW_STR1024;
typedef wchar_t          TW_UNI512[512],   FAR *pTW_UNI512;
```

Constants

Note: For a description of these constants see the previous version of the TWAIN specification.

Capability Argument Constants

ACAP_AUDIOFILEFORMAT

Constant ID	Numeric ID
TWAF_WAV	0
TWAF_AIFF	1
TWAF_AU	2
TWAF_SND	3

Section		(defined as)
Data Argument Types (DAT_)	DAT_SETUPFILEXFER2 DAT_TWUNKIDENTITY	0x301 0x000B
ItemTypes for Capability Container structures	TWTY_STR1024 TWTY_UNI512	0x000d 0x000e

Capabilities	CAP_SUPPORTEDCAPSEXT	0x100c	
	CAP_FILESYSTEM	0x????	
	CAP_CLEARBUFFERS	0x101D	/* Added 1.8 */
	CAP_PAGEMULTIPLEACQUIRE	0x1023	/* Added 1.8 */
	CAP_PAPERBINDING	0x102f	/* Added 1.8 */
	CAP_PASSTHRU	0x1031	/* Added 1.8 */
	CAP_POWERDOWNTIME	0x1034	/* Added 1.8, deprecated */ /* 0x1034 has been reused */ /* by CAP_CAMERASIDE */
Messages	MSG_INVOKE_CALLBACK	0x0903	
	MSG_CHECKSTATUS	0x0201	
Query Support	TWQC_CONSTRAINABLE	0x0040	
Capability values	TWSX_FILE2	3	
	TWFS_FILESYSTEM	0	
	TWFS_RECURSIVEDELETE	1	
	TWPT_BGR	12	
	TWPT_CIELAB	13	
	TWPT_CIELUV	14	
	TWPT_YCBCR	15	
	TWSS_B	8	
	TWSS_B5LETTER	TWSS_JISB5	
	TWSS_A4LETTER	TWSS_A4	
	TWSS_B3	TWSS_ISOB3	
	TWSS_B4	TWSS_ISOB4	
	TWSS_B6	TWSS_ISOB6	
	TWLG_DAN	TWLG_DANISH	
	TWLG_DUT	TWLG_DUTCH	
	TWLG_ENG	TWLG_ENGLISH	
	TWLG_USA	TWLG_ENGLISH_USA	
	TWLG_FIN	TWLG_FINNISH	
	TWLG_FRN	TWLG_FRENCH	
	TWLG_FCF	TWLG_FRENCH_CANADIAN	
	TWLG_GER	TWLG_GERMAN	
	TWLG_ICE	TWLG_ICELANDIC	
	TWLG_ITN	TWLG_ITALIAN	
	TWLG_NOR	TWLG_NORWEGIAN	
	TWLG_POR	TWLG_PORTUGUESE	
	TWLG_SPA	TWLG_SPANISH	
	TWLG_SWE	TWLG_SWEDISH	
	TWCB_AUTO	0	
	TWCB_CLEAR	1	
	TWCB_NOCLEAR	2	

Structures

```
typedef struct {
    TW_MEMREF     FileName;
    TW_UINT16     FileNameType;
    TW_UINT16     Format;
    TW_INT16      VRefNum;
    TW_UINT32     parID;
} TW_SETUPFILEXFER2, FAR * pTW_SETUPFILEXFER2;

typedef struct {
    TW_IDENTITY   identity;
    TW_STR255     dsPath;
} TW_TWUNKIDENTITY, FAR * pTW_TWUNKIDENTITY;

typedef struct {
    TW_INT8       destFlag;
    TW_IDENTITY   dest;
    TW_INT32      dataGroup;
    TW_INT16      dataArgType;
    TW_INT16      message;
    TW_INT32      pDataSize;
} TW_TWUNKDSENTRYPARAMS, FAR * pTW_TWUNKDSENTRYPARAMS;

typedef struct {
    TW_UINT16     returnCode;
    TW_UINT16     conditionCode;
    TW_INT32      pDataSize;
} TW_TWUNKDSENTRYRETURN, FAR * pTW_TWUNKDSENTRYRETURN;

typedef struct {
    TW_UINT16     Cap;
    TW_UINT16     Properties;
} TW_CAPEXT, FAR * pTW_CAPEXT;

typedef struct {
    TW_STR255     FileName;
    TW_UINT16     Format;
    TW_INT16      VRefNum;
} TW_SETUPAUDIOFILEXFER, FAR * pTW_SETUPAUDIOFILEXFER;
```

9

Extended Image Information Definitions

Chapter Contents

The following sections contain information about extended image attributes.

TWAIN 1.7 Extended Image Attribute Capabilities

The following extended image attribute capabilities have been defined. If a data source wishes to create additional custom image attribute capabilities, it should define its `TWEI_CUSTOM`*`xxx`* identifiers with a base starting ID of `TWEI_CUSTOM+(`*`x`*`)` where *x* is a unique positive number defined by the data source.

For all extended image attributes see: `DG_IMAGE/DAT_EXTIMAGEINFO/MSG_GET`

Bar Code Recognition

TWEI_BARCODECOUNT

Description:	Returns the number of bar codes found on the document image. A value of 0 means the bar code engine was enabled but that no bar codes were found. A value of -1 means the bar code engine was not enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BARCODECONFIDENCE

Description:	This number reflects the degree of certainty the bar code engine has in the accuracy of the information obtained from the scanned image and ranges from 0 (no confidence) to 100 (supreme confidence). The Source may return a value of -1 if it does not support confidence reporting.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BARCODEROTATION

Description:	The bar code's orientation on the scanned image is described in reference to a Western-style interpretation of the image.	
Value Type:	`TWTY_UINT32`	
Allowed Values:	`TWBCOR_ROT0`	Normal reading orientation
	`TWBCOR_ROT90`	Rotated 90 degrees clockwise
	`TWBCOR_ROT180`	Rotated 180 degrees clockwise
	`TWBCOR_ROT270`	Rotated 270 degrees clockwise
	`TWBCOR_ROTX`	The orientation is not known.

TWEI_BARCODETEXTLENGTH

Description:	The number of ASCII characters derived from the bar code.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BARCODETEXT

Description:	The text of a bar code found on a page.
Value Type:	`TWTY_HANDLE`
Allowed Values:	Any handle to a string

TWEI_BARCODEX

Description:	The X coordinate in pixels of a bar code found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BARCODEY

Description:	The Y coordinate in pixels of a bar code found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BARCODETYPE

Description:	The type of bar code found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	Any of `TWBT_xxxx`

Shaded Area Detection and Removal

TWEI_DESHADECOUNT

Description:	Returns the number of shaded regions found and erased in the document image. A value of 0 means the deshade engine was enabled but that no regions were processed. A value of -1 means the deshade engine was not enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADETOP

Description:	The top coordinate in pixels of a shaded region found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADELEFT

Description:	The left coordinate in pixels of a shaded region found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEHEIGHT

Description:	The height in pixels of a shaded region found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWIDTH

Description:	The width in pixels of a shaded region found on a page.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADESIZE

Description:	The width in pixels of the dots within the shade region.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEBLACKCOUNTOLD

Description:	The total number of black pixels in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEBLACKCOUNTNEW

Description:	The total number of black pixels in the region after deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEBLACKRLMIN

Description:	The shortest black pixel run-length in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEBLACKRLMAX

Description:	The longest black pixel run-length in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWHITECOUNTOLD

Description:	The total number of white pixels in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWHITECOUNTNEW

Description:	The total number of white pixels in the region after deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWHITERLMIN

Description:	The shortest white pixel run-length in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWHITERLAVE

Description:	The average length of all white pixel run-lengths in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_DESHADEWHITERLMAX

Description:	The longest white pixel run-length in the region prior to deshading. If this value is unknown the Source returns -1.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

Speckle Removal

TWEI_SPECKLESREMOVED

Description:	The number of speckles removed from the image when de-speckle is enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_BLACKSPECKLESREMOVED

Description:	The number of black speckles removed from the image when despeckle is enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_WHITESPECKLESREMOVED

Description:	The number of white speckles removed (black speckles added) from the image when despeckle is enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

Horizontal Line Detection and Removal

TWEI_HORZLINECOUNT

Description:	Returns the number of horizontal lines found and erased in the document image. A value of 0 means the line removal engine was enabled but that no lines were found. A value of -1 means the line engine was not enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_HORZLINEXCOORD

Description:	The X coordinate in pixels of a horizontal line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_HORZLINEYCOORD

Description:	The Y coordinate in pixels of a horizontal line detected in the image
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_HORZLINELENGTH

Description:	The length in pixels of a horizontal line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_HORZLINETHICKNESS

Description:	The thickness (height) in pixels of a horizontal line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

Vertical Line Detection and Removal

TWEI_VERTLINECOUNT

Description:	Returns the number of vertical lines found and erased in the document image. A value of 0 means the line removal engine was enabled but that no lines were found. A value of -1 means the line engine was not enabled.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_VERTLINEXCOORD

Description:	The X coordinate in pixels of a vertical line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_VERTLINEYCOORD

Description:	The Y coordinate in pixels of a vertical line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_VERTLINELENGTH

Description:	The length in pixels of a vertical line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_VERTLINETHICKNESS

Description:	The thickness (width) in pixels of a vertical line detected in the image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

Patch Code Detection (Job Separation)

TWEI_PATCHCODE

Description:	The patch code detected.
Value Type:	`TWTY_UINT32`
Allowed Values:	`TWPCH_PATCH1, TWPCH_PATCH2, TWPCH_PATCH3, TWPCH_PATCH4, TWP= CH_PATCH6, TWPCH_PATCHT`

Skew detection and Removal

TWEI_DESKEWSTATUS

Description:	Returns the status of the deskew operation.	
Value Type:	`TWTY_UINT32`	
Allowed Values:	`TWDSK_SUCCESS`	Image successfully deskewed
	`TWDSK_REPORTONLY`	Deskew information only
	`TWDSK_FAIL`	Deskew failed
	`TWDSK_DISABLED`	Deskew engine not enabled

TWEI_SKEWORIGINALANGLE

Description:	The amount of skew in the original image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWFINALANGLE

Description:	The amount of skew in the deskewed image. This number may not be zero.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWCONFIDENCE

Description:	This number reflects the degree of certainty the deskew engine has in the accuracy of the deskewing of the current image and ranges from 0 (no confidence) to 100 (supreme confidence). The Source may return a value of -1 if it does not support confidence reporting.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWX1

Description:	This is the X image coordinate in pixels of the upper left corner of the virtual deskewed image. It may be negative indicating the deskewed corner is not represented by actual pixels.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWY1

Description:	The Y image coordinate in pixels of the upper left corner of the virtual deskewed image. It may be negative indicating the deskewed corner is not represented by actual pixels.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWX2

Description:	The X image coordinate in pixels of the upper right corner of the virtual deskewed image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWY2

Description:	The Y image coordinate in pixels of the upper right corner of the virtual deskewed image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWX3

Description:	This is the X image coordinate in pixels of the lower left corner of the virtual deskewed image. It may be negative indicating the deskewed corner is not represented by actual pixels.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWY3

Description:	The Y image coordinate in pixels of the lower left corner of the virtual deskewed image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWX4

Description:	The X image coordinate in pixels of the lower right corner of the virtual deskewed image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_SKEWWINDOWY4

Description:	The Y image coordinate in pixels of the lower right corner of the deskewed image.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

Endorsed / Imprinted Text

TWEI_ENDORSEDTEXT

Description:	The text that was endorsed on the paper by the scanner.
Value Type:	`TWTY_STR255`
Allowed Values:	Any string

Forms Recognition

TWEI_FORMCONFIDENCE

Description:	The confidence that the specified form was detected. This is an array property with a confidence factor for each form In the data set with 0 meaning no match and 100 meaning absolute certainty. Typically values over 70 imply a good form match with the template.
Value Type:	`TWTY_UINT32`
Allowed Values:	0 to 100

TWEI_FORMTEMPLATEMATCH

Description:	The array of file names for the master forms matched against a form. If multi-page master forms are used, the associated page numbers are contained in the `FORMTEMPLATEPAGEMATCH` capability array.
Value Type:	`TWTY_STR255`
Allowed Values:	Any string

TWEI_FORMTEMPLATEPAGEMATCH

Description:	An array containing the number of the page from a multi-page master form matched against a form image. It is useful when matching a form image against the pages of a multi-page master form. The file name of the master form is contained in the `FORMTEMPLATEMATCH` capability.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_FORMHORZDOCOFFSET

Description:	An array containing the perceived horizontal offsets in pixels of the form image being matched against a set of master forms. This is useful for page registration once the form has been recognized.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_FORMVERTDOCOFFSET

Description:	An array containing the perceived vertical offsets in pixels of the form image being matched against a set of master forms. This is useful for page registration once the form has been recognized.
Value Type:	`TWTY_UINT32`
Allowed Values:	>= 0

TWAIN 1.9 Extended Image Attribute Capabilities

These next items, taken together, provide a way to unambiguously identify the physical source of an image. Applications can use this information to associate scanned images from the same side of a sheet of paper, the sheet of paper itself, or a set of sheets comprising a document. While much of this information is available using `DAT_IMAGELAYOUT`, it is provided here for performance reasons; to allow an Application to glean as much information about the image as possible through a single call.

These items are mandatory, if a Source supports `DAT_EXTIMAGEINFO`, then these items must be present in the data returned by the Source.

TWEI_BOOKNAME

Description:	This is new with TWAIN 1.9, expanding on the document/page/frame numbers described by previous versions of TWAIN in the `TW_IMAGELAYOUT` structure. The ordering is book/chapter/ document/page(camera)/frame, and increases the detail of image addressing that a Source can provide for an Application. TWAIN 1.9 Sources that support extended image info must provide this information, even if the value is always fixed at 1.
Value Type:	`TWTY_STR255`
Allowed Values:	Any valid string data.
See Also:	`DAT_IMAGELAYOUT` `TW_IMAGELAYOUT`

TWEI_CHAPTERNUMBER

Description:	This is new with TWAIN 1.9, expanding on the document/page/frame numbers described by previous versions of TWAIN in the `TW_IMAGELAYOUT` structure. The ordering is book/chapter/ document/page(camera)/frame, and increases the detail of image addressing that a Source can provide for an Application. TWAIN 1.9 Sources that support extended image info must provide this information, even if the value is always fixed at 1.
Value Type:	`TWTY_UINT32`
Allowed Values:	1 to 2^{32}-1
See Also:	`DAT_IMAGELAYOUT` `TW_IMAGELAYOUT`

TWEI_DOCUMENTNUMBER

Description:	This must be the same value returned by a call to `DG_CONTROL` / `DAT_IMAGELAYOUT` / `MSG_GET`. The ordering is book/chapter/document/page(camera)/frame, and increases the amount of image addressing that a Source can provide for an Application. TWAIN 1.9 Sources that support extended image info must provide this information, even if the value is always fixed at 1.
Value Type:	`TWTY_UINT32`
Allowed Values:	1 to 2^{32}-1
See Also:	`DAT_IMAGELAYOUT` `TW_IMAGELAYOUT`

TWEI_PAGENUMBER

Description:	This must be the same value returned by a call to `DG_CONTROL` / `DAT_IMAGELAYOUT` / `MSG_GET`. The ordering is book/chapter/document/page(camera)/frame, and increases the amount of image addressing that a Source can provide for an Application. TWAIN 1.9 Sources that support extended image info must provide this information, even if the value is always fixed at 1.
Value Type:	`TWTY_UINT32`
Allowed Values:	1 to 2^{32}-1
See Also:	`DAT_IMAGELAYOUT` `TW_IMAGELAYOUT`

TWEI_CAMERA

Description: The primary use of this value is to determine if the image is from the top or the bottom side of a sheet of paper. This is accomplished by naming the camera that was used to obtain the image. For Sources that support DAT_FILESYSTEM, the Application can use the string to determine if the camera is capturing images from the top or bottom side of the paper.

Applications should browse the available camera devices in State 4 to create a lookup table mapping the various camera filenames to the side they represent. DAT_FILESYSTEM is not supported, then the Application should watch for the strings “TOP” and “BOTTOM”.

Value Type: TWTY_STR255

Allowed Values: TWFY_CAMERA, TWFY_CAMERATOP and TWFY_CAMERABOTTOM filenames maintained by the Source and accessible using the DAT_FILESYSTEM triplet. This string must be exactly the same as that supplied by the Source when the Application issues a DAT_FILESYSTEM / MSG_GETINFO (or related command).

The Source must identify the exact camera used. This means that even if the Source has been set to use a TWFY_CAMERA device, it must report the TWFY_CAMERATOP or TWFY_CAMERABOTTOM device as appropriate.

If the Source does not support the use of DAT_FILESYSTEM, then it must return the string “TOP” for images from the top side of the sheet of paper, and “BOTTOM” for images on the bottom side of the sheet of paper.

TWEI_FRAMENUMBER

Description: This must be the same value returned by a call to DG_CONTROL / DAT_IMAGELAYOUT / MSG_GET. TWAIN 1.9 Sources that support extended image info must provide this information.

Value Type: TWTY_UINT32

Allowed Values: 1 to 2^{32}-1

See Also: DAT_IMAGELAYOUT
TW_IMAGELAYOUT

TWEI_FRAME

Description:	Returns the coordinates of the current segment within the current document in pixels. Since segments may be acquired at various resolutions, the physical location must be calculated using the resolution reported by `DAT_IMAGEINFO` for the current segment. The top left coordinate in pixels stored in the FRAME is relative to the top left of the scanned document.
Value Type:	`TWTY_FRAME`
Allowed Values:	Any valid FRAME describing a segment within the boundaries of the current document

See Also:	`TWEI_SEGMENTNUMBER`
	`DAT_IMAGELAYOUT`
	`TW_IMAGELAYOUT`
	`ICAP_FRAMES`

TWEI_PIXELFLAVOR

Description:	This value must correctly describe the pixel flavor of the current image, the same data that is available through `ICAP_PIXELFLAVOR`. TWAIN 1.9 Sources that support extended image info must provide this information.
Value Type:	`TWTY_UINT16`
Allowed Values:	`TWPF_CHOCOLATE` `TWPF_VANILLA`
See Also:	`ICAP_PIXELFLAVOR`

TWAIN 1.91 Extended Image Attribute Capabilities

These next items add image segmentation and ICC Profile metadata returns.

TWEI_ICCPROFILE

Description:	Returns the name of the ICC profile that was used to render the current image. This may be a fully qualified path indicating the exact location of the ICC profile. If this is not a fully qualified path, then the default location is operating system dependant. Windows default location: <windows path>\system32\spool
Value Type:	`TWTY_STR255`
Allowed Values:	Any valid ICC profile file name or fully qualified path

TWEI_LASTSEGMENT

Description:	Returns `TRUE` if the current segment is the last segment of a page.
Value Type:	`TWTY_BOOL`
Allowed Values:	`TRUE` or `FALSE`
See Also:	

TWEI_SEGMENTNUMBER

Description:	Returns a number identifying the segment of an image. Segments allow independent image processing strategies on a document for more accurate document reproduction and smaller file sizes. For instance, a document containing text with a picture may be segmented into a high resolution bitonal image consisting of the text and a lower resolution color image consisting of the picture. Sources that support this item must support `TWEI_FRAME`, and must specify a left and top value to position the segment in the final image. Segments may overlap.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

See Also:	`TWEI_FRAME`
	`DAT_IMAGELAYOUT`
	`TW_IMAGELAYOUT`
	`ICAP_FRAMES`

TWAIN 2.0 Extended Image Attribute Capabilities

These next items provide support for MICR.

TWEI_MAGTYPE

Description:	Reports back that magnetic data was found and it is of a string. Data may contain placeholders for unrecognized characters.
Value Type:	`TWTY_UINT16`
Allowed Values:	`TWMD_MICR`, `TWMD_RAW`, or `TWMD_INVALID`

TWAIN 2.1 Extended Image Attribute Capabilities

TWEI_FILESYSTEMSOURCE

Description:	Returns a `DAT_FILESYSTEM` string describing the camera that captured the image data.
Value Type:	`TWTY_STR255`
Allowed Values:	Any of the camera values returned by `DAT_FILESYSTEM` / `MSG_GETFIRSTFILE` or `MSG_GETNEXTFILE`

TWEI_IMAGEMERGED

Description:	Indicates that the current image is the result of a merger between the front and rear images of a duplex capture. See `ICAP_IMAGEMERGE` for more information.
Value Type:	`TWTY_BOOL`
Allowed Values:	TRUE if the front and rear images were merged.

TWEI_MAGDATA

Description:	This is a "blob" of data with a byte count retrieved from the driver/ device. The interpretation of the data comes from `TWEI_MAGTYPE`.
Value Type:	`TWTY_HANDLE`, or `TWTY_STR255`
Allowed Values:	Any handle to a blob of data

TWEI_MAGDATALENGTH

Description:	This describes the length of the magnetic data. Either in bytes for "blob" or data or characters for string data.
Value Type:	`TWTY_UINT32`
Allowed Values:	>=0

TWEI_PAGESIDE

Description:	Returns a value indicating if the image represents the front or rear of the sheet of paper.
Value Type:	`TWTY_UINT16`
Allowed Values:	`TWCS_TOP` (front of sheet) `TWCS_BOTTOM` (rear of sheet)

TWAIN 2.2 Extended Image Attribute Capabilities

TWEI_PAPERCOUNT

Description:	This is the number of sheets of paper passed through the ADF for the current batch. TWAIN 2.2 Sources that support extended image info must provide this information.
Value Type:	`TWTY_UINT32`
Allowed Values:	1 to $2^{32}-1$
See Also:	`TWEI_PAGENUMBER`

TWAIN 2.3 Extended Image Attribute Capabilities

TWEI_PRINTERTEXT

Description:	The text printed on the document; use `TW_INFO[].NumItems` to specify the number of lines.
Value Type:	`TWTY_STR255`
Allowed Values:	Any string

TWAIN 2.4 Extended Image Attribute Capabilities

TWEI_TWAINDIRECTMETADATA

Description:	Metadata describing the image.
Value Type:	`TWTY_HANDLE`
Allowed Values:	The metadata for the current image, encoded as a NUL-terminated UTF-8 C style string, in JSON format. For information about the JSON format, refer to the TWAIN Direct Specification at http://www.twaindirect.org

10

Capabilities

Chapter Contents

Overview

Sources *may* support a large number of capabilities but are *required* to support very few. To determine if a capability is supported by a Source, the application can query the Source using a `DG_CONTROL / DAT_CAPABILITY / MSG_GET, MSG_GETCURRENT`, or `MSG_GETDEFAULT` operation. The application specifies the particular capability by storing its identifier in the Cap field of the `TW_CAPABILITY` structure. This is the structure pointed to by the `pData` parameter in the `DSM_Entry( )` call.

`DG_CONTROL / DAT_CAPABILITY` operations for capability negotiation include:

`MSG_GET`	Returns the Current, Default and Available settings for a capability.
`MSG_GETCURRENT`	Returns the Current setting for a capability.
`MSG_GETDEFAULT`	Returns the value of the Source's preferred Default values.
`MSG_RESET`	Returns the capability to its TWAIN Default (power-on) condition (i.e. all previous negotiation is ignored).
`MSG_RESETALL`	Returns all of the current values to the default settings used when the driver was first installed.
`MSG_SET`	Allows the application to set the Current value of a capability.
`MSG_SETCONSTRAINT`	Allows the application to set the Current and Default value(s) and restrict the Available values to some subset of the Source's power-on set of values. Sources are strongly encouraged to allow the application to set as many of its capabilities as possible, and further to reflect these changes in the Source's user interface. This will ensure that the user can only select images with characteristics that are useful to the consuming application.

Best Practices

The content in the next two sections applies to all TWAIN Capabilities. It's been centralized in this spot to make it easier to find, and to guarantee that it's consistent across the entire TWAIN Specification.

Best Practices for Applications

- Use `DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT` whenever possible; only use `DG_CONTROL / DAT_CAPABILITY / MSG_GET` when there is a need to understand a capability's possible values, or prior to setting constraints.
- If the value reported by `DG_CONTROL / DAT_CAPABILITY / MSG_GETCURRENT` is the desired value, avoid doing a `MSG_SET` on it.
- Use a `MSG_GETCURRENT / MSG_SET` combination whenever possible. If `MSG_GETCURRENT` returns `TW_ONEVALUE` container that has been generated by the data source, it can be examined, updated with the new value, and then sent back to the data source with `MSG_SET`. This reduces the likelihood of mismatches in container and/or data types. This can't be done with `TW_ARRAY` containers, because the new array may be larger than the old array, and the data source may not have allocated enough memory to re-use the container.
- There is no good reason to ever use `MSG_GETDEFAULT`.
- Accept `TWRC_CHECKSTATUS` as an indication of a successful `DG_CONTROL / DAT_CAPABILITY / MSG_*` operation, unless after further examination the current contents of the capability cannot be supported by the application.
- Check that `CAP_XFERCOUNT` is set to -1 for batch scans, values of 1 (single image) or 2 (front and rear) may be supported.
- Set `ICAP_XFERMECH` to indicate the `DG_IMAGE / DAT_IMAGE*XFER / MSG_GET` operation that will be used.
- Set `ICAP_UNITS` to `TWUN_INCHES`, unless there is a need to show a different unit on the driver's GUI, or have different units reported by `TW_IMAGEINFO` and `TW_EXTIMAGEINFO` (typically for the resolution).
- Check the value of `ICAP_COMPRESSION`. If possible, constrain it to values supported by the application. If this cannot be done, or if `DG_IMAGE / DAT_IMAGEINFO / MSG_GET` reports an unsupported `TW_IMAGEINFO.Compression` value in State 7 after receiving `TWRC_XFERDONE`, then gracefully abort the image capture session using `DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET`, if `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER` returns a non-zero value for `TW_PENDINGXFERS.Count`.

Best Practices for Data Sources

- If the requested capability isn't supported, return `TWRC_FAILURE / TWCC_CAPUNSUPPORTED`.
- If the capability is supported but the requested MSG_ operation isn't, return `TWRC_FAILURE / TWCC_CAPBADOPERATION`.
- If the capability cannot be accessed due to the setting of a related capability, return `TWRC_FAILURE / TWCC_CAPSEQERROR`.
- If a capability is supported, but if accessed would return `TWCC_CAPSEQERROR`, then a call to `DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT` returns a value of 0.

- Calls to DG_CONTROL / DAT_CAPABILITY / MSG_QUERYSUPPORT for any capability never fail, they return 0 if the capability is unsupported or inaccessible.
- Read only capabilities must support MSG_GET, MSG_GETCURRENT, MSG_GETDEFAULT and MSG_QUERYSUPPORT. The three get operations must return the same value. TW_BOOL capabilities are an exception. Container type for MSG_GET is TW_ENUMERATION.
- Writeable capabilities must support MSG_GET, MSG_GETCURRENT, MSG_GETDEFAULT, MSG_SET, MSG_RESET and MSG_QUERYSUPPORT, even if only one array value, one enumeration or one value in a range is supported.
- If the value in a MSG_SET cannot be exactly matched, but there is a reasonable alternative (usually near misses with TW_RANGE capabilities or TW_FIX32 data types), then set the alternate value and return TWRC_CHECKSTATUS.

Required Capabilities

The list of required capabilities can be found in Chapter 5, "Source Implementation".

Sources must implement and make available to TWAIN applications the advertised features of the devices they support. This is especially true in “no-UI mode.” A Source must support a capability if its device supports it, even if the capability is listed as required by none.

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).

Audible Alarms

CAP_ALARMS	Turns specific audible alarms on and off.
CAP_ALARMVOLUME	Controls the volume of a device’s audible alarm.

Audio

ACAP_XFERMECH	Allows application and source to identify which audio transfer mechanisms they have in common.

Automatic Adjustments

CAP_AUTOMATICSENSEMEDIUM	Configures a Source to check for paper in the Automatic Document Feeder.
ICAP_AUTODISCARDBLANKPAGES	Discards blank pages.
ICAP_AUTOMATICBORDERDETECTION	Turns automatic border detection on and off.
ICAP_AUTOMATICCOLORENABLED	Detects the pixel type of the image and returns either a color image or a non-color image specified by ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE.
ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE	Specifies the non-color pixel type to use when automatic color is enabled.
ICAP_AUTOMATICCROPUSESFRAME	Reduces the amount of data captured from the device, potentially improving the performance of the driver.
ICAP_AUTOMATICDESKEW	Turns automatic skew correction on and off.
ICAP_AUTOMATICLENGTHDETECTION	Controls the automatic detection of the length of a document, this is intended for use with an Automatic Document Feeder.
ICAP_AUTOMATICROTATE	When TRUE, depends on source to automatically rotate the image.
ICAP_AUTOSIZE	Force the output image dimensions to match either the current value of ICAP_SUPPORTEDSIZES or any of its current allowed values.
ICAP_FLIPROTATION	Orients images that flip orientation every other image.
ICAP_IMAGEMERGE	Merges the front and rear image of a document in one of four orientations: front on the top.
ICAP_IMAGEMERGEHEIGHTTHRESHOLD	Specifies a Y-Offset in ICAP_UNITS units.

Automatic Capture

CAP_AUTOMATICCAPTURE	Specifies the number of images to automatically capture.
CAP_TIMEBEFOREFIRSTCAPTURE	Selects the number of seconds before the first picture taken.
CAP_TIMEBETWEENCAPTURES	Selects the hundredths of a second to wait between pictures taken.

Automatic Scanning

CAP_AUTOSCAN	Enables the source's automatic document scanning process.
CAP_CAMERAENABLED	Delivers images from the current camera.

CAP_CAMERAORDER	Selects the order of output for Single Document Multiple Image mode.
CAP_CAMERASIDE	Sets the top and bottom values of cameras in a scanning device.
CAP_MAXBATCHBUFFERS	Describes the number of pages that the scanner can buffer when CAP_AUTOSCAN is enabled.

Bar Code Detection Search Parameters

ICAP_BARCODEDETECTIONENABLED	Turns bar code detection on and off.
ICAP_SUPPORTEDBARCODETYPES	Provides a list of bar code types that can be detected by current data source.
ICAP_BARCODEMAXRETRIES	Restricts the number of times a search will be retried if no bar codes are found.
ICAP_BARCODEMAXSEARCHPRIORITIES	Specifies the maximum number of supported search priorities.
ICAP_BARCODESEARCHMODE	Restricts bar code searching to certain orientations, or prioritizes one orientation over another.
ICAP_BARCODESEARCHPRIORITIES	A prioritized list of bar code types dictating the order in which they will be sought.
ICAP_BARCODETIMEOUT	Restricts the total time spent on searching for bar codes on a page.

Capability Negotiation Parameters

CAP_EXTENDEDCAPS	Capabilities negotiated in States 5, 6 and 7
CAP_SUPPORTEDCAPS	Inquire Source's capabilities valid for MSG_GET
CAP_SUPPORTEDDATS	Inquire Source's DAT messages that are valid.

Color

ICAP_COLORMANAGEMENTENABLED	Disables the Source's color and gamma tables for color and grayscale images, resulting in output that that could be termed "raw".
ICAP_FILTER	Color characteristics of the subtractive filter applied to the image data
ICAP_GAMMA	Gamma correction value for the image data
ICAP_ICCPROFILE	Embeds or links ICC profiles into files
ICAP_PLANARCHUNKY	Color data format - Planar or Chunky

Compression

ICAP_BITORDERCODES	CCITT Compression
ICAP_CCITTKFACTOR	CCITT Compression
ICAP_COMPRESSION	Compression method for Buffered Memory Transfers
ICAP_JPEGPIXELTYPE	JPEG Compression
ICAP_JPEGQUALITY	JPEG quality
ICAP_JPEGSUBSAMPLING	JPEG subsampling
ICAP_PIXELFLAVORCODES	CCITT Compression
ICAP_TIMEFILL	CCITT Compression

Device Parameters

CAP_DEVICEONLINE	Determines if hardware is on and ready
CAP_DEVICETIMEDATE	Date and time of a device's clock.
CAP_SERIALNUMBER	The serial number of the currently selected source device.
ICAP_MINIMUMHEIGHT	Allows the source to define the minimum height (Y-axis) that the source can acquire.
ICAP_MINIMUMWIDTH	Allows the source to define the minimum width (X-axis) that the source can acquire.
ICAP_EXPOSURETIME	Exposure time used to capture the image, in seconds
ICAP_FLASHUSED2	For devices that support a flash, MSG_SET selects the flash to be used; MSG_GET reports the current setting.
ICAP_IMAGEFILTER	For devices that support image filtering, selects the algorithm to be used.
ICAP_LAMPSTATE	Is the lamp on?
ICAP_LIGHTPATH	Image was captured transmissively or reflectively
ICAP_LIGHTSOURCE	Describes the color characteristic of the light source used to acquire the image
ICAP_NOISEFILTER	For devices that support noise filtering, selects the algorithm to be used.
ICAP_OVERSCAN	For devices that support overscanning, controls whether additional rows or columns are appended to the image.
ICAP_PHYSICALHEIGHT	Maximum height Source can acquire (in ICAP_UNITS)
ICAP_PHYSICALWIDTH	Maximum width Source can acquire (in ICAP_UNITS)
ICAP_UNITS	Unit of measure (inches, centimeters, etc.)
ICAP_ZOOMFACTOR	With MSG_GET, returns all camera supported lens zooming range.

Doublefeed Detection

CAP_DOUBLEFEEDDETECTION	Control DFD functionality
CAP_DOUBLEFEEDDETECTIONLENGTH	Set the minimum length
CAP_DOUBLEFEEDDETECTIONSENSITIVITY	Set detector sensitivity
CAP_DOUBLEFEEDDETECTIONRESPONSE	Describe Source behavior in case of DFD

Imprinter/Endorser Functionality

CAP_ENDORSER	Allows the application to specify the starting endorser / imprinter number.
CAP_PRINTER	MSG_GET returns current list of available printer devices; MSG_SET selects the device for negotiation.
CAP_PRINTERCHARROTATION	Orientation of each character in the font
CAP_PRINTERENABLED	Turns the current CAP_PRINTER device on or off.
CAP_PRINTERFONTSTYLE	Style (ex: bold, italic…)
CAP_PRINTERINDEX	Starting number for the CAP_PRINTER device.
CAP_PRINTERINDEXLEADCHAR	Lead character for padding
CAP_PRINTERINDEXMAXVALUE	Maximum allowed printer index value
CAP_PRINTERINDEXNUMDIGITS	Maximum allowed digits in printer index value
CAP_PRINTERINDEXSTEP	Increment between printer index values
CAP_PRINTERINDEXTRIGGER	Triggers for incrementing the printer index value
CAP_PRINTERMODE	Specifies appropriate current CAP_PRINTER device mode.
CAP_PRINTERSTRING	String(s) to be used in the string component when CAP_PRINTER device is enabled.
CAP_PRINTERSTRINGPREVIEW	Preview of what a printer string will look like
CAP_PRINTERSUFFIX	String to be used as current CAP_PRINTER device's suffix.
CAP_PRINTERVERTICALOFFSET	Y-Offset for current CAP_PRINTER device

Image Information

CAP_AUTHOR	Author of acquired image (may include a copyright string)
CAP_CAPTION	General note about acquired image
CAP_TIMEDATE	Date and Time the image was acquired (entered State 7)
ICAP_EXTIMAGEINFO	Allows the application to query the data source to see if it supports the new operation triplet DG_IMAGE/ DAT_EXTIMAGEINFO/ MSG_GET.

ICAP_SUPPORTEDEXTIMAGEINFO	Lists all of the information that the Source is capable of returning from a call to DAT_EXTIMAGEINFO.

Image Parameters for Acquire

CAP_THUMBNAILSENABLED	Allows an application to request the delivery of thumbnail representations for the set of images that are to be delivered.
ICAP_AUTOBRIGHT	Enable Source's Auto-brightness function
ICAP_BRIGHTNESS	Source brightness values
ICAP_CONTRAST	Source contrast values
ICAP_HIGHLIGHT	Lightest highlight, values lighter than this value will be set to this value
ICAP_IMAGEDATASET	Gets or sets the image indices that will be delivered during the standard image transfer done in States 6 and 7.
ICAP_MIRROR	Source can, or should, mirror image.
ICAP_ORIENTATION	Defines which edge of the paper is the top: Portrait or Landscape
ICAP_ROTATION	Source can, or should, rotate image this number of degrees
ICAP_SHADOW	Darkest shadow, values darker than this value will be set to this value
ICAP_XSCALING	Source Scaling value (1.0 = 100%) for x-axis
ICAP_YSCALING	Source Scaling value (1.0 = 100%) for y-axis

Image Type

ICAP_BITDEPTH	Pixel bit depth for Current value of ICAP_PIXELTYPE
ICAP_BITDEPTHREDUCTION	Allows a choice of the reduction method for bit depth loss
ICAP_BITORDER	Specifies how the bytes in an image are filled by the Source
ICAP_CUSTHALFTONE	Square-cell halftone (dithering) matrix to be used
ICAP_HALFTONES	Source halftone patterns
ICAP_PIXELFLAVOR	Sense of the pixel whose numeric value is zero
ICAP_PIXELTYPE	The type of pixel data (B/W, gray, color, etc.)
ICAP_THRESHOLD	Specifies the dividing line between black and white values

Language Support

CAP_LANGUAGE	Allows application and source to identify which languages they have in common.

MICR

CAP_MICRENABLED	Enables actions needed to support check scanning.

Pages

CAP_SEGMENTED	Describes the segmentation setting for captured images
ICAP_FRAMES	Size and location of frames on page
ICAP_MAXFRAMES	Maximum number of frames possible per page
ICAP_SUPPORTEDSIZES	Fixed frame sizes for typical page sizes

Paper Handling

CAP_AUTOFEED	MSG_SET to TRUE to enable Source's automatic feeding
CAP_CLEARPAGE	MSG_SET to TRUE to eject current page and leave acquire area empty
CAP_DUPLEX	Indicates whether the scanner supports duplex.
CAP_DUPLEXENABLED	Enables the user to set the duplex option to be TRUE or FALSE.
CAP_FEEDERALIGNMENT	Indicates the alignment of the document feeder.
CAP_FEEDERENABLED	If TRUE, Source's feeder is available
CAP_FEEDERLOADED	If TRUE, Source has documents loaded in feeder (MSG_GET only)
CAP_FEEDERORDER	Specifies whether feeder starts with top of first or last page.
CAP_FEEDERPOCKET	Report what pockets are available as paper leaves a device.
CAP_FEEDERPREP	Improve the movement of paper through a scanner ADF.
CAP_FEEDPAGE	MSG_SET to TRUE to eject current page and feed next page
CAP_PAPERDETECTABLE	Determines whether source can detect documents on the ADF or flatbed.
CAP_PAPERHANDLING	Control paper handling
CAP_REACQUIREALLOWED	Capable of acquring muliple images of the same page wihtout changing the physical registraion of that page.
CAP_REWINDPAGE	MSG_SET to TRUE to do a reverse feed
ICAP_FEEDERTYPE	Allows application to set scan parameters depending on the type of feeder being used.

Patch Code Detection

ICAP_PATCHCODEDETECTIONENABLED	Turns patch code detection on and off.
ICAP_SUPPORTEDPATCHCODETYPES	List of patch code types that can be detected by current data source.
ICAP_PATCHCODEMAXSEARCHPRIORITIES	Maximum number of search priorities.
ICAP_PATCHCODESEARCHPRIORITIES	List of patch code types dictating the order in which patch codes will be sought.
ICAP_PATCHCODESEARCHMODE	Restricts patch code searching to certain orientations, or prioritizes one orientation over another.
ICAP_PATCHCODEMAXRETRIES	Restricts the number of times a search will be retried if none are found on a page.
ICAP_PATCHCODETIMEOUT	Restricts total time for searching for a patch code on a page.

Power Monitoring

CAP_BATTERYMINUTES	The minutes of battery power remaining on a device.
CAP_BATTERYPERCENTAGE	With MSG_GET, indicates battery power status.
CAP_POWERSAVETIME	With MSG_SET, sets the camera power down timer in seconds; with MSG_GET, returns the current setting of the power down time.
CAP_POWERSUPPLY	MSG_GET reports the kinds of power available; MSG_GETCURRENT reports the current power supply to use.

Resolution

ICAP_XNATIVERESOLUTION	Native optical resolution of device for x-axis
ICAP_XRESOLUTION	Current/Available optical resolutions for x-axis
ICAP_YNATIVERESOLUTION	Native optical resolution of device for y-axis
ICAP_YRESOLUTION	Current/Available optical resolutions for y-axis

Transfers

CAP_JOBCONTROL	Allows multiple jobs in batch mode.
CAP_SHEETCOUNT	Number of sheets the application is willing to accept this session (valid when CAP_XFERCOUNT is -1)
CAP_XFERCOUNT	Number of images the application is willing to accept this session
ICAP_COMPRESSION	Buffered Memory transfer compression schemes

ICAP_IMAGEFILEFORMAT	File formats for file transfers
ICAP_TILES	Tiled image data
ICAP_UNDEFINEDIMAGESIZE	The application will accept undefined image size
ICAP_XFERMECH	Transfer mechanism - used to learn options and set-up for upcoming transfer

User Interface

CAP_CAMERAPREVIEWUI	Queries the source for UI support for preview mode.
CAP_CUSTOMDSDATA	Allows the application to query the data source to see if it supports the new operation triplets DG_CONTROL/ DAT_CUSTOMDSDATA / MSG_GET and DG_CONTROL/ DAT_CUSTOMDSDATA / MSG_SET.
CAP_CUSTOMINTERFACEGUID	Uniquely identifies an interface for a Data Source.
CAP_ENABLEDSUIONLY	Queries an application to see if it implements the new user interface settings dialog.
CAP_INDICATORS	Use the Source's progress indicator? (valid only when ShowUI==FALSE)
CAP_INDICATORSMODE	List of messages types that can be display if ICAP_INDICATORS is TRUE
CAP_UICONTROLLABLE	Indicates that Source supports acquisitions with UI disabled

The Capability Listings

The following section lists descriptions of all TWAIN capabilities in alphabetical order. The format of each capability entry is:

NAME OF CAPABILITY

Description

Description of the capability

Application

(Optional) Information for the application

Source

(Optional) Information for the Source

Values

Type:	Data structure for the capability.
Value after MSG_OPENDS:	Indicates the value that the data source has for this capability immediately after being opened.
After MSG_RESET/MSG_RESETALL:	Indicates the value that the data source has for this capability after being reset.
Allowed Values:	Definition of the values allowed for this capability.

Containers

MSG_GET	Acceptable containers for use on `MSG_GET` operations.
MSG_GETCURRENT	Acceptable containers for use on `MSG_GETCURRENT` operations.
MSG_GETDEFAULT	Acceptable containers for use on `MSG_GETDEFAULT` operations.
MSG_SET	Acceptable containers for use on `MSG_SET` operations.
MSG_RESET	Acceptable containers for use on `MSG_RESET` operations, and remove any constraints.
MSG_SETCONSTRAINT	Acceptable containers for use on `MSG_SETCONSTRAINT` operations.

Required By

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

TWAIN Version Introduced

Version x.x

See Also

Associated capabilities and data structures.

ACAP_XFERMECH

Description

Allows the Application and Source to identify which audio transfer mechanisms they have in common.

Application

The current setting of `ACAP_XFERMECH` must match the constant used by the application to specify the audio transfer mechanism when starting the transfer using the triplet: `DG_AUDIO` / `DAT_AUDIOxxxxXFER` / `MSG_GET`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	`TWSX_NATIVE`
After MSG_RESET/MSG_RESETALL:	`TWSX_NATIVE`
Allowed Values:	`TWSX_NATIVE` `TWSX_FILE`

Containers

MSG_GET	`TW_ENUMERATION` `TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Audio Sources

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`DG_AUDIO / DAT_AUDIOFILEXFER / MSG_GET`

CAP_ALARMS

Description

Turns specific audible alarms on and off.

Application

Note that an application may opt to turn off all alarms by issuing a MSG_SET with no data. Therefore, an application should also be prepared to receive an empty array from a Source with an MSG_GET. (i.e., pTW_ARRAY->NumItems == 0)

The easiest way to test for allowed values is to try to set them all with MSG_SET. If not all are allowed, the Source will return TWRC_CHECKSTATUS with those values that it supports.

Source

It is worth noting that the alarms do not have to be present in the device for a Source to make use of this capability. If the device is capable of alerting the Source to these various kinds of conditions, but is unable to generate the alarms, itself; then the Source may opt to generate them on its behalf.

TWAL_ALARM is a catchall for alarms not explicitly listed. It is also used where a device only provides control over a single, multi-use alarm. For instance, if a device beeps for both jams and bar-codes, but doesn't allow independent control of the alarms, then it should report TWAL_ALARM to cover them, and not TWAL_BARCODE, TWAL_JAM.

TWAL_FEEDERERROR covers paper handling errors such as jams, double-feeds, skewing and the like; conditions that most likely stop scanning.

TWAL_FEEDERWARNING covers non-fatal events, such as feeder empty.

TWAL_DOUBLEFEED, TWAL_JAM and TWALSKEW cover paper handling errors.

TWAL_BARCODE and TWAL_PATCHCODE generate alarms when an image with this kind of data is recognized.

TWAL_POWER generates alarms for any changes in power to the device.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a pervious session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWAL_ALARM TWAL_FEEDERERROR TWAL_FEEDERWARNING TWAL_BARCODE TWAL_DOUBLEFEED TWAL_JAM TWAL_PATCHCODE TWAL_POWER TWAL_SKEW

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY

MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_ALARMVOLUME`

CAP_ALARMVOLUME

Description

The volume of a device's audible alarm. Note that this control affects the volume of all alarms; no specific volume control for individual types of alarms is provided.

Application

Take note of the range step, some Sources may only offer a step of 100, which turns the alarm on or off.

Source

If 0, the audible alarm is turned off. All other values control the volume of the alarm.

Windows only - If the alarm is managed in the Source, as opposed to the device, then it should be consistent with the control panel Accessibility Options (i.e., the user should get visual notification if that is the current setting for the desktop).

Values

Type:	`TW_INT32`
Value after MSG_OPENDS:	(may be remembered from a pervious session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	0 - 100

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_ALARMS`

CAP_AUTHOR

Description

The name or other identifying information about the Author of the image. It may include a copyright string.

Values

Type:	`TW_STR128`
Value after MSG_OPENDS:	`“\0”` (empty string)
After MSG_RESET/MSG_RESETALL:	`“\0”` (empty string)
Allowed Values:	Any string

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_CAPTION`
`CAP_TIMEDATE`

CAP_AUTOFEED

Description

If TRUE, the Source will automatically feed the next page from the document feeder after the number of frames negotiated for capture from each page are acquired. CAP_FEEDERENABLED must be TRUE to use this capability.

Application

Set the capability to TRUE to enable the Source's automatic feed process, or FALSE to disable it. After the completion of each transfer, check TW_PENDINGXFERS. Count to determine if the Source has more images to transfer. A -1 means there are more images to transfer but the exact number is not known.

CAP_FEEDERLOADED indicates whether the Source's feeder is loaded. (The automatic feed process continues whenever this capability is TRUE.)

Source

If CAP_FEEDERENABLED equals FALSE, return TWRC_FAILURE / TWCC_CAPSEQERROR (capability is not supported in current settings).

If it is supported, return TWRC_SUCCESS and enable the device's automatic feed process: After all frames negotiated for capture from each page are acquired, put the current document in the output area and advance the next document from the input area to the feeder image acquisition area. If the feeder input area is empty, the automatic feeding process is suspended but should continue when the feeder is reloaded.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TRUE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

All Sources with Feeder Devices

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

CAP_CLEARPAGE
CAP_FEEDERENABLED
CAP_FEEDERLOADED
CAP_FEEDPAGE
CAP_REWINDPAGE

CAP_AUTOMATICCAPTURE

Description

The number of images to automatically capture. This does *not* refer to the number of images to be sent to the Application, use CAP_XFERCOUNT for that.

Source

When 0, Automatic Capture is disabled. When 1 or greater, that number of images is captured by the device.

Automatic capture implies that the device is capable of capturing images without the presence of the Application. This means that it must be possible for the Application to close the Source and reopen it later, after the images have been captured.

Values

Type:	TW_INT32
Value after MSG_OPENDS:	0
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	0 or greater

Containers

MSG_GET	TW_ONEVALUE TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_TIMEBEFOREFIRSTCAPTURE
CAP_TIMEBETWEENCAPTURES
CAP_XFERCOUNT
DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY

CAP_AUTOMATICSENSEMEDIUM

Description

Configures a Source to check for paper in the Automatic Document Feeder, and if it finds any, then automatically capture all of its images from the Feeder. If the Feeder is empty when acquisition starts, then all images are automatically captured from the Flatbed.

Application

This capability offers a less complex method to let the Source automatically choose whether to acquire images from the Automatic Document Feeder or from the Flatbed.

Note: If this capability is not supported, Applications can simulate this behavior by examining `CAP_FEEDERLOADED`. If it is `TRUE`, set `CAP_FEEDERENABLED` to `TRUE`. If it is `FALSE`, set `CAP_FEEDERENABLED` to `FALSE`. And after that begin to acquire images.

Source

If the Source supports `CAP_PAPERDETECTABLE`, and it has both an Automatic Document Feeder and a Flatbed, then it should support this capability.

When this capability is set to `TRUE` the Source ignores the value of `CAP_FEEDERENABLED`. It always attempts to acquire its first image from the Automatic Document Feeder. If paper is not present, then images are acquired from the Flatbed.

When this capability is set to `FALSE` the source of images is determined by `CAP_FEEDERENABLED`.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All ADF/Flatbed combo scanners.

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

`CAP_FEEDERENABLED`
`CAP_FEEDERLOADED`

CAP_AUTOSCAN

Description

This capability is intended to boost the performance of a Source. The fundamental assumption behind AutoScan is that the device is able to capture the number of images indicated by the value of `CAP_XFERCOUNT` without waiting for the Application to request the image transfers. This is only possible if the device has internal buffers capable of caching the images it captures.

Some high volume devices are incapable of anything but `CAP_AUTOSCAN` being equal to `TRUE`. If the scanner supports both `TRUE` and `FALSE` it should set its `MSG_RESET` value to `TRUE`. Most of the applications on the market expect this behavior.

Application

The application should check the `TW_PENDINGXFERS.Count`, and continue to scan until it becomes `0`.

When `AutoScan` is set to `TRUE`, the `Application` should not rely on just the paper sensors (for example, `CAP_FEEDERLOADED`) to determine if there are images to be transferred. The latency between the `Source` and the `Application` makes it very likely that at the time the sensor reports `FALSE`, there may be more than one image waiting for the transfer inside of the device's buffers. Applications should use the `TW_PENDINGXFERS.Count` returned from `DG_CONTROL` / `DAT_PENDINGXFERS` / `MSG_ENDXFER` to determine whether or not there are more images to be transferred.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TRUE`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

When a mid- to high-volume Source supports transfer of multiple images ahead of retrieval.

TWAIN Version Introduced

Version 1.6

See Also

Best Practices

`CAP_AUTOFEED`
`CAP_MAXBATCHBUFFERS`

`DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER`

CAP_BATTERYMINUTES

Description

The minutes of battery power remaining to the device.

This is a read only capability.

Source

-2 indicates that the available power is infinite.

-1 indicates that the device cannot report the remaining battery power.

0 and greater indicates the minutes of battery life remaining.

Values

Type:	`TW_INT32`
Allowed Values:	-2, -1, 0, and greater

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_BATTERYPERCENTAGE`
`CAP_POWERSUPPLY`

CAP_BATTERYPERCENTAGE

Description

When used with MSG_GET, return the percentage of battery power level on camera. If -1 is returned, it indicates that the battery is not present.

This is a read only capability.

Application

Use this capability with MSG_GET to indicate to the user about the battery power status. It is recommended to use CAP_POWERSUPPLY to identify the power source first.

Source

-2 indicates that the available power is infinite.

-1 indicates that the device cannot report the remaining battery power.

0 to 100 indicates the percentage of battery life remaining.

Values

Type:	TW_INT16
Allowed Values:	-2, -1, 0 to 100.

Containers

MSG_GET	TW_ONEVALUE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None. Highly recommended for digital cameras that are equipped with batteries.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_BATTERYMINUTES
CAP_POWERSUPPLY

CAP_CAMERAENABLED

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.

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`.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(dependent on `ICAP_PIXELTYPE`)
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`CAP_CAMERAORDER`
`CAP_CAMERASIDE`

`DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY`

CAP_CAMERAORDER

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 if `CAP_CAMERAENABLED` is set for more than one pixel type on the same camera, putting the scanner into Single Document Multiple Images mode. `DAT_FILESYSTEM` is required to perform the proper addressing. `DAT_FILESYSTEM` is used to address each camera.

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

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	All applicable `ICAP_PIXELTYPE` values

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`CAP_CAMERAENABLED`

`DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY`

CAP_CAMERAPREVIEWUI

Description

This capability queries the Source for UI support for preview mode. If `TRUE`, the Source supports preview UI.

This is a read only capability.

Application

Use this capability to query the preview UI support by the Source. However, the application can choose to use the Source's UI or not even if the Source supports it.

Values

Type:	`TW_BOOL`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION` **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None. Highly recommended for digital cameras.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_CAMERASIDE

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.

Consider the following sequence:

```
CAP_CAMERASIDE set to TWCS_TOP
ICAP_XRESOLUTION set to 200
CAP_CAMERASIDE set to TWCS_BOTTOM
ICAP_XRESOLUTION set to 300
CAP_CAMERASIDE set to TWCS_BOTH
```

At this point getting the value of `ICAP_XRESOLUTION` will return a value of 200, even though the bottom is currently set to 300. This is acceptable behavior. It is up to the application to correctly use `CAP_CAMERASIDE`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWCS_BOTH`

Allowed Values:	`TWCS_BOTH`
	`TWCS_TOP`
	`TWCS_BOTTOM`

Containers

MSG_GET	`TW_ENUMERATION` `TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.91

See Also

Best Practices

`DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY`

CAP_CAPTION

Description

A general note about the acquired image.

Values

Type:	`TW_STR255`
Value after MSG_OPENDS:	"`\0`" (empty string)
After MSG_RESET/MSG_RESETALL:	"`\0`" (empty string)
Allowed Values:	Any string

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_AUTHOR`
`CAP_TIMEDATE`

CAP_CLEARPAGE

Description

If TRUE, the Source will eject the current page being acquired from and will leave the feeder acquire area empty.

If CAP_AUTOFEED is TRUE, a fresh page will be advanced.

CAP_FEEDERENABLED must equal TRUE to use this capability.

This capability must have been negotiated as an extended capability to be used in States 5 and 6.

Application

Do a MSG_SET on this capability to advance the document in the feeder acquire area to the output area and abort all transfers pending on this page.

This capability is used in States 5 and 6 by applications controlling the Source's feeder (usually without the Source user interface).

This capability can also be used while CAP_AUTOFEED equals TRUE to abort all remaining transfers on this page and continue with the next page.

Source

If CAP_FEEDERENABLED equals FALSE, return TWRC_FAILURE / TWCC_CAPSEQERROR (capability is not supported in current settings).

If supported, advance the document in the feeder-acquire area to the output area and abort all pending transfers from this page.

The Source will perform this action once whenever the capability is MSG_SET to TRUE. The Source should then revert the Current value to FALSE.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	FALSE
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

CAP_AUTOFEED	CAP_FEEDERLOADED
CAP_EXTENDEDCAPS	CAP_FEEDPAGE
CAP_FEEDERENABLED	CAP_REWINDPAGE

CAP_CUSTOMDSDATA

Description

Allows the application to query the data source to see if it supports the new operation triplets DG_CONTROL/ DAT_CUSTOMDSDATA / MSG_GET and DG_CONTROL/ DAT_CUSTOMDSDATA / MSG_SET.

If TRUE, the source will support the DG_CONTROL/ DAT_CUSTOMDSDATA/MSG_GET message.

This is a read only capability.

Values

Type:	TW_BOOL
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

DG_CONTROL / DAT_CUSTOMDSDATA / MSG_GET

CAP_CUSTOMINTERFACEGUID

Description

Uniquely identifies an interface for a Data Source, so that an Application can properly interpret its custom content.

This is a read only capability.

Application

Use the value from this capability to interpret all of the numeric values referenced in the See Also section below.

Consider the following example, which results in three GUID's, one for Vendor ABC and two for Vendor XYZ:

- Vendor ABC's Scanner models Fred and Wilma have a custom capability called `CAP_MYFEATURE` with a numeric value of `0x8001`.
- Vendor XYZ's Scanner model Barney has a custom capability `CAP_OURFEATURE` with a numeric value of `0x8001`, but their Scanner Model Betty has a different custom capability `CAP_BETTERFEATURE` with a numeric value of `0x8001`.

The challenge for the Application is to know what `0x8001` means. Historically, this has been determined from the Source's `TW_IDENTITY` structure. But this is hard to maintain, and requires the Application to constantly update its recognition code, even in the case of Vendor ABC whose interface stays the same from one model to the next.

Using `CAP_CUSTOMINTERFACEGUID` the Application can immediately identity Vendor ABC's unique interface, without having to check its `TW_IDENTITY` structure.

Source

The Source writer is responsible for creating a GUID. This GUID guarantees that the custom numeric values have exactly the same meaning for any Source that reports that GUID.

If you need to create a GUID, but don't know how, go to the TWAIN Working Group website and click on FAQ.

Values

Type:	`TW_STR255`
Allowed Values:	A string in GUID format
	`{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_RESET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

Sources that support Custom Content.

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

Chapter 6, "Custom Data Argument Types" (`DAT_CUSTOMBASE`)

Chapter 6, "Custom Messages" (`MSG_CUSTOMBASE`)

Chapter 8, "Capability Constants" (`CAP_CUSTOMBASE`)

Chapter 8, "CAP_DEVICEEVENT" (`TWDE_CUSTOMEVENTS`)

Chapter 8, "Return Codes (TWRC_)" (`TWRC_CUSTOMBASE`)

Chapter 8, "Condition Codes (TWCC_)" (`TWCC_CUSTOMBASE`)

CAP_DEVICEEVENT

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_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_CHECKSTATUS and remove unsupported events when an Application requests events not supported by the Source.

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

Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(empty array)
Allowed Values:	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

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	TW_ONEVALUE TW_ARRAY
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ARRAY
MSG_RESET	TW_ARRAY
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT (from Source to Application)
DG_CONTROL / DAT_DEVICEEVENT / MSG_GET

Device Events Article

CAP_DEVICEONLINE

Description

If TRUE, the physical hardware (e.g., scanner, digital camera, image database, etc.) that represents the image source is attached, powered on, and communicating.

This is a read only capability.

Application

This capability can be issued at any time to determine the availability of the image source hardware.

Source

The receipt of this capability request should trigger a test of the status of the physical link to the image source. The source should not assume that the link is still active since the last transaction, but should issue a transaction that actively tests this condition.

Values

Type:	TW_BOOL
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All image Sources

TWAIN Version Introduced

Version 1.6

See Also

Best Practices

CAP_DEVICETIMEDATE

Description

The date and time of the device's clock.

Managed in the form "YYYY/MM/DD HH:mm:SS:sss" where YYYY is the year, MM is the numerical month, DD is the numerical day, HH is the hour, mm is the minute, SS is the second, and sss is the millisecond.

Source

The internal date and time of the device. Be sure to leave the space between the ending of the date and the beginning of the time fields. All fields must be specified for `MSG_SET`.

Values

Type:	`TW_STR32`
Value after MSG_OPENDS:	(selected by the data source writer)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any date

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_TIMEDATE`

CAP_DOUBLEFEEDDETECTION

Description

Enables or disables double feed detection.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(empty array)
Allowed Values:	TWDF_ULTRASONIC TWDF_BYLENGTH TWDF_INFRARED

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	TW_ONEVALUE TW_ARRAY
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ARRAY
MSG_RESET	TW_ARRAY
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

CAP_DOUBLEFEEDDETECTIONLENGTH
CAP_DOUBLEFEEDDETECTIONRESPONSE
CAP_DOUBLEFEEDDETECTIONSENSITIVITY

CAP_DOUBLEFEEDDETECTIONLENGTH

Description

When CAP_DOUBLEFEEDDETECTION includes TWDF_BYLENGTH, it allows an Application to set the minimum length for detecting double feed documents. A value of zero always means "off".

Source

The current value of this setting specifies the differences of paper length.

If the value is not supported, return TWRC_CHECKSTATUS and set to the closest supported value.

Values

Type:	TW_FIX32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	0 to 32767 in ICAP_UNITS

Containers

MSG_GET	TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

CAP_DOUBLEFEEDDETECTION
ICAP_UNITS

CAP_DOUBLEFEEDDETECTIONRESPONSE

Description

When CAP_DOUBLEFEEDDETECTION is set to anything but Disabled, it allows an Application to set how the source will respond to detecting a double feed.

- TWDP_STOP – when a multi-feed is detected the driver must end the scanner session, returning TWCC_PAPERDOUBLEFEED for the an DAT_IMAGE*XFER call that applied to the bad image.
- TWDP_STOPANDWAIT – the driver/device will manage the multi-feed; the application is not notified; after the problem is fixed by the operator, image capture is continued as if no multi-feed occurred.
- TWDP_SOUND – the driver/device will generate an audible alert when a multi-feed is detected
- TWDP_DONOTIMPRINT – the device will not print on multi-fed documents. All combinations are valid, excluding TWDP_STOP and TWDP_STOPANDWAIT as only one of those can appear at a time. If the driver is asked for both, it may pick one and return TWRC_CHECKSTATUS.

Application

The Application cannot ask for TWDP_STOP and TWDP_STOPANDWAIT at the same time.

After it receives TWCC_PAPERDOUBLEFEED, the Application can use DAT_PENDINGXFERS / MSG_GET to see if it can resume the current session or if it must start a new one. If TWDP_STOPANDWAIT is in the list, then Application should not expect any errors to be returned if double-feed occurs.

Source

Application cannot ask for TWDP_STOP and TWDP_STOPANDWAIT at the same time. If an Application sends TWDP_STOP and TWDP_STOPANDWAIT at the same time, then pick more suitable one and return TWRC_CHECKSTATUS.

If the Source UI and Indicators are disabled, or TWDP_STOP is in the list, then return TWRC_FAILURE/TWCC_PAPERDOUBLEFEED.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWDP_STOP TWDP_STOPANDWAIT TWDP_SOUND TWDP_DONOTIMPRINT

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	TW_ONEVALUE TW_ARRAY

MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`CAP_DOUBLEFEEDDETECTION`

CAP_DOUBLEFEEDDETECTIONSENSITIVITY

Description

When `CAP_DOUBLEFEEDDETECTION` includes `TWDF_ULTRASONIC`, allows an Application to set how sensitive the double feed detection is.

Source

The Source has to remap the scanner's sensor sensitivity to the TWAIN defined sensitivity. See allowed values.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWUS_LOW`
	`TWUS_MEDIUM`
	`TWUS_HIGH`

Containers

MSG_GET	`TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`CAP_DOUBLEFEEDDETECTION`

CAP_DUPLEX

Description

This indicates whether the scanner supports duplex. If so, it further indicates whether one-path or two-path duplex is supported.

This is a read only capability.

Application

Application can send `MSG_GET` to find out whether the scanner supports duplex.

Source

Source should determine level of duplex support returning the values accordingly.

Values

Type:	`TW_UINT16`

Allowed Values:	`TWDX_NONE`
	`TWDX_1PASSDUPLEX`
	`TWDX_2PASSDUPLEX`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Sources that support Duplex scanning.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`CAP_DUPLEXENABLED`

CAP_DUPLEXENABLED

Description

The user can set the duplex option to be TRUE or FALSE. If TRUE, the scanner scans both sides of a paper; otherwise, the scanner will scan only one side of the image.

Application

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

Source

Source should return TRUE or FALSE based on the level of duplex support; otherwise, return TWRC_FAILURE / TWCC_CAPUNSUPPORTED.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

All Sources that support Duplex scanning.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

CAP_DUPLEX

CAP_ENABLEDSUIONLY

Description

Allows an application to query a source to see if it implements the new user interface settings dialog. If a source reports that it has the capability `CAP_ENABLEDSUIONLY`, then it must implement the operation triplet `DG_CONTROL/ DAT_USERINTERFACE/ MSG_ENABLEDSUIONLY` to display the source user interface without acquiring an image.

If `TRUE`, the source will support the `DG_CONTROL/ DAT_USERINTERFACE / MSG_ENABLEDSUIONLY` message.

This is a read only capability.

Values

Type:	`TW_BOOL`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION` **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDSUIONLY`

CAP_ENDORSER

Description

Allows the application to specify the scanner's starting endorser / imprinter number.

When available, use `CAP_PRINTERINDEX`, instead. See the Legacy Issues section on `CAP_ENDORSER` vs `CAP_PRINTER` for more information.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`1`
Allowed Values:	Any value

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`	 **// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

Sources that control an Endorser.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`CAP_PRINTERINDEX`

CAP_EXTENDEDCAPS

Description

Allows the application and Source to negotiate capabilities to be used in States 5, 6 and 7.

Application

`MSG_GET` and `MSG_GETCURRENT` return an array of the capabilities the Source supports in States 5, 6 and 7. If either the Source or the application is older than TWAIN 2.3, use `MSG_GET` to get the list of allowed capabilities, and `MSG_GETCURRENT` to check the capabilities currently set.

`MSG_SET` is only needed with Sources older than TWAIN 2.3, to set the capabilities the application wants to negotiate in States 5, 6 and 7.

Stated another way, beginning with TWAIN 2.3 `CAP_EXTENDEDCAPS` works more like `CAP_SUPPORTEDCAPS`; it should be treated as a read only array, but data sources must still permit `MSG_SET` and `MSG_RESET` operations for legacy applications.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(all values supported by the data source)
After MSG_RESET/MSG_RESETALL:	(all values supported by the data source)
Allowed Values:	Any *x*CAP_*xxxx*

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_SUPPORTEDCAPS`

CAP_FEEDERALIGNMENT

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.

Application

The Application can use this to determine if it must center the framing information sent to the Source. With some Sources it might be possible for the Application to select whether the paper is center fed or not.

Source

Use this capability to report the state of the feeder.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWFA_NONE`
	`TWFA_LEFT`
	`TWFA_CENTER`
	`TWFA_RIGHT`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE` if supported	
MSG_SETCONSTRAINT	`TW_ONEVALUE` if supported	
MSG_RESET	`TW_ONEVALUE` if supported	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_FEEDERENABLED

Description

If TRUE, Source must acquire data from the document feeder acquire area and other feeder capabilities can be used. If FALSE, Source must acquire data from the non-feeder acquire area and no other feeder capabilities can be used.

Application

The application should MSG_SET this capability to TRUE before attempting to use any other feeder capabilities. This sets the current acquire area to the feeder area (it may not be a different physical area on some Sources).

The application can MSG_SET this capability to FALSE to use the Source's non-feeder acquisition area and disallow the further use of feeder capabilities.

Source

This setting should reflect the current acquire area:

If TRUE, feeder acquire area should be used

If FALSE, use non-feeder acquire area

Usually, the feeder acquire area and non-feeder acquire area of the Source will be the same. For example, a flatbed scanner may feed a page onto the flatbed platen then scanning always takes place from the platen.

The counter example is a flatbed scanner that moves the scan bar over the platen when CAP_FEEDERENABLED is FALSE, but moves the paper over the scan bar when it is TRUE.

Default Support Guidelines for Sources

- Flatbed scanner (without an optional ADF installed) - Default to FALSE. Do not allow setting to TRUE (return TWRC_FAILURE / TWCC_BADVALUE) but support the capability (never return TWRC_FAILURE / TWCC_CAPUNSUPPORTED).
- A device that uses the same acquire area for feeder and non-feeder, and has a feeder installed - Default to TRUE and allow settings to TRUE or FALSE (meaning allow or don't allow other feeder capabilities).
- A device that operates differently when acquiring from the feeder and non-feeder areas (for example, physical pages sizes are different) - Default to preferred area and allow setting to either TRUE or FALSE.
- A sheet feed scanner or image database - Default to TRUE (meaning there is only one acquire area - the feeder area) and do not allow setting to FALSE (return TWRC_FAILURE / TWCC_BADVALUE).
- A handheld scanner would not support this capability (return TWRC_FAILURE / TWCC_CAPUNSUPPORTED).

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
	`TW_ENUMERATION`	**// 2.0 and higher**
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

All Sources with feeder devices

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_AUTOFEED`
`CAP_CLEARPAGE`
`CAP_FEEDERLOADED`
`CAP_FEEDERPOCKET`
`CAP_FEEDERPREP`
`CAP_FEEDPAGE`
`CAP_REWINDPAGE`

CAP_FEEDERLOADED

Description

Reflect whether there are documents loaded in the Source's feeder.

This is a read only capability.

Application

Used by application to inquire whether there are documents loaded in the Source's feeder.

Source

If `CAP_FEEDERENABLED` equals `FALSE`, return `TWRC_FAILURE` / `TWCC_CAPSEQERROR` (capability is not supported in current settings).

If `CAP_FEEDERENABLED` equals `TRUE`, return the status of the feeder (documents loaded = `TRUE`; no documents loaded = `FALSE`).

The Source is responsible for reporting instructions to users on using the device. This includes instructing the user to place documents in the feeder when `CAP_FEEDERLOADED` equals `FALSE` and the application has requested a feed page (manually or automatically).

Values

Type:	`TW_BOOL`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION` **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Sources with feeder devices

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_AUTOFEED`	`CAP_FEEDPAGE`
`CAP_CLEARPAGE`	`CAP_REWINDPAGE`
`CAP_FEEDERENABLED`	

CAP_FEEDERORDER

Description

TWFO_FIRSTPAGEFIRST if the feeder starts with the top of the first page.
TWFO_LASTPAGEFIRST is the feeder starts with the top of the last page.

Source

Source must support MSG_SET if the scanner is capable of changing feeder order.

Application

An Application can use this to determine if it should reorganize the stream of images received from a Source.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWFO_FIRSTPAGEFIRST TWFO_LASTPAGEFIRST

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 (if MSG_Set is supported)
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE if supported	
MSG_SETCONSTRAINT	TW_ONEVALUE if supported	
MSG_RESET	TW_ONEVALUE if supported	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_FEEDERENABLED

CAP_FEEDERPOCKET

Description

Report what pockets are available to receive paper as it exits from the device..

Source

This capability enumerates the available output or collation pockets on the device. `TWFP_POCKET1 - TWFP_POCKET16` are organized from top to bottom and left to right, facing in the direction of the motion of the paper.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWFP_POCKET1 - TWFP_POCKET16`
	`TWFP_POCKETERROR`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`CAP_FEEDERENABLED`
`CAP_MICRENABLED`

CAP_FEEDERPREP

Description

Improve the movement of paper through a scanner ADF.

Source

If `CAP_FEEDERENABLED` is `TRUE`, and `CAP_FEEDERPREP` is `TRUE`, then the scanner will perform any action needed to improve the movement of paper through the transport.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`FALSE`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`CAP_FEEDERENABLED`
`CAP_MICRENABLED`

CAP_FEEDPAGE

Description

If TRUE, the Source will eject the current page and advance the next page in the document feeder into the feeder acquire area.

If CAP_AUTOFEED is TRUE, the same action just described will occur and CAP_AUTOFEED will remain active.

CAP_FEEDERENABLED must equal TRUE to use this capability.

This capability must have been negotiated as an extended capability to be used in States 5 and 6.

Application

Do a MSG_SET to TRUE on this capability to advance the next document in the feeder to the feeder acquire area.

This capability is used in States 5 and 6 by applications controlling the Source's feeder (usually without the Source's user interface).

This capability can also be used while CAP_AUTOFEED equals TRUE to abort all remaining transfers on this page and continue with the next page.

Source

If CAP_FEEDERENABLED equals FALSE, return TWRC_FAILURE / TWCC_CAPSEQERROR (capability is not supported in current settings).

If supported, advance the document in the feeder-acquire area to the output area and abort all pending transfers from this page.

Advance the next page in the input area to the feeder acquire area. If there are no documents in the input area, return: TWRC_FAILURE / TWCC_BADVALUE.

The Source will perform this action once whenever the capability is MSG_SET to TRUE. The Source should then revert the Current value to FALSE.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	FALSE
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_AUTOFEED`
`CAP_CLEARPAGE`
`CAP_EXTENDEDCAPS`
`CAP_FEEDERENABLED`
`CAP_FEEDERLOADED`
`CAP_REWINDPAGE`

CAP_INDICATORS

Description

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

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

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

Application

If the application plans to enable the Source with `TW_USERINTERFACE. ShowUI = FALSE`, it can also suppress the Source's progress indicator by using this capability.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	`TRUE`
After MSG_RESET/MSG_RESETALL:	`TRUE`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.0 and higher**
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.1

See Also

Best Practices

`CAP_INDICATORSMODE`
`DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS`

CAP_INDICATORSMODE

Description

Specify what messages can be displayed if `ICAP_INDICATORS` is `TRUE`.

Source

The default value must include all possible supported values.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(all values supported by the data source)
After MSG_RESET/MSG_RESETALL:	(all values supported by the data source)

Allowed Values:	`TWCI_INFO`
	`TWCI_WARNING`
	`TWCI_ERROR`
	`TWCI_WARMUP`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`CAP_INDICATORS`

CAP_JOBCONTROL

Description

Allows multiple jobs in batch mode. The application can decide how the job can be processed, according to the flags listed below.

`TWJC_NONE`	No job control.
`TWJC_JSIC`	Detect and include job separator and continue scanning.
`TWJC_JSIS`	Detect and include job separator and stop scanning.
`TWJC_JSXC`	Detect and exclude job separator and continue scanning.
`TWJC_JSXS`	Detect and exclude job separator and stop scanning.

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.

To distinguish between jobs, a job separator sheet containing patch code can be inserted. If the application knows the how to save different jobs, the `TWJC_JSIC` or `TWJC_JSXC` can be used. When this job separator is detected, the application will give a separate name for each job. If the application does not know how to save different jobs, it can use `TWJC_JSIS` or `TWJC_JSXS` to stop scanning and ask the user for different job name.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWJC_NONE`

Allowed Values:	`TWJC_NONE`
	`TWJC_JSIC`
	`TWJC_JSIS`
	`TWJC_JSXC`
	`TWJC_JSXS`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

```
DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_GET
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET
DG_CONTROL / DAT_PENDINGXFERS / MSG_STOPFEEDER
```

CAP_LANGUAGE

Description

Allows Application and Source to identify which languages they have in common for the exchange of string data, and to select the language of the internal UI.

Note: Since the `TWLG_xxxx` codes include language and country data, there is no separate capability for selecting the country.

Application

In multi-lingual environments, it is the responsibility of the Application to recall the last selected language for a given User.

Source

The current value of this setting specifies the language used by the Source (and possibly the device). The Source must first default to the Application's current language. If that fails then it must default to the User's Locale (c.f., the Win32 call GetLocaleInfo()). If that fails then the Source should make the best choice it can, preferably using a common secondary language (i.e., English, French…).

Note:

- `TWLG_ARABIC_UAE` is for the United Arabic Emirates.
- `TWLG_CHINESE_PRC` is for the People's Republic of China

Values

Type:	`TW_UINT16`
Default Value:	**In order of priority:** 1) `appIdentity->Version.Language` 2) `TWLG_USERLOCALE` 3) Source's choice

Allowed Values:	`TWLG_USERLOCALE`
	// pre 1.8 values…
	`TWLG_DAN`
	`TWLG_DUT`
	`TWLG_ENG`
	`TWLG_FCF`
	`TWLG_FIN`
	`TWLG_FRN`
	`TWLG_GER`
	`TWLG_ICE`
	`TWLG_ITN`
	`TWLG_NOR`
	`TWLG_POR`
	`TWLG_SPA`
	`TWLG_SWE`
	`TWLG_USA`

// 1.8 should use these…

```
TWLG_AFRIKAANS
TWLG_ALBANIA
TWLG_ARABIC
TWLG_ARABIC_ALGERIA
TWLG_ARABIC_BAHRAIN
TWLG_ARABIC_EGYPT
TWLG_ARABIC_IRAQ
TWLG_ARABIC_JORDAN
TWLG_ARABIC_KUWAIT
TWLG_ARABIC_LEBANON
TWLG_ARABIC_LIBYA
TWLG_ARABIC_MOROCCO
TWLG_ARABIC_OMAN
TWLG_ARABIC_QATAR
TWLG_ARABIC_SAUDIARABIA
TWLG_ARABIC_SYRIA
TWLG_ARABIC_TUNISIA
TWLG_ARABIC_UAE
TWLG_ARABIC_YEMEN
TWLG_BASQUE
TWLG_BYELORUSSIAN
TWLG_BULGARIAN
TWLG_CATALAN
TWLG_CHINESE
TWLG_CHINESE_HONGKONG
TWLG_CHINESE_PRC
TWLG_CHINESE_SINGAPORE
TWLG_CHINESE_SIMPLIFIED
TWLG_CHINESE_TAIWAN
TWLG_CHINESE_TRADITIONAL
TWLG_CROATIA
TWLG_CZECH
TWLG_DANISH
TWLG_DUTCH
TWLG_DUTCH_BELGIAN
TWLG_ENGLISH
TWLG_ENGLISH_AUSTRALIAN
TWLG_ENGLISH_CANADIAN
TWLG_ENGLISH_IRELAND
TWLG_ENGLISH_NEWZEALAND
TWLG_ENGLISH_SOUTHAFRICA
TWLG_ENGLISH_UK
TWLG_ENGLISH_USA
TWLG_ESTONIAN
TWLG_FAEROESE
TWLG_FARSI
TWLG_FINNISH
TWLG_FRENCH
TWLG_FRENCH_BELGIAN
TWLG_FRENCH_CANADIAN
TWLG_FRENCH_LUXEMBOURG
TWLG_FRENCH_SWISS
TWLG_GERMAN
TWLG_GERMAN_AUSTRIAN
TWLG_GERMAN_LUXEMBOURG
TWLG_GERMAN_LIECHTENSTEIN
TWLG_GERMAN_SWISS
TWLG_GREEK
TWLG_HEBREW
TWLG_HUNGARIAN
TWLG_ICELANDIC
TWLG_INDONESIAN
TWLG_ITALIAN
TWLG_ITALIAN_SWISS
TWLG_JAPANESE
TWLG_KOREAN
TWLG_KOREAN_JOHAB
TWLG_LATVIAN
TWLG_LITHUANIAN
TWLG_NORWEGIAN
TWLG_NORWEGIAN_BOKMAL
TWLG_NORWEGIAN_NYNORSK
TWLG_POLISH
TWLG_PORTUGUESE
TWLG_PORTUGUESE_BRAZIL
TWLG_ROMANIAN
TWLG_RUSSIAN
TWLG_SERBIAN_LATIN
TWLG_SLOVAK
TWLG_SLOVENIAN
TWLG_SPANISH
TWLG_SPANISH_MEXICAN
TWLG_SPANISH_MODERN
TWLG_SWEDISH
TWLG_THAI
TWLG_TURKISH
TWLG_UKRANIAN
TWLG_ASSAMESE
TWLG_BENGALI
TWLG_BIHARI
TWLG_BODO
TWLG_DOGRI
TWLG_GUJARATI
TWLG_HARYANVI
TWLG_HINDI
TWLG_KANNADA
TWLG_KASHMIRI
TWLG_MALAYALAM
TWLG_MARATHI
```

TWLG_MARWARI	TWLG_SIKKIMI
TWLG_MEGHALAYAN	TWLG_SWEDISH_FINLAND
TWLG_MIZO	TWLG_TAMIL
TWLG_NAGA	TWLG_TELUGU
TWLG_ORISSI	TWLG_TRIPURI
TWLG_PUNJABI	TWLG_URDU
TWLG_PUSHTU	TWLG_VIETNAMESE
TWLG_SERBIAN_CYRILLIC	

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_MAXBATCHBUFFERS

Description

Describes the number of pages that the scanner can buffer when `CAP_AUTOSCAN` is enabled.

Application

`MSG_GET` returns the supported values

`MSG_SET` sets the current number pages to be buffered (if the Source allows this to be set)

Source

If supported, report the maximum batch buffer settings during `MSG_GET`. If `MSG_SET` is supported, limit batch buffers to the requested value for future transfers.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_ENUMERATION`
	`TW_RANGE`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_ENUMERATION`
	`TW_RANGE`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

When a mid- to high-volume Source supports transfer of multiple images ahead of retrieval.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_AUTOSCAN`

CAP_MICRENABLED

Description

Get this capability to determine if the Source supports check scanning. If set to TRUE check scanning is enabled, if set to FALSE check scanning is disabled.

Source

When disabled the scanner ignores all related capabilities (refer to the See Also section).

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

CAP_FEEDERPREP
CAP_FEEDERPOCKET

DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET

CAP_PAPERDETECTABLE

Description

This capability determines whether the device has a paper sensor that can detect documents on the ADF or Flatbed.

This is a read only capability.

Application

If the source returns `FALSE`, the application should not rely on values such as `CAP_FEEDERLOADED`, and continue as if the paper is loaded.

Source

If supported, the source is responsible for detecting whether document is loaded or not.

Values

Type:	`TW_BOOL`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION` **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Sources with feeder devices

TWAIN Version Introduced

Version 1.6

See Also

Best Practices

`CAP_FEEDERLOADED`

CAP_PAPERHANDLING

Description

Use this capability to control paper handling. This capability may affect scanning speed.

Application

Use it to improve paper handling.

The Source may not support all combinations of values it returns in `MSG_GET`, so check the current value if the Source returns `TWRC_CHECKSTATUS`.

Source

If the Source does not support a combination of values, then it should pick the most suitable values and return `TWRC_CHECKSTATUS`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWPH_NORMAL`

Allowed Values:	`TWPH_NORMAL`
	`TWPH_FRAGILE`
	`TWPH_THICK`
	`TWPH_TRIFOLD`
	`TWPH_PHOTOGRAPH`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

CAP_POWERSAVETIME

Description

When used with `MSG_SET`, set the camera power down timer in seconds. When used with `MSG_GET`, return the current setting of the power down time.

Application

Use this capability with `MSG_SET` to set the user selected camera power down time, when no activity is detected by the camera. The default value of -1 means no power down, power is always on.

Values

Type:	`TW_INT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	>= -1

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_RANGE`	**// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None. Highly recommended for digital cameras.

```
MSG_GET
MSG_SET / MSG_RESET
```

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_POWERSUPPLY

Description

`MSG_GET` reports the kinds of power available to the device. `MSG_GETCURRENT` reports the current power supply in use.

This is a read only capability.

Values

Type:	`TW_UINT16`
Allowed Values:	`TWPS_EXTERNAL` `TWPS_BATTERY`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER

Description

`MSG_GET` returns the current list of available printer devices, along with the one currently being used for negotiation. `MSG_SET` selects the current device for negotiation, and optionally constrains the list. `MSG_RESET` restores all the available devices (useful after `MSG_SET` has been used to constrain the list).

Top/Bottom refer to duplex devices, and indicate if the printer is writing on the top or the bottom of the sheet of paper. Simplex devices use the top settings.

Before/After indicates whether printing occurs before or after the sheet of paper has been scanned.

Application

Use this capability to determine which printers are available for negotiation, and to select a specific printer prior to negotiation.

Source

Imprinters are used to print data on documents at the time of scanning, and may be used for any purpose. Endorsers are more specific in nature, stamping some kind of proof of scanning on the document. Applications may opt to use imprinters for endorsing documents.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWPR_IMPRINTERTOPBEFORE`
	`TWPR_IMPRINTERTOPAFTER`
	`TWPR_IMPRINTERBOTTOMBEFORE`
	`TWPR_IMPRINTERBOTTOMAFTER`
	`TWPR_ENDORSERTOPBEFORE`
	`TWPR_ENDORSERTOPAFTER`
	`TWPR_ENDORSERBOTTOMBEFORE`
	`TWPR_ENDORSERBOTTOMAFTER`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Sources that control a printer type device.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_PRINTERCHARROTATION`	`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERENABLED`	`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERFONTSTYLE`	`CAP_PRINTERMODE`
`CAP_PRINTERINDEX`	`CAP_PRINTERSTRING`
`CAP_PRINTERINDEXLEADCHAR`	`CAP_PRINTERSTRINGPREVIEW`
`CAP_PRINTERINDEXMAXVALUE`	`CAP_PRINTERSUFFIX`
`CAP_PRINTERINDEXNUMDIGITS`	`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERENABLED

Description

Turns the current CAP_PRINTER device on or off.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

All Sources that control a printer type device.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER	CAP_PRINTERINDEXSTEP
CAP_PRINTERCHARROTATION	CAP_PRINTERINDEXTRIGGER
CAP_PRINTERFONTSTYLE	CAP_PRINTERMODE
CAP_PRINTERINDEX	CAP_PRINTERSTRING
CAP_PRINTERINDEXLEADCHAR	CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERINDEXMAXVALUE	CAP_PRINTERSUFFIX
CAP_PRINTERINDEXNUMDIGITS	CAP_PRINTERVERTICALOFFSET

CAP_PRINTERCHARROTATION

Description

Specify the angle of rotation of individual characters within the string to be printed by the current printer.

A value of 0 specifies that if you hold a document with the *feed edge* on the left and the printed text facing you, characters are upright. A non-zero value specifies character rotation clockwise from that orientation, in degrees.

Note: This does not control the rotation or angle of the printed string. TWAIN does not offer that capability. Individual characters are rotated in-place.

Many scanners will support only a single value for this capability, or a few multiples of 90

Application

Specify the printer's font orientation for string or strings that will be printed on the page during the next acquisition.

Source

Set the printer's font orientation for string or strings that will be printed on the page during next acquisition. The font's orientation is in relation to the leading edge for a document feeder.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`0`
Allowed Values:	0 to 359

Containers

MSG_GET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_SETCONSTRAINT	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

`CAP_PRINTER`	`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERENABLED`	`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERFONTSTYLE`	`CAP_PRINTERMODE`
`CAP_PRINTERINDEX`	`CAP_PRINTERSTRING`
`CAP_PRINTERINDEXLEADCHAR`	`CAP_PRINTERSTRINGPREVIEW`
`CAP_PRINTERINDEXMAXVALUE`	`CAP_PRINTERSUFFIX`
`CAP_PRINTERINDEXNUMDIGITS`	`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERFONTSTYLE

Description

Designates the printer font styles to be used during the next acquisition.

Application

Specify the printer's font style for all of the string data that will be printed on the page during the next acquisition.

Source

Set the printer's font style for all strings that will be printed on the page during the next acquisition.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWPF_NORMAL`

Allowed Values:	`TWPF_NORMAL`
	`TWPF_ITALIC`
	`TWPF_BOLD`
	`TWPF_SMALLSIZE`
	`TWPF_LARGESIZE`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

`CAP_PRINTER`	`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERCHARROTATION`	`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERENABLED`	`CAP_PRINTERMODE`
`CAP_PRINTERINDEX`	`CAP_PRINTERSTRING`
`CAP_PRINTERINDEXLEADCHAR`	`CAP_PRINTERSTRINGPREVIEW`
`CAP_PRINTERINDEXMAXVALUE`	`CAP_PRINTERSUFFIX`
`CAP_PRINTERINDEXNUMDIGITS`	`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERINDEX

Description

The counter value of the current CAP_PRINTER device.

TWAIN assumes that every printer has an associated integer *counter* whose current value can be printed on each scanned page and automatically incremented.

Other CAP_PRINTER* capabilities control whether the counter is printed, how it is formatted, and when and how the counter is incremented.

Source

This capability represents the counter of the currently selected CAP_PRINTER.
When the current value of this capability is set or changed, set the printer's counter to the new value.
When this capability is read, use the current value of the printer's counter for the current value of this capability.

Values

Type:	TW_UINT32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	1
Allowed Values:	Any values.

Containers

MSG_GET	TW_ONEVALUE	
	TW_RANGE	**// 2.3 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER	CAP_PRINTERINDEXSTEP
CAP_PRINTERCHARROTATION	CAP_PRINTERINDEXTRIGGER
CAP_PRINTERENABLED	CAP_PRINTERMODE
CAP_PRINTERFONTSTYLE	CAP_PRINTERSTRING
CAP_PRINTERINDEXLEADCHAR	CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERINDEXMAXVALUE	CAP_PRINTERSUFFIX
CAP_PRINTERINDEXNUMDIGITS	CAP_PRINTERVERTICALOFFSET

CAP_PRINTERINDEXLEADCHAR

Description

User can set the character to be used for filling the leading digits before the counter value if the counter digits are fewer than `CAP_PRINTERINDEXNUMDIGITS`.

Examples:
```
If the lead character is a zero:   00001
If the lead character is a blank:      1
```

Application

Send a string containing a single character to the source.

Source

Use the first character of the string as the leading character to pad the current counter out to `CAP_PRINTERNUMDIGITS`.

Values

Type:	`TW_STR32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
Allowed Values:	Any value

Containers

MSG_GET	`TW_ONEVALUE, TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE, TW_ENUMERATION`
MSG_SETCONSTRAINT	`TW_ONEVALUE, TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

`CAP_PRINTER`	`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERCHARROTATION`	`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERENABLED`	`CAP_PRINTERMODE`
`CAP_PRINTERFONTSTYLE`	`CAP_PRINTERSTRING`
`CAP_PRINTERINDEX`	`CAP_PRINTERSTRINGPREVIEW`
`CAP_PRINTERINDEXMAXVALUE`	`CAP_PRINTERSUFFIX`
`CAP_PRINTERINDEXNUMDIGITS`	`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERINDEXMAXVALUE

Description

The User can set the maximum value of the counter of the current `CAP_PRINTER` device. After the counter reaches this value, it will automatically reset to the value specified by `CAP_PRINTERINDEX`.

Application

Set this value to specify the counter's maximum value for the current `CAP_PRINTER` device. Set it to 0 for the device's maximum limit.

Source

This value allows the user to set maximum value of the current printer's counter.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`0`
Allowed Values:	Any value

Containers

MSG_GET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_SETCONSTRAINT	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

`CAP_PRINTER`
`CAP_PRINTERCHARROTATION`
`CAP_PRINTERENABLED`
`CAP_PRINTERFONTSTYLE`
`CAP_PRINTERINDEX`
`CAP_PRINTERINDEXLEADCHAR`
`CAP_PRINTERINDEXNUMDIGITS`
`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERMODE`
`CAP_PRINTERSTRING`
`CAP_PRINTERSTRINGPREVIEW`
`CAP_PRINTERSUFFIX`
`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERINDEXNUMDIGITS

Description

Right justify the counter of the current `CAP_PRINTER` device. The fill character is set by `CAP_PRINTERLEADCHAR`.

Application

Set it to 0 to make it left justified. Allowing the counter to exceed the number of available digits results in undefined behavior.

Source

Set the current printer's counter field width (the numbers are right justified). If set to zero the numbers are left justified.

If the counter value exceeds the number of available digits, the source can choose the action. For example, the source could show the correct number, adding the extra needed digits. It could truncate the number to the specified number of digits. Or, finally, it could end the session with an error.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`0`
Allowed Values:	Positive values

Containers

MSG_GET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_SETCONSTRAINT	`TW_ONEVALUE, TW_ENUMERATION, TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

CAP_PRINTER	CAP_PRINTERINDEXSTEP
CAP_PRINTERCHARROTATION	CAP_PRINTERINDEXTRIGGER
CAP_PRINTERENABLED	CAP_PRINTERMODE
CAP_PRINTERFONTSTYLE	CAP_PRINTERSTRING
CAP_PRINTERINDEX	CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERINDEXLEADCHAR	CAP_PRINTERSUFFIX
CAP_PRINTERINDEXMAXVALUE	CAP_PRINTERVERTICALOFFSET

CAP_PRINTERINDEXSTEP

Description

Set the counter increment for the current CAP_PRINTER device to any positive value. Base value of the counter is specified by CAP_PRINTERINDEX. Counter will increment with CAP_PRINTERINDEXSTEP every time when event specified in CAP_PRINTERINDEXTRIGGER occurs.

Values

Type:	TW_UINT32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	1
Allowed Values:	Any positive value

Containers

MSG_GET	TW_ONEVALUE, TW_ENUMERATION, TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE, TW_ENUMERATION, TW_RANGE
MSG_SETCONSTRAINT	TW_ONEVALUE, TW_ENUMERATION, TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

CAP_PRINTER	CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERCHARROTATION	CAP_PRINTERINDEXTRIGGER
CAP_PRINTERENABLED	CAP_PRINTERMODE
CAP_PRINTERFONTSTYLE	CAP_PRINTERSTRING
CAP_PRINTERINDEX	CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERINDEXLEADCHAR	CAP_PRINTERSUFFIX
CAP_PRINTERINDEXMAXVALUE	CAP_PRINTERVERTICALOFFSET

CAP_PRINTERINDEXTRIGGER

Description

Specify the events which cause the printer's counter to increment its value. If no trigger is specified (the array is empty) then the counter never increments.

TWCT_PAGE should be used alone. The other values can be mixed in any combination.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TWCT_PAGE
Allowed Values:	TWCT_PAGE TWCT_PATCH1 TWCT_PATCH2 TWCT_PATCH3 TWCT_PATCH4 TWCT_PATCHT TWCT_PATCH6

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	TW_ARRAY
MSG_SETCONSTRAINT	TW_ARRAY
MSG_RESET	TW_ARRAY
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

CAP_PRINTER
CAP_PRINTERCHARROTATION
CAP_PRINTERENABLED
CAP_PRINTERFONTSTYLE
CAP_PRINTERINDEX
CAP_PRINTERINDEXLEADCHAR
CAP_PRINTERINDEXMAXVALUE
CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERINDEXSTEP
CAP_PRINTERMODE
CAP_PRINTERSTRING
CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERSUFFIX
CAP_PRINTERVERTICALOFFSET

CAP_PRINTERMODE

Description

Based on the value of this capability, printed text will consist of:

- TWPM_SINGLESTRING a single line consisting of the value of CAP_PRINTERSTRING.
- TWPM_MULTISTRING all the strings in the value of CAP_PRINTERSTRING, printed one per line, in order.
- TWPM_COMPOUNDSTRING the value of CAP_PRINTERSTRING followed by the printer's current counter value, followed by the value of CAP_PRINTERSUFFIX.

Application

Negotiate this capability to specify the mode of printing to use when the current CAP_PRINTER device is enabled.

Source

If supported, use the specified mode for future image acquisitions.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWPM_SINGLESTRING TWPM_MULTISTRING TWPM_COMPOUNDSTRING

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

When a Source supports CAP_PRINTERSTRING and / or CAP_PRINTERSUFFIX.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER	CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERCHARROTATION	CAP_PRINTERINDEXSTEP
CAP_PRINTERENABLED	CAP_PRINTERINDEXTRIGGER
CAP_PRINTERFONTSTYLE	CAP_PRINTERSTRING
CAP_PRINTERINDEX	CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERINDEXLEADCHAR	CAP_PRINTERSUFFIX
CAP_PRINTERINDEXMAXVALUE	CAP_PRINTERVERTICALOFFSET

CAP_PRINTERSTRING

Description

Specifies the string(s) that are to be used in the string component when the current CAP_PRINTER device is enabled.

Application

Negotiate this capability to specify the string or strings to be used for printing (depending on printer mode). Use enumeration to print multiple lines of text, one line per string in the enumerated list. Be sure to check the status codes if attempting multiple lines, since not all devices support this feature.

Source

If supported, use the specified string for printing during future acquisitions.

Values

Type:	TW_STR255
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any string

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE TW_ENUMERATION
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER
CAP_PRINTERCHARROTATION
CAP_PRINTERENABLED
CAP_PRINTERFONTSTYLE
CAP_PRINTERINDEX
CAP_PRINTERINDEXLEADCHAR
CAP_PRINTERINDEXMAXVALUE
CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERINDEXSTEP
CAP_PRINTERINDEXTRIGGER
CAP_PRINTERMODE
CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERSUFFIX
CAP_PRINTERVERTICALOFFSET

CAP_PRINTERSTRINGPREVIEW

Description

Return the text that would be printed next, by the imprinter designated by `CAP_PRINTER`, based on current printer/endorser settings.

The complete text is an N-element `TW_ARRAY` with each element containing the text of the corresponding line.

This is a read only capability.

Source

This is a simulation of the printer text string that the source provides to its best ability.

Values

Type:	`TW_STR255`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any string

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`Not allowed`
MSG_SETCONSTRAINT	`Not allowed`
MSG_RESET	`Not allowed`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.3

See Also

Best Practices

`CAP_PRINTER`
`CAP_PRINTERCHARROTATION`
`CAP_PRINTERENABLED`
`CAP_PRINTERFONTSTYLE`
`CAP_PRINTERINDEX`
`CAP_PRINTERINDEXLEADCHAR`
`CAP_PRINTERINDEXMAXVALUE`
`CAP_PRINTERINDEXNUMDIGITS`
`CAP_PRINTERINDEXSTEP`
`CAP_PRINTERINDEXTRIGGER`
`CAP_PRINTERMODE`
`CAP_PRINTERSTRING`
`CAP_PRINTERSUFFIX`
`CAP_PRINTERVERTICALOFFSET`

CAP_PRINTERSUFFIX

Description

Specifies the string that shall be used as the current CAP_PRINTER device's suffix.

Application

Negotiate this capability to specify the string that is used as the suffix for printing if TWPM_COMPOUNDSTRING is used.

Values

Type:	TW_STR255
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any string

Containers

MSG_GET	TW_ONEVALUE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_PRINTER
CAP_PRINTERCHARROTATION
CAP_PRINTERENABLED
CAP_PRINTERFONTSTYLE
CAP_PRINTERINDEX
CAP_PRINTERINDEXLEADCHAR
CAP_PRINTERINDEXMAXVALUE
CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERINDEXSTEP
CAP_PRINTERINDEXTRIGGER
CAP_PRINTERMODE
CAP_PRINTERSTRING
CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERVERTICALOFFSET

CAP_PRINTERVERTICALOFFSET

Description

Specifies a Y-Offset in ICAP_UNITS units for the current CAP_PRINTER device.

Source

This allows the user to set Y-Offset for the current CAP_PRINTER device. Some scanners may not support a 0 offset.

Values

Type:	TW_FIX32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any value

Containers

MSG_GET	TW_ONEVALUE TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

CAP_PRINTER
CAP_PRINTERCHARROTATION
CAP_PRINTERENABLED
CAP_PRINTERFONTSTYLE
CAP_PRINTERINDEX
CAP_PRINTERINDEXLEADCHAR
CAP_PRINTERINDEXMAXVALUE
CAP_PRINTERINDEXNUMDIGITS
CAP_PRINTERINDEXSTEP
CAP_PRINTERINDEXTRIGGER
CAP_PRINTERMODE
CAP_PRINTERSTRING
CAP_PRINTERSTRINGPREVIEW
CAP_PRINTERSUFFIX

CAP_REACQUIREALLOWED

Description

Indicates whether the physical hardware (e.g. scanner, digital camera) is capable of acquiring multiple images of the same page without changes to the physical registration of that page.

This is a read only capability.

Application

Use this capability to enable or disable modes of operation where multiple image acquisitions of the page are required. Examples: preview mode, automated image analysis mode.

Source

If supported, return `TRUE` if the device is capable of capturing the page image multiple times without refeeding the page or otherwise causing physical registration changes. Return `FALSE` otherwise.

Support Guidelines for Sources

- A flat bed scanner that can retain the page on the platen and moves the scan bar past the page would return `TRUE`.
- A sheet-fed scanner that physically moves the page past the scan bar would return `FALSE`.
- A hand held scanner would return `FALSE`.

Values

Type:	`TW_BOOL`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION` **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_AUTOFEED`
`CAP_CLEARPAGE`
`CAP_FEEDERENABLED`
`CAP_FEEDPAGE`
`CAP_REWINDPAGE`

CAP_REWINDPAGE

Description

If TRUE, the Source will return the current page to the input side of the document feeder and feed the last page from the output side of the feeder back into the acquisition area.

If CAP_AUTOFEED is TRUE, automatic feeding will continue after all negotiated frames from this page are acquired.

CAP_FEEDERENABLED must equal TRUE to use this capability.

This capability must have been negotiated as an extended capability to be used in States 5 and 6.

Application

This capability is used in States 5 and 6 by applications controlling the Source's feeder (usually without the Source's user interface).

If CAP_AUTOFEED is TRUE, the normal automatic feeding will continue after all frames of this page are acquired.

Source

If CAP_FEEDERENABLED equals FALSE, return TWRC_FAILURE / TWCC_CAPSEQERROR (capability is not supported in current settings).

If there are no documents in the output area, return: TWRC_FAILURE / TWCC_BADVALUE.

The Source will perform this action once whenever the capability is MSG_SET to TRUE. The Source should then revert the Current value to FALSE.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	FALSE
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE	
	TW_ENUMERATION	**// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE	
	TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

CAP_AUTOFEED	CAP_FEEDERENABLED
CAP_CLEARPAGE	CAP_FEEDERLOADED
CAP_EXTENDEDCAPS	CAP_FEEDPAGE

CAP_SEGMENTED

Description

Describes the segmentation setting for captured images. Image segmentation occurs when either the device, the Source or the Application breaks up an image into a stream of image components (text, pictures, graphics) that must be assembled by the application to reconstruct the original document. Applications must use the `DAT_EXTIMAGEINFO / TWEI_SEGMENTNUMBER` field to identify pieces of an image that are associated with each other through segmentation.

Manual segmentation is an advanced way of setting the Source to transfer image components with different parameters from the same page. Not all Sources support this feature.

Application

Applications should be able to `GET/SET` whether segmentation will be applied to captured images.

If the Application sets Manual segmentation, it can specify different settings for every frame in `ICAP_FRAMES`. In this mode `ICAP_FRAMES` states before `ICAP_COLORMANAGMENTENABLED` in capability ordering. The Application can set the current frame by setting `ICAP_FRAMES` using the `TW_ENUMERATION` container and changing CurrentIndex only. The number of Frames and their parameters must be kept unchanged during this process, otherwise a Source will set all capabilities the same for all frames.

A Source may not support separate settings for all supported capabilities for different frames. Get the list of these capabilities from `CAP_SUPPORTEDCAPSSEGMENTUNIQUE`. Applications must set all common capabilities before it sets the Manual Segmentation mode, and then set only capabilities which are different for different frames.

Source

If the Source is in Manual segmentation, and the Application changed the number of frames and their parameters, then the Source will make all capabilities the same for all frames using values for the current frame (prior this operation). In this mode the Source must ignore the capability order for `ICAP_FRAMES`. It may allow different settings for capabilities which are placed below `ICAP_MINIMUMWIDTH` in the capabilities order. The Source must report these capabilities in `CAP_SUPPORTEDCAPSSEGMENTUNIQUE`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWSG_NONE`

Allowed Values:	`TWSG_AUTO`
	`TWSG_NONE`
	`TWSG_MANUAL`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.91

See Also

Best Practices

ICAP_FRAMES
ICAP_MAXFRAMES
CAP_SUPPORTEDCAPSSEGMENTUNIQUE
DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET (TWEI_SEGMENTNUMBER)
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT
DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET
TW_IMAGELAYOUT

CAP_SERIALNUMBER

Description

A string containing the serial number of the currently selected device in the Source. Multiple devices may all report the same serial number.

This is a read only capability.

Application

The value is device specific, Applications should not attempt to parse the information.

Values

Type:	`TW_STR255`
Allowed Values:	Any value

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

CAP_SHEETCOUNT

Description

Capture the specified number of sheets of paper. A single sheet of paper may result in zero or more images, depending on various settings in the driver (i.e. CAP_DUPLEXENABLED, ICAP_AUTODISCARDBLANKPAGES, etc.)This is a read only capability.

Application

An application must set CAP_XFERMECH to -1 before using this capability.

A CAP_SHEETCOUNT value of 0 requests that all of the available sheets are captured.
A value of 1 or greater requests that only that number of sheets are captured.

Use DAT_EXTIMAGEINFO to determine which sheet and which side of a sheet an image came from

Source

If CAP_XFERCOUNT is not equal to -1, then CAP_SHEETCOUNT is ignored.

If the value of CAP_SHEETCOUNT is equal to 0, then all of the available sheets are captured. For values greater than 0, the scanner captures the specified number of sheets. If the scanner runs out of sheets before the CAP_SHEETCOUNT value is reached, then the scanner automatically transitions to state 5.

Values

Type:	TW_UINT32
Value after MSG_OPENDS:	0
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	0 – (2^32)-1

Containers

MSG_GET	TW_ONEVALUE TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE TW_RANGE
MSG_SETCONSTRAINT	TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.4

See Also

Best Practices

CAP_XFERCOUNT

TW_PENDINGXFERS.Count

CAP_SUPPORTEDCAPS

Description

Returns a list of all the capabilities for which the Source will answer inquiries. Does not indicate which capabilities the Source will allow to be set by the application. Some capabilities can only be set if certain setup work has been done so the Source cannot globally answer which capabilities are "set-able."

This is a read only capability.

Values

Type:	`TW_UINT16`
Allowed Values:	Any "get-able" capability

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_EXTENDEDCAPS`
`CAP_SUPPORTEDDATS`

CAP_SUPPORTEDCAPSSEGMENTUNIQUE

Description

For Sources that allow unique values to be set for things like the top and bottom or for each segment on a page.

Returns a list of all the capabilities for which the Source allows to have unique values.

This is a read only capability.

Values

Type:	`TW_UINT16`
Allowed Values:	Any “get-able” capability

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

2.2 or greater sources that allow different settings for camera or segment.

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`CAP_SUPPORTEDCAPS`
`CAP_SEGMENTED`

CAP_SUPPORTEDDATS

Description

Returns a list of all the Data Argument Types (`DAT_xxx`) for which the Source will answer inquiries. This list does not indicate which DATs the Source will allow to be set by the application. Some DATs can only be set if certain setup work has been done so the Source cannot globally answer which DATs are "set-able."

`HIWORD` of the value is `DG` of `DAT`. `LOWORD` of the value is the `DAT` itself.

This is a read only capability.

Application

`MSG_GET` a quick way to determine if a `DAT` is supported by the Source.

Values

Type:	`TW_UINT32`
Allowed Values:	All standard and custom `DAT_xxx` understood by the Source

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All 2.2 Sources

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`CAP_SUPPORTEDCAPS`

CAP_TIMEBEFOREFIRSTCAPTURE

Description

For automatic capture, this value selects the number of milliseconds before the first picture is to be taken, or the first image is to be scanned.

Values

Type:	`TW_INT32`
Value after MSG_OPENDS:	`0`
After MSG_RESET/MSG_RESETALL:	`0`
Allowed Values:	0 or greater

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_AUTOMATICCAPTURE`
`CAP_TIMEBETWEENCAPTURES`
`CAP_XFERCOUNT`

CAP_TIMEBETWEENCAPTURES

Description

For automatic capture, this value selects the milliseconds to wait between pictures taken, or images scanned.

Values

Type:	`TW_INT32`
Value after MSG_OPENDS:	`0`
After MSG_RESET/MSG_RESETALL:	`0`
Allowed Values:	0 or greater

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`CAP_AUTOMATICCAPTURE`
`CAP_TIMEBEFOREFIRSTCAPTURE`
`CAP_XFERCOUNT`

CAP_TIMEDATE

Description

The date and time the image was acquired.

Note: CAP_TIMEDATE does not return the *exact* time the image was acquired; rather, it returns the *closest available approximation* of the time the physical phenomena represented by the image was recorded. If the application needs the exact time of acquisition, the application should generate that value itself during the image acquisition procedure.

Stored in the form "YYYY/MM/DD HH:mm:SS.sss" where YYYY is the year, MM is the numerical month, DD is the numerical day, HH is the hour, mm is the minute, SS is the second, and sss is the millisecond.

This capability must be negotiated during State 7 before the call to the DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER triplet. It must also be listed in the CAP_EXTENDEDCAPS capability by the data source.

This is a read only capability.

Source

The time and date when the image was originally acquired (when the Source entered State 7).

Be sure to leave the space between the ending of the date and beginning of the time fields. Pad the unused characters after the string with zeros.

Values

Type:	TW_STR32
Allowed Values:	Any date

Containers

MSG_GET	TW_ONEVALUE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

CAP_AUTHOR
CAP_CAPTION

CAP_THUMBNAILSENABLED

Description

Allows an application to request the delivery of thumbnail representations for the set of images that are to be delivered.

Setting `CAP_THUMBNAILSENABLED` to `TRUE` turns on thumbnail mode. Images transferred thereafter will be sent at thumbnail size (exact thumbnail size is determined by the Data Source). Setting this capability to `FALSE` turns thumbnail mode off and returns full size images.

Application

A successful set of this capability to `TRUE` will cause the Source to deliver image thumbnails during normal data transfer operations. This mode remains in effect until this capability is set back to `FALSE`.

Source

A successful set of this capability to `TRUE` should enable the delivery of thumbnail images during normal data transfer. Setting this capability to `FALSE` will disable thumbnail delivery.

If not supported, return `TWRC_FAILURE/ TWCC_CAPUNSUPPORTED`.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	`FALSE`
After MSG_RESET/MSG_RESETALL:	`FALSE`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

All Image Store Data Sources.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`ICAP_IMAGEDATASET`

CAP_UICONTROLLABLE

Description

If TRUE, indicates that this Source supports acquisition with the UI disabled; i.e., TW_USERINTERFACE's ShowUI field can be set to FALSE. If FALSE, indicates that this Source can only support acquisition with the UI enabled.

This is a read only capability.

Source

This capability was introduced in TWAIN 1.6. All Sources compliant with TWAIN 1.6 and above must support this capability. Sources that are not TWAIN 1.6-compliant may return TWRC_FAILURE / TWCC_BADCAP if they do not support this capability.

All Sources compliant with TWAIN 1.9 and above must support the ability to scan without the UI (TW_USERINTERFACE.ShowUI = 0 and CAP_INDICATORS = FALSE), therefore they must report a value of TRUE for this capability.

Application

A return value of TWRC_FAILURE / TWCC_CAPUNSUPPORTED indicates that the Source in use is not TWAIN 1.6-compliant. Therefore, the Source may ignore TW_USERINTERFACE's ShowUI field when MSG_ENABLEDS is issued. See the description of DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS for more details.

Values

Type:	TW_BOOL
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Sources **// 1.9 and higher**

TWAIN Version Introduced

Version 1.6

See Also

Best Practices

CAP_INDICATORS
DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS

CAP_XFERCOUNT

Description

The application is willing to accept this number of images.

Application

Set this capability to the number of images you are willing to transfer per session. Common values are:

1	Application wishes to transfer only one image this session
-1	Application is willing to transfer multiple images

Source

If the application limits the number of images it is willing to receive, the Source should not make more transfers available than the specified number.

If the application sets the value to 0, then the Source sets its value to -1 and returns back `TWRC_FAILURE / TWCC_CHECKSTATUS`.

Values

Type:	`TW_INT16`
Value after MSG_OPENDS:	-1
After MSG_RESET/MSG_RESETALL:	-1
Allowed Values:	-1 and 1 – 32767

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_RANGE`	**// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

All Sources and applications

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_SHEETCOUNT`

`TW_PENDINGXFERS.Count`

ICAP_AUTOBRIGHT

Description

TRUE enables and FALSE disables the Source's Auto-brightness function (if any).

Source

If TRUE, apply auto-brightness function to acquired image before transfer.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_BRIGHTNESS

ICAP_AUTODISCARDBLANKPAGES

Description

Use this capability to have the `Source` discard blank images. The `Application` never sees these images during the scanning session.

`TWBP_DISABLE` – this must be the default state for the Source. It indicates that all images will be delivered to the `Application`, none of them will be discarded.

`TWBP_AUTO` – if this is used, then the Source will decide if an image is blank or not and discard as appropriate.

If the specified value is a positive number in the range 0 to 2^{31}–1, then this capability will use it as the byte size cutoff point to identify which images are to be discarded. If the size of the image is less than or equal to this value, then it will be discarded. If the size of the image is greater than this value, then it will be kept so that it can be transferred to the Application.

Values

Type:	`TW_INT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWBP_DISABLE`

Allowed Values:	`TWBP_DISABLE`
	`TWBP_AUTO`
	Byte count 0 to 2^{31}-1

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET`

ICAP_AUTOMATICBORDERDETECTION

Description

Turns automatic border detection on and off.

Application

Negotiate this capability to determine the state of the AutoBorder detection.

ICAP_UNDEFINEDIMAGESIZE must be enabled for this feature to work.

Source

If supported, enable or disable automatic border detection according to the value specified. Default to FALSE for backward compatibility. For this capability to be enabled, ICAP_UNDEFINEDIMAGESIZE must be enabled.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TRUE if DAT_IMAGEINFO values in State 6 are identical to the DAT_IMAGEINFO values after TWRC_XFERDONE, otherwise FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_UNDEFINEDIMAGESIZE
ICAP_AUTOMATICDESKEW
ICAP_AUTOSIZE

ICAP_AUTOMATICCOLORENABLED

Description

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

Application

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

Source

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

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE, FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

ICAP_PIXELTYPE
ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE

ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE

Description

Specifies the non-color pixel type to use when automatic color is enabled.

Application

`ICAP_AUTOMATICCOLORENABLED` must be `TRUE`. When it is, the `Application` sets this capability to specify the pixel type the `Source` uses when transferring non-color images.

Source

When `ICAP_AUTOMATICCOLORENABLED` is `TRUE`, this capability determines the pixel type of the non-color images.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWPT_BW`
Allowed Values:	`TWPT_BW` `TWPT_GRAY`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

`ICAP_PIXELTYPE`
`ICAP_AUTOMATICCOLORENABLED`

ICAP_AUTOMATICCROPUSESFRAME

Description

Set TRUE if DAT_IMAGELAYOUT, ICAP_SUPPORTEDSIZES or ICAP_FRAMES reduces the amount of data captured from the device, potentially improving the performance of the driver, even if any automatic detection capability like ICAP_AUTOMATICBORDERDECTION is set TRUE.

This is a read only capability.

Application

If this capability reports TRUE then the Application may assume a performance benefit from specifying a cropping frame using DAT_IMAGELAYOUT, ICAP_SUPPORTEDSIZES or ICAP_FRAMES.

The Application sets the frame after turning on any automated capabilities. For instance, if the Application wants automatic border detection, but knows that the largest paper size it will receive is US Letter, then it sets ICAP_AUTOMATICBORDERDETECTION to TRUE and then sets ICAP_SUPPORTEDSIZES to TWSS_USLETTER.

Source

The Source reports TRUE if it uses the cropping frame specified by DAT_IMAGELAYOUT, ICAP_SUPPORTEDSIZES or ICAP_FRAMES to reduce the amount of data physically transferred from the device to the Source.

The Source is not obligated to exactly match the frame requested by the Application, but it should use it as a hint to improve the performance of the capture.

Values

Type:	TW_BOOL
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

```
ICAP_AUTOMATICBORDERDETECTION
ICAP_FRAMES
ICAP_SUPPORTEDSIZES
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT
```

ICAP_AUTOMATICDESKEW

Description

Turns automatic deskew correction on and off.

Application

Negotiate this capability to enable or disable Automatic deskew.

Source

If supported, enable or disable the Automatic deskew feature according to the value specified for future transfers. Default to FALSE for backward compatibility. Some Sources may require ICAP_UNDEFINEDIMAGESIZE to be enabled.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TRUE if DAT_IMAGEINFO values in State 6 are identical to the DAT_IMAGEINFO values after TWRC_XFERDONE, otherwise FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_AUTOMATICBORDERDETECTION
ICAP_AUTOMATICROTATE
ICAP_AUTOSIZE
ICAP_UNDEFINEDIMAGESIZE

ICAP_AUTOMATICLENGTHDETECTION

Description

Controls the automatic detection of the length of a document, this is intended for use with an Automatic Document Feeder.

Application

If this capability is present, but does not support TWQC_SET when MSG_QUERYSUPPORT is called, then it indicates the fixed behavior of the Source (always TRUE or always FALSE).

If this capability reports TWQC_SET, then the Application can control the automatic detection of the length of a document.

If ICAP_AUTOMATICBORDERDETECTION (which detects width and length) is set to TRUE, then this capability is ignored.

Source

If set to TRUE, the Source automatically crops the height of the image to the length of the document.

If set to FALSE (and assuming ICAP_AUTOMATICBORDERDETECTION is FALSE), the Source returns the full height specified by ICAP_FRAME or DAT_IMAGELAYOUT, regardless of the actual height of the captured document (for instance, a check in an A4 size area).

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TRUE if DAT_IMAGEINFO values in State 6 are identical to the DAT_IMAGEINFO values after TWRC_XFERDONE, otherwise FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

`ICAP_AUTOMATICBORDERDETECTION`
`ICAP_FRAMES`
`ICAP_SUPPORTEDSIZES`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET`
`DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET`

ICAP_AUTOMATICROTATE

Description

When `TRUE` this capability depends on intelligent features within the Source to automatically rotate the image to the correct position.

Application

If this capability is set to `TRUE`, then it must be assumed that no other correction is required (deskew, rotation, etc…); the Source is guaranteeing that it will deliver images in the correct orientation.

Source

There are no criteria for how this automatic rotation is determined. A Source may use a field of text, or some distinguishing non-text field, such as a barcode or a logo, or it may rely on form recognition to help rotate the document.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`FALSE`
Allowed Values:	`TRUE` or `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	 **// 2.0 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_AUTOMATICDESKEW`
`ICAP_ORIENTATION`
`ICAP_ROTATION`

ICAP_AUTOSIZE

Description

Force the output image dimensions to match either the current value of `ICAP_SUPPORTEDSIZES` or any of its current allowed values.

Source

This capability takes precedence over `CAP_AUTOMATICBORDERDETECTION` and `ICAP_AUTOMATICLENGTHDETECTION`.

Source will adjust dimensions of the images to exactly match the current value of `ICAP_SUPPORTEDSIZES` if this capability is set to `TWAS_CURRENT`.

Source will adjust dimensions of the images to exactly match one of the values of `ICAP_SUPPORTEDSIZES` if this capability is set to `TWAS_AUTO`.

If set to `TWAS_NONE`, then no action is taken.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWAS_NONE`

Allowed Values:	`TWAS_NONE`
	`TWAS_AUTO`
	`TWAS_CURRENT`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.0

See Also

Best Practices

`ICAP_AUTOMATICBORDERDETECTION`
`ICAP_AUTOMATICDESKEW`
`ICAP_ORIENTATION`
`ICAP_ROTATION`
`ICAP_SUPPORTEDSIZES`

ICAP_BARCODEDETECTIONENABLED

Description

Turns bar code detection on and off.

Source

Support this capability if the scanner supports any Bar code recognition. If the device allows this feature to be turned off, then default to off. If the device does not support disabling this feature, report TRUE and disallow attempts to set FALSE.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_SUPPORTEDBARCODETYPES
ICAP_SUPPORTEDPATCHCODETYPES
ICAP_BARCODEMAXSEARCHPRIORITIES
ICAP_BARCODESEARCHPRIORITIES
ICAP_BARCODESEARCHMODE
ICAP_BARCODEMAXRETRIES
ICAP_BARCODETIMEOUT

ICAP_BARCODEMAXRETRIES

Description

Restricts the number of times a search will be retried if none are found on each page.

Application

Refine this capability to limit the number of times the bar code search algorithm is retried on a page that contains no bar codes.

Source

If supported, limit the number of retries the value specified.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_BARCODEDETECTIONENABLED`
`ICAP_SUPPORTEDBARCODETYPES`
`ICAP_SUPPORTEDPATCHCODETYPES`
`ICAP_BARCODEMAXSEARCHPRIORITIES`
`ICAP_BARCODESEARCHPRIORITIES`
`ICAP_BARCODESEARCHMODE`
`ICAP_BARCODETIMEOUT`

ICAP_BARCODEMAXSEARCHPRIORITIES

Description

The maximum number of supported search priorities.

Application

Query this value to determine how many bar code detection priorities can be set.

Set this value to limit the number of priorities to speed the detection process.

Source

If bar code searches can be prioritized, report the maximum number of priorities allowed for a search.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_BARCODEDETECTIONENABLED
ICAP_SUPPORTEDBARCODETYPES
ICAP_SUPPORTEDPATCHCODETYPES
ICAP_BARCODESEARCHPRIORITIES
ICAP_BARCODESEARCHMODE
ICAP_BARCODEMAXRETRIES
ICAP_BARCODETIMEOUT

ICAP_BARCODESEARCHMODE

Description

Restricts bar code searching to certain orientations, or prioritizes one orientation over the other.

Application

Negotiate this capability if the orientation of bar codes is already known to the application. Refinement of this capability can speed the bar code search.

Source

If set then apply the specified refinements to future bar code searches.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWBD_HORZ`
	`TWBD_VERT`
	`TWBD_HORZVERT`
	`TWBD_VERTHORZ`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_BARCODEDETECTIONENABLED
ICAP_SUPPORTEDBARCODETYPES
ICAP_SUPPORTEDPATCHCODETYPES
ICAP_BARCODEMAXSEARCHPRIORITIES
ICAP_BARCODESEARCHPRIORITIES
ICAP_BARCODEMAXRETRIES
ICAP_BARCODETIMEOUT

ICAP_BARCODESEARCHPRIORITIES

Description

A prioritized list of bar code types dictating the order in which bar codes will be sought.

Application

Set this capability to specify the order and priority for bar code searching. Refining the priorities to only the bar code types of interest to the application can speed the search process.

Source

If this type of search refinement is supported, then report the current values.

If set, then limit future searches to the specified bar codes in the specified priority order.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWBT_2OF5DATALOGIC`
	`TWBT_2OF5IATA`
	`TWBT_2OF5INDUSTRIAL`
	`TWBT_2OF5INTERLEAVED`
	`TWBT_2OF5MATRIX`
	`TWBT_2OF5NONINTERLEAVED`
	`TWBT_3OF9`
	`TWBT_3OF9FULLASCII`
	`TWBT_CODABAR`
	`TWBT_CODABARWITHSTARTSTOP`
	`TWBT_CODE128`
	`TWBT_CODE93`
	`TWBT_EAN13`
	`TWBT_EAN8`
	`TWBT_MAXICODE`
	`TWBT_PDF417`
	`TWBT_POSTNET`
	`TWBT_QRCODE`
	`TWBT_UCC128`
	`TWBT_UPCA`
	`TWBT_UPCE`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`

MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_BARCODEDETECTIONENABLED	ICAP_BARCODESEARCHMODE
ICAP_SUPPORTEDBARCODETYPES	ICAP_BARCODEMAXRETRIES
ICAP_SUPPORTEDPATCHCODETYPES	ICAP_BARCODETIMEOUT
ICAP_BARCODEMAXSEARCHPRIORITIES	

ICAP_BARCODETIMEOUT

Description

Restricts the total time spent on searching for a bar code on each page.

Application

Refine this value to tune the length of time the search algorithm is allowed to execute before giving up.

Source

If supported, limit the duration of a bar code search to the value specified.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_BARCODEDETECTIONENABLED`
`ICAP_SUPPORTEDBARCODETYPES`
`ICAP_SUPPORTEDPATCHCODETYPES`
`ICAP_BARCODEMAXSEARCHPRIORITIES`
`ICAP_BARCODESEARCHPRIORITIES`
`ICAP_BARCODESEARCHMODE`
`ICAP_BARCODEMAXRETRIES`

ICAP_BITDEPTH

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.

Application

The application should loop through all the `ICAP_PIXELTYPE`s it is interested in and negotiate the `ICAP_BITDEPTH`(s) for each.

For all allowed settings of `ICAP_PIXELTYPE`

- Set `ICAP_PIXELTYPE`
- Set `ICAP_BITDEPTH` for the current `ICAP_PIXELTYPE`

Source

If the bit depth in a `MSG_SET` is not supported for the current `ICAP_PIXELTYPE` setting, return `TWRC_FAILURE` / `TWCC_BADVALUE`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(dependent on `ICAP_PIXELTYPE`)
Allowed Values:	>=1

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_PIXELTYPE`

ICAP_BITDEPTHREDUCTION

Description

Specifies the Reduction Method the Source should use to reduce the bit depth of the data. Most commonly used with ICAP_PIXELTYPE = TWPT_BW to reduce gray data to black and white.

Application

Set the capability to the reduction method to be used in future acquisitions

Also select the Halftone or Threshold to be used.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWBR_THRESHOLD TWBR_HALFTONE TWBR_CUSTHALFTONE TWBR_DIFFUSION TWBR_DYNAMICTHRESHOLD

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Sources when ICAP_PIXELTYPE is TWPT_BW.

TWAIN Version Introduced

Version 1.5

See Also

Best Practices

ICAP_CUSTHALFTONE
ICAP_HALFTONES
ICAP_PIXELTYPE
ICAP_THRESHOLD

ICAP_BITORDER

Description

Specifies how the bytes in an image are filled by the Source. `TWBO_MSBFIRST` indicates that the leftmost bit in the byte (usually bit 7) is the byte's Most Significant Bit.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	`TWBO_MSBFIRST`
After MSG_RESET/MSG_RESETALL:	`TWBO_MSBFIRST`
Allowed Values:	`TWBO_LSBFIRST` `TWBO_MSBFIRST`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_BITORDERCODES`

ICAP_BITORDERCODES

Description

Used for CCITT data compression only. Indicates the bit order representation of the stored compressed codes.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	`TWBO_LSBFIRST`
After MSG_RESET/MSG_RESETALL:	`TWBO_LSBFIRST`
Allowed Values:	`TWBO_LSBFIRST` `TWBO_MSBFIRST`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_COMPRESSION`

ICAP_BRIGHTNESS

Description

The brightness values available within the Source.

Application

The application can use this capability to inquire, set, or restrict the values for `BRIGHTNESS` used in the Source.

Source

Source should normalize the values into the range. Make sure that a '0' value is available as the Current Value when the Source starts up. If the Source's ± range is asymmetric about the '0' value, set range maxima to ±1000 and scale homogeneously from the '0' value in each direction. This will yield a positive range whose step size differs from the negative range's step size.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	-1000 to +1000

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_AUTOBRIGHT`
`ICAP_CONTRAST`

ICAP_CCITTKFACTOR

Description

Used for CCITT Group 3 2-dimensional compression. The 'K' factor indicates how often the new compression baseline should be re-established. A value of 2 or 4 is common in facsimile communication. A value of zero in this field will indicate an infinite K factor—the baseline is only calculated at the beginning of the transfer.

Values:

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	4
Allowed Values:	0 to 2^{16}

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_RANGE`	**// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

When the `ICAP_COMPRESSION` value is `TWCP_GROUP32D`.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_COMPRESSION`

ICAP_COLORMANAGEMENTENABLED

Description

Disables the Source's color and gamma tables for color and grayscale images, resulting in output that could be termed "raw".

Application

When the Application sets this capability to `FALSE`, it takes responsibility for profiling the color and grayscale output of the device, and applying the desired color and gamma corrections itself. The Application is completely responsible for the quality of the finished image.

Source

When this capability is `FALSE` the `Source` turns off as much of its color and gamma correction as it can. There is no universal standard for this behavior, so it makes its best effort.

It is recommended that the Source not expose this capability unless it can do a credible job of outputting "raw" image data.

Values

Type:	`TW_BOOL`
Value after MSG_OPENDS:	`TRUE`
After MSG_RESET/MSG_RESETALL:	`TRUE`
Allowed Values:	`TRUE`, `FALSE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

`ICAP_PIXELTYPE`

ICAP_COMPRESSION

Description

Allows the application and Source to identify which compression schemes they have in common for Buffered Memory and File transfers.

Note for File transfers:

Since only certain file formats support compression, this capability must be negotiated after setting the desired file format with `ICAP_IMAGEFILEFORMAT`.

`TWCP_NONE`	All Sources must support this.
`TWCP_PACKBITS`	Can be used with TIFF or PICT
`TWCP_GROUP31D,`	
`TWCP_GROUP31DEOL,`	
`TWCP_GROUP32D,`	
`TWCP_GROUP4`	Are all from the CCITT specification (now ITU), intended for document images (can be used with TIFF).
`TWCP_JPEG`	Intended for the compression of color photographs (can be used with TIFF, JFIF or SPIFF).
`TWCP_LZW`	A compression licensed by UNISYS (can be used with TIFF).
`TWCP_JBIG`	Intended for bitonal and grayscale document images (can be used with TIFF or SPIFF).
`TWCP_PNG`	This compression can only be used if `ICAP_IMAGEFILEFORMAT` is set to `TWFF_PNG`.
`TWCP_RLE4,`	
`TWCP_RLE8,`	
`TWCP_BITFIELDS`	These compressions can only be used if `ICAP_IMAGEFILEFORMAT` is set to `TWFF_BMP`.
`TWCP_ZIP`	Per RFC 1951 (AKA 'Flate' and 'Deflate')
`TWCP_JPEG2000`	Per ISO/IEC 15444

Application

Applications must not assume that a Source can provide compressed Buffered Memory or File transfers, because many cannot. The application should use `MSG_SET` on a `TW_ONEVALUE` container to specify the compression type for future transfers.

Source

The current value of this setting specifies the compression method to be used in future transfers. If the image transfer mechanism is changed, then the allowed list must be modified to reflect the supported values. If the current value is not available on the new allowed list, then the Source must change it to its preferred value.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)

After MSG_RESET/MSG_RESETALL:	TWCP_NONE
Allowed Values:	TWCP_NONE TWCP_PACKBITS TWCP_GROUP31D TWCP_GROUP31DEOL TWCP_GROUP32D TWCP_GROUP4 TWCP_JPEG TWCP_LZW TWCP_JBIG TWCP_PNG TWCP_RLE4 TWCP_RLE8 TWCP_BITFIELDS TWCP_ZIP TWCP_JPEG2000

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

CAP_XFERCOUNT
ICAP_IMAGEFILEFORMAT
ICAP_JPEGQUALITY
ICAP_JPEGSUBSAMPLING

DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET
DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET

ICAP_CONTRAST

Description

The contrast values available within the Source.

Application

The application can use this capability to inquire, set or restrict the values for CONTRAST used in the Source.

Source

Scale the values available internally into a homogeneous range between -1000 and 1000. Make sure that a '0' value is available as the Current value when the Source starts up. If the Source's ± range is asymmetric about the '0' value, set range maxima to ±1000 and scale homogeneously from the '0' value in each direction. This will yield a positive range whose step size differs from the negative range's step size.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	-1000 to +1000

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_BRIGHTNESS`

ICAP_CUSTHALFTONE

Description

Specifies the square-cell halftone (dithering) matrix the Source should use to halftone the image.

Application

The application should also set `ICAP_BITDEPTHREDUCTION` to `TWBR_CUSTHALFTONE` to use this capability.

Values

Type:	`TW_UINT8`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any rectangular array

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

When the `ICAP_BITDEPTHREDUCTION` value is `TWBR_CUSTHALFTONE`.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_BITDEPTHREDUCTION`

ICAP_EXPOSURETIME

Description

Specifies the exposure time used to capture the image, in seconds.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	>0

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_FLASHUSED2`
`ICAP_LAMPSTATE`
`ICAP_LIGHTPATH`
`ICAP_LIGHTSOURCE`

ICAP_EXTIMAGEINFO

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_ are available.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TRUE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`ICAP_SUPPORTEDEXTIMAGEINFO`
`DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET`

ICAP_FEEDERTYPE

Description

Allows the Application to set scan parameters depending upon the type of feeder.

If the Source has a general type of the feeder only, default scan parameters can depend upon the type of scan (general document or photo). In this case, negotiating this capability will allow the Source adjusting the settings accordingly. It is advised, therefore, that this capability be negotiated prior to the capabilities related to specific Source settings (like `ICAP_*RESOLUTION`, `ICAP_PIXELTYPE`, etc.) but after the other feeder-related capabilities (`CAP_FEEDERENABLED`, `CAP_FEEDERLOADED`).

Application

`MSG_GET` provides a list of available feeder types. `MSG_SET` specifies which type of feeder to use.

Source

Use this capability to report either the types of feeders available or (in the case where there is a general type feeder only) the scan types supported through the feeder.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	`TWFE_GENERAL` `TWFE_PHOTO`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.91

See Also

Best Practices

`CAP_FEEDERENABLED` `CAP_FEEDERLOADED`

ICAP_FILMTYPE

Description

When `ICAP_LIGHTPATH` is set to `TWLP_TRANSMISSIVE` it allows an Application to set what kind of film is being scanned.

Application

Allows setting of current type of transmissive media you wish to scan.

Source

If supported, the Source should compensate for the type of media to return a positive image.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	`TWFM_POSITIVE` `TWFM_NEGATIVE`

Containers

MSG_GET	`TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`ICAP_LIGHTPATH`

ICAP_FILTER

Description

Describes the color characteristic of the subtractive filter applied to the image data. Multiple filters may be applied to a single acquisition.

If the Source supports `DAT_FILTER` as well, then it will apply the filter set by the last `SET` operation invoked by the Application. Setting/Resetting `ICAP_FILTER` will clear the filter associated with `DAT_FILTER`. Setting/Resetting `DAT_FILTER` will clear the filter associated with `ICAP_FILTER`.

Source

If the Source only supports application of a single filter during an acquisition and multiple filters are specified by the application, set the current filter to the first one requested and return `TWRC_CHECKSTATUS`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(empty array)

Allowed Values:	`TWFT_RED`
	`TWFT_GREEN`
	`TWFT_BLUE`
	`TWFT_NONE`
	`TWFT_WHITE`
	`TWFT_CYAN`
	`TWFT_MAGENTA`
	`TWFT_YELLOW`
	`TWFT_BLACK`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

```
DG_IMAGE / DAT_FILTER / MSG_GET
DG_IMAGE / DAT_FILTER / MSG_GETDEFAULT
DG_IMAGE / DAT_FILTER / MSG_SET
DG_IMAGE / DAT_FILTER / MSG_RESET
```

ICAP_FLASHUSED2

Description

For devices that support flash. `MSG_SET` selects the flash to be used (if any). `MSG_GET` reports the current setting. This capability replaces `ICAP_FLASHUSED`, which is only able to negotiate the flash being on or off.

Application

Note that an image with flash may have a different color composition than an image without flash.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWFL_NONE`
	`TWFL_OFF`
	`TWFL_ON`
	`TWFL_AUTO`
	`TWFL_REDEYE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

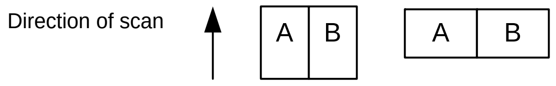
`ICAP_FLASHUSED` in the TWAIN 2.0 Specification

ICAP_FLIPROTATION

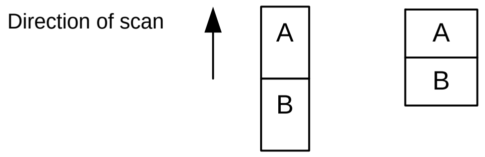
Description

Flip rotation is used to properly orient images that flip orientation every other image.

`TWFR_BOOK`	The images to be scanned are viewed in book form, flipping each page from left to right or right to left.



`TWFR_FANFOLD`	The images to be scanned are viewed in fanfold paper style, flipping each page up or down.



On duplex paper, the As are all located on the top, and the Bs are all located on the bottom. If `ICAP_FLIPROTATION` is set to `TWFR_BOOK`, and fanfold paper is scanned, then every B image will be upside down. Setting the capability to `TWFR_FANFOLD` instructs the Source to rotate the B images 180 degrees around the x-axis.

Because this capability is described to act upon every other image, it will work correctly in simplex mode, assuming that every other simplex image is flipped in the manner described above.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWFR_BOOK`
Allowed Values:	`TWFR_BOOK` `TWFR_FANFOLD`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_FRAMES

Description

The list of frames the Source will acquire on each page.

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.

Values

Type:	`TW_FRAME`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Device dependent

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`TW_IMAGELAYOUT`
`CAP_SEGMENTED`
`CAP_SUPPORTEDCAPSSEGMENTUNIQUE`
`ICAP_MAXFRAMES`
`ICAP_SUPPORTEDSIZES`

ICAP_GAMMA

Description

Gamma correction value for the image data.

Application

Do not use with `TW_CIECOLOR`, `TW_GRAYRESPONSE`, or `TW_RGBRESPONSE` data.

Source

If the application supplies an invalid gamma value, the Source selects the closest value and returns `TWRC_FAILURE / TWCC_CHECKSTATUS`.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any value

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_RANGE`	**// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_HALFTONES

Description

A list of names of the halftone patterns available within the Source.

Application

The application may not rename any halftone pattern.

The application should also set `ICAP_BITDEPTHREDUCTION` to use this capability.

For backwards compatibility, Applications need to be aware that a TWAIN 1.0 Data Sources might respond with a `TW_ARRAY` container.

Values

Type:	`TW_STR32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	Any halftone name

Containers

MSG_GET	`TW_ENUMERATION` `TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ENUMERATION`
MSG_SETCONSTRAINT	`TW_ENUMERATION` `TW_ONEVALUE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

When the `ICAP_BITDEPTHREDUCTION` value is `TWBR_HALFTONE`.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_BITDEPTHREDUCTION`
`ICAP_CUSTHALFTONE`
`ICAP_THRESHOLD`

ICAP_HIGHLIGHT

Description

Specifies which value in an image should be interpreted as the lightest "highlight." All values "lighter" than this value will be clipped to this value. Whether lighter values are smaller or larger can be determined by examining the Current value of `ICAP_PIXELFLAVOR`.

Source

If more or less than 8 bits are used to describe the image, the actual data values should be normalized to fit within the 0-255 range. The normalization need not result in a homogeneous distribution if the original distribution was not homogeneous.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	255
Allowed Values:	0 to 255

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_SHADOW`

ICAP_ICCPROFILE

Description

Informs the application if the source has an ICC profile and can embed or link it in the current `ICAP_IMAGEFILEFORMAT`. Tells the source if the application would like an ICC profile embedded or linked into the image file the source will write.

Application

Use this ICAP to determine if the source supports embedding or linking of ICC profiles into files and to control whether or not the source does so.

Source

This should only be supported if `ICAP_IMAGEFILEFORMAT` is set to a file format that supports the embedding or linking of profiles and the source has an ICC profile it can embed.

Since the given `ICAP_PIXELTYPE` may not have been determined at the time this is called, the source should ignore the current `ICAP_PIXELTYPE`. For example, if the source has an ICC profile for color data, but not grayscale or monochrome data, it should offer values as if all pixeltypes are supported.

In the case of `TWPT_SRGB`, the source should embed the sRGB ICC profile to the file if told to embed a profile.

Values

Type:	`TW_UNIT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWIC_NONE`
	`TWIC_EMBED`
	`TWIC_LINK`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.91

See Also

Best Practices

`DG_IMAGE / DAT_ICCPROFILE / MSG_GET`

ICAP_IMAGEDATASET

Description

Gets or sets the image indices that will be delivered during the standard image transfer done in States 6 and 7. Indices are assumed to start at 1, so a `TW_ONEVALUE` container sets an implied range from 1 to the number specified. `TW_RANGE` returns are useful for those cases where the images are contiguous (5 .. 36). `TW_ARRAY` returns should be used were index values are discontinuous (as could be the case where the user previously set such a data set). See the note in the Values section below.

Application

A `MSG_RESET` operation should always be done before a `MSG_GET` if the application wishes to get the complete list of available images. A `MSG_SET` operation will define the number and order of images delivered during States 6 and 7.

Source

For `MSG_GET`, if a contiguous range of images are available starting from the first index (e.g., 1 .. 36) it is recommended that the `TW_ONEVALUE` container is used specifying just the total number of available images (e.g., 36).

If not supported, return `TWRC_FAILURE/ TWCC_CAPUNSUPPORTED`.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(entire range or set of available images)
After MSG_RESET/MSG_RESETALL:	(entire range or set of available images)
Allowed Values:	0 to 2^{32} -1 (for `MSG_GET`) 1 to 2^{32} -1 (for `MSG_SET`)

Containers

MSG_GET	`TW_ARRAY` (see note below) `TW_RANGE` (see note below) `TW_ONEVALUE` (see note below)
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` (see note below) `TW_ARRAY` (see note below) `TW_RANGE` (see note below)
MSG_SETCONSTRAINT	`TW_ONEVALUE` (see note below) `TW_ARRAY` (see note below) `TW_RANGE` (see note below)
MSG_RESET	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Note: Source must use the container type used during the last successful `SET/RESET` operation. These container types are supported for the returning discontinuous indices that have been previously set by the application. It is highly recommended that for an initialized or reset *Image Store* device, the `TW_ONEVALUE` container be the only one returned by the

`MSG_GET` operation. In other words, the data source should not expose the details of the internal memory management of the Image Store device by claiming that it has a hole in its storage locations due to user deletions. For example, a camera that currently has data for pictures 1 to 10 should report that it has 10 images available. If the user later deletes pictures 5, 7, and 9, it should now report that it has 7 images available (i.e., 1 to 7), and not claim that it has pictures 1, 2, 3, 4, 6, 8, and 10 available. To do so would expose the internal memory management constraints of the device and serves little use but to confuse the user.

Required By

All Image Store Data Sources.

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

ICAP_IMAGEFILEFORMAT

Description

Informs the application which file formats the Source can generate (`MSG_GET`). Tells the Source which file formats the application can handle (`MSG_SET`).

`TWFF_TIFF`	Used for document imaging. Native Linux format. Native Macintosh format memory if both application and data source are version 2.4 or later.
`TWFF_PICT`	Native Macintosh format if either the application or the data source is TWAIN 2.3 and earlier.
`TWFF_BMP`	Native Microsoft format
`TWFF_XBM`	Used for document imaging
`TWFF_JFIF`	Wrapper for JPEG images
`TWFF_FPX`	FlashPix, used with digital cameras
`TWFF_TIFFMULTI`	Multi-page TIFF files
`TWFF_PNG`	An image format standard intended for use on the web, replaces GIF
`TWFF_SPIFF`	A standard from JPEG, intended to replace JFIF, also supports JBIG
`TWFF_EXIF`	File format for use with digital cameras.
`TWFF_PDF`	A file format from Adobe
`TWFF_JP2`	A file format from the Joint Photographic Experts Group ISO/IEC 15444-1
`TWFF_JPX`	A file format from the Joint Photographic Experts Group ISO/IEC 15444-2
`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
`TWFF_PDFRASTER`	A simplified PDF format (https://www.pdfraster.org)

Application

Use this ICAP to determine which formats are available for file transfers, and set the context for other capability negotiations such as `ICAP_COMPRESSION`.

Be sure to use the `DG_CONTROL` / `DAT_SETUPFILEXFER` / `MSG_SET` operation to specify the format to be used for a particular acquisition.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWFF_TIFF`
	`TWFF_PICT`
	`TWFF_BMP`

TWFF_XBM
TWFF_JFIF
TWFF_FPX
TWFF_TIFFMULT
TWFF_PNG
TWFF_SPIFF
TWFF_EXIF
TWFF_PDF
TWFF_JP2
TWFF_JPX
TWFF_DEVAVU
TWFF_PDFA
TWFF_PDFRASTER

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_COMPRESSION
DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET
DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET

ICAP_IMAGEFILTER

Description

For devices that support image enhancement filtering. This capability selects the algorithm used to improve the quality of the image.

Application

- `TWIF_LOWPASS` is good for halftone images.
- `TWIF_BANDPASS` is good for improving text.
- `TWIF_HIGHPASS` is good for improving fine lines.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWIF_NONE`
	`TWIF_AUTO`
	`TWIF_LOWPASS`
	`TWIF_BANDPASS`
	`TWIF_HIGHPASS`
	`TWIF_TEXT`
	`TWIF_FINELINE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_IMAGEMERGE

Description

Merges the front and rear image of a document in one of four orientations: front on the top, front on the bottom, front on the left or front on the right.

Application

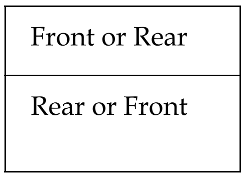
The front and rear have the same settings. It is a customization for a source to allow different settings, for instance a front of `TWPT_RGB` and a rear of `TWPT_BW`.

The merged image can be found at an origin of (0, total-image-height / 2) or (total-image-width, 0), depending on the value of this capability.

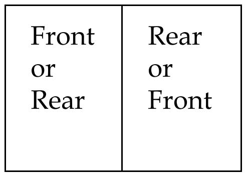
Use the `TWEI_IMAGEMERGED` value with `DAT_EXTIMAGEINFO` to determine if an image is the result of a merge.

Source

This capability only has meaning when scanning duplex.



For `TWIM_FRONTONTOP` and `TWIM_FRONTONBOTTOM` the final image is twice the pixel height of the larger of the two images. The top image has its origin at the upper left hand corner. The bottom image has its origin on the left but down (total-image-height / 2) pixels.



For `TWIM_FRONTONLEFT` and `TWIM_FRONTONRIGHT` the final image is twice the pixel width of the larger of the two images. The left image has its origin at the upper left hand corner. The right image has its origin on the top but left (total-image-width / 2) pixels.

The source chooses how many differences it wants to support between the front and the rear. The only one it is obligated to deal with is differences in the width and height. In both cases the larger value must be selected, and the extra space in the smaller image filled in with some color.

If the source cannot negotiate this capability because of a difference in the front and rear settings, it returns `TWCC_CAPSEQERROR`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWIM_NONE`

Allowed Values:	`TWIM_NONE`
	`TWIM_FRONTONTOP`
	`TWIM_FRONTONBOTTOM`
	`TWIM_FRONTONLEFT`
	`TWIM_FRONTONRIGHT`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

`CAP_DUPLEXENABLED`
`ICAP_IMAGEMERGEHEIGHTTHRESHOLD`
`TWEI_IMAGEMERGED`

ICAP_IMAGEMERGEHEIGHTTHRESHOLD

Description

Specifies a Y-Offset in ICAP_UNITS units. Front and rear images less than or equal to this value are merged according to the settings of ICAP_IMAGEMERGE. If either the front or the rear image is greater than this value, they are not merged.

Application

The Application specifies this value to help with mixed batches of different paper sizes. For instance, a value of 4.0 inches would be enough to merge check-size documents, while leaving larger paper sizes unmerged.

If ICAP_AUTOMATICDESKEW is FALSE, then this value must allow for image skew in the height. If ICAP_AUTOMATICDESKEW is TRUE, then some small amount above the expected document height is still recommended.

Source

This capability only has meaning when CAP_INDICATORS is set to a value other than TWIM_NONE.

Values

Type:	TW_FIX32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0.0
Allowed Values:	0.0 to ICAP_PHYSICALHEIGHT

Containers

MSG_GET	TW_ONEVALUE TW_RANGE
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

CAP_INDICATORS

ICAP_JPEGPIXELTYPE

Description

Allows the application and Source to agree upon a common set of color descriptors that are made available by the Source. This ICAP is only useful for JPEG-compressed buffered memory image transfers.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWPT_BW`
	`TWPT_GRAY`
	`TWPT_RGB`
	`TWPT_PALETTE`
	`TWPT_CMY`
	`TWPT_CMYK`
	`TWPT_YUV`
	`TWPT_YUVK`
	`TWPT_CIEXYZ`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

When the `ICAP_COMPRESSION` value is `TWCP_JPEG`.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_COMPRESSION`

ICAP_JPEGQUALITY

Description

Use this capability as a shortcut to select JPEG quantization tables that tradeoff quality versus compressed image size. Used in concert with `DAT_JPEGCOMPRSSION` it is possible for an Application to obtain the tables that are appropriate for varying percentages of quality within a given Source.

`TWJQ_UNKNOWN` is a read-only value (`MSG_GET` or `MSG_GETCURRENT`), the Application cannot set the Source to this value. This value is reported if the Application uses DAT_JPEGCOMPRESSION to select the quantization tables, and the Source is unable to resolve those tables to a percentage value.

The next three `TWJQ_` values are intended as markers into the quality range, and are only applicable with `MSG_SET`.

`MSG_GET`, `MSG_GETCURRENT` and `MSG_GETDEFAULT` only return values in the range 0 – 100. If an Application wishes to map a `TWJQ_` value to a corresponding value in the range 0 – 100, then it must issue a `MSG_GET` after a `MSG_SET` with one of the three `TWJQ_` values.

No assumption is made about the meaning of the range 0 – 99, it may be derived from the JPEG standard or it may be optimized for the Source's device. 100, though, implies a lossless form of compression. Applications are not encouraged to use this value since it results in poor compression, as well as a format that is not currently widely supported in the industry.

`TWJQ_UNKNOWN` – read-only; must be the setting for this capability if the user sets the JPEG compression tables using `DAT_JPEGCOMPRESSION`, and the Source is not able to map the selected tables to a specific percentage of quality.

`TWJQ_LOW` – write-only; implies low quality; the images are at the maximum compression recommended by the Source.

`TWJQ_MEDIUM` – write-only; implies medium quality; the images are at the balance point between good compression and good images. This is an arbitrary setting on the part of the Source writer that is expected to best represent their device. This is the value that Applications are most encouraged to use.

`TWJQ_HIGH` – write-only; implies high quality; the images display the maximum quality that produces any kind of meaningful compression. Note that images at this setting are still considered to be lossy.

Values

Type:	`TW_INT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWJQ_UNKNOWN`
	`TWJQ_LOW`
	`TWJQ_MEDIUM`
	`TWJQ_HIGH`
	`0 - 100`

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

When the `ICAP_COMPRESSION` value is `TWCP_JPEG`.

TWAIN Version Introduced

Version 1.9

See Also

Best Practices

`ICAP_COMPRESSION`
`ICAP_JPEGSUBSAMPLING`

`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET`
`DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET`

ICAP_JPEGSUBSAMPLING

Description

Use this capability and ICAP_JPEGQUALITY as a shortcut to select JPEG quantization. Used in concert with DAT_JPEGCOMPRSSION it is possible for an Application to obtain the tables that are appropriate for varying percentages of quality within a given Source. It has meaning for color images only.

Source

If requested image is bitonal or grayscale, return TWRC_FAILURE / TWCC_CAPSEQERROR.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWJS_444YCBCR TWJS_444RGB TWJS_422 TWJS_421 TWJS_411 TWJS_420 TWJS_410 TWJS_311

Containers

MSG_GET	TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

ICAP_COMPRESSION
ICAP_JPEGQUALITY

DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GET
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_GETDEFAULT
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_RESET
DG_IMAGE / DAT_JPEGCOMPRESSION / MSG_SET

ICAP_LAMPSTATE

Description

TRUE means the lamp is currently, or should be set to ON. Sources may not support MSG_SET operations.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_EXPOSURETIME
ICAP_FLASHUSED2
ICAP_LIGHTPATH
ICAP_LIGHTSOURCE

ICAP_LIGHTPATH

Description

Describes whether the image was captured transmissively or reflectively.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	`TWLP_REFLECTIVE` `TWLP_TRANSMISSIVE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_EXPOSURETIME`
`ICAP_FILMTYPE`
`ICAP_FLASHUSED2`
`ICAP_LAMPSTATE`
`ICAP_LIGHTSOURCE`

ICAP_LIGHTSOURCE

Description

Describes the general color characteristic of the light source used to acquire the image.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWLS_RED TWLS_GREEN TWLS_BLUE TWLS_NONE TWLS_WHITE TWLS_UV TWLS_IR

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_EXPOSURETIME
ICAP_FLASHUSED2
ICAP_LAMPSTATE
ICAP_LIGHTPATH

ICAP_MAXFRAMES

Description

The maximum number of frames the Source can provide or the application can accept per page.

This is a bounding capability only. It does not establish current or future behavior.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{16}

Containers

MSG_GET	`TW_ONEVALUE`	
	`TW_RANGE`	**// 2.3 and higher**
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_FRAMES`
`TW_IMAGELAYOUT`

ICAP_MINIMUMHEIGHT

Description

Allows the source to define the minimum height (Y-axis) that the source can acquire.

This is a read only capability.

Source

The minimum height that the device can scan. This may be different depending on the value of `CAP_FEEDERENABLED`.

Values

Type:	`TW_FIX32`
Allowed Values:	0 to 32767 in `ICAP_UNITS`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`CAP_FEEDERENABLED`
`ICAP_PHYSICALHEIGHT`
`ICAP_UNITS`

ICAP_MINIMUMWIDTH

Description

Allows the source to define the minimum width (X-axis) that the source can acquire.

This is a read only capability.

Source

The minimum width that the device can scan. This may be different depending on the value of `CAP_FEEDERENABLED`.

Values

Type:	`TW_FIX32`
Allowed Values:	0 to 32767 in `ICAP_UNITS`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.7

See Also

Best Practices

`CAP_FEEDERENABLED`
`ICAP_PHYSICALWIDTH`
`ICAP_UNITS`

ICAP_MIRROR

Description

How the Source can/should mirror the scanned image data prior to transfer. Operation is performed in conjunction with `ICAP_ORIENTATION` and `ICAP_ROTATION`.

Source

Recommend order - Mirror then Rotation.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWMR_NONE`

Allowed Values:	`TWMR_NONE`
	`TWMR_VERTICAL`
	`TWMR_HORIZONTAL`

Containers

MSG_GET	`TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 2.2

See Also

Best Practices

`ICAP_ORIENTATION`
`ICAP_ROTATION`

ICAP_NOISEFILTER

Description

For devices that support noise filtering. This capability selects the algorithm used to remove noise.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWNF_NONE`
	`TWNF_AUTO`
	`TWNF_LONEPIXEL`
	`TWNF_MAJORITYRULE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_ORIENTATION

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_ISO`A4 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`.

The upper-left of the image is defined as the location where both the primary and secondary scans originate. (The X axis is the primary scan direction and the Y axis is the secondary scan direction.) For a flatbed scanner, the light bar moves in the secondary scan direction. For a handheld scanner, the scanner is drug in the secondary scan direction. For a digital camera, the secondary direction is the vertical axis when the viewed image is considered upright.

Application

If one pivots the image about its center, then orienting the image in `TWOR_LANDSCAPE` has the effect of rotating the original image 90 degrees to the "left." `TWOR_PORTRAIT` mode does not rotate the image. The image may be oriented along any of the four axes located 90 degrees from the unrotated image. Note that:

`TWOR_ROT0 == TWOR_PORTRAIT and TWOR_ROT270 == TWOR_LANDSCAPE`.

Source

The Source is responsible for rotating the image if it allows this capability to be set.

Values

Type:	`TW_UINT16`	
Value after MSG_OPENDS:	(may be remembered from a previous session)	
After MSG_RESET/MSG_RESETALL:	`TWOR_PORTRAIT`	
Allowed Values:	`TWOR_ROT0`	
	`TWOR_ROT90`	
	`TWOR_ROT180`	
	`TWOR_ROT270`	
	`TWOR_PORTRAIT`	`(equals TWOR_ROT0)`
	`TWOR_LANDSCAPE`	`(equals TWOR_ROT270)`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`

MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

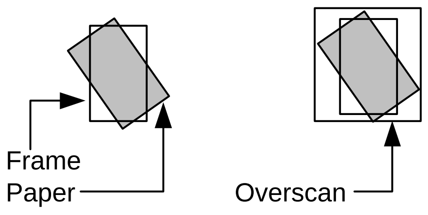
`ICAP_AUTOSIZE`
`ICAP_MIRROR`
`ICAP_ROTATION`

ICAP_OVERSCAN

Description

Overscan is used to scan outside of the boundaries described by `ICAP_FRAMES`, and is used to help acquire image data that may be lost because of skewing.

Consider the following:



This is primarily of use for transport scanners which rely on edge detection to begin scanning. If overscan is supported, then the device is capable of scanning in the inter-document gap to get the skewed image information.

Application

Use this capability, if available, to help software processing images for deskew and border removal.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWOV_NONE`

Allowed Values:	`TWOV_NONE`
	`TWOV_AUTO`
	`TWOV_TOPBOTTOM`
	`TWOV_LEFTRIGHT`
	`TWOV_ALL`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_PATCHCODEDETECTIONENABLED

Description

Turns patch code detection on and off.

Source

Support this capability if the scanner supports any patch code recognition. If the device allows this feature to be turned off, then default to off. If the device does not support disabling this feature, report TRUE and disallow attempts to set FALSE.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

ICAP_PATCHCODEMAXSEARCHPRIORITIES	ICAP_PATCHCODEMAXRETRIES
ICAP_PATCHCODESEARCHPRIORITIES	ICAP_PATCHCODETIMEOUT
ICAP_PATCHCODESEARCHMODE	ICAP_SUPPORTEDPATCHCODETYPES

ICAP_PATCHCODEMAXRETRIES

Description

Restricts the number of times a search will be retried if none are found on each page.

Application

Refine this capability to limit the number of times the patch code search algorithm is retried on a page that contains no patch codes.

Source

If supported, limit the number of retries the value specified.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED`
`ICAP_SUPPORTEDPATCHCODETYPES`
`ICAP_PATCHCODEMAXSEARCHPRIORITIES`
`ICAP_PATCHCODESEARCHPRIORITIES`
`ICAP_PATCHCODESEARCHMODE`
`ICAP_PATCHCODETIMEOUT`

ICAP_PATCHCODEMAXSEARCHPRIORITIES

Description

The maximum number of supported search priorities.

Application

Query this value to determine how many patch code detection priorities can be set.

Source

Set this value to limit the number of priorities to speed the detection process.

If patch code searches can be prioritized, report the maximum number of priorities allowed for a search.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to $2^{32} - 1$

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED`
`ICAP_SUPPORTEDPATCHCODETYPES`
`ICAP_PATCHCODESEARCHPRIORITIES`
`ICAP_PATCHCODESEARCHMODE`
`ICAP_PATCHCODEMAXRETRIES`
`ICAP_PATCHCODETIMEOUT`

ICAP_PATCHCODESEARCHMODE

Description

Restricts patch code searching to certain orientations, or prioritizes one orientation over the other.

Application

Negotiate this capability if the orientation of patch codes is already known to the application. Refinement of this capability can speed the patch code search.

Source

If set then apply the specified refinements to future patch code searches.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWBD_HORZ`
	`TWBD_VERT`
	`TWBD_HORZVERT`
	`TWBD_VERTHORZ`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED`
`ICAP_SUPPORTEDPATCHCODETYPES`
`ICAP_PATCHCODEMAXSEARCHPRIORITIES`
`ICAP_PATCHCODESEARCHPRIORITIES`
`ICAP_PATCHCODEMAXRETRIES`
`ICAP_PATCHCODETIMEOUT`

ICAP_PATCHCODESEARCHPRIORITIES

Description

A prioritized list of patch code types dictating the order in which patch codes will be sought.

Application

Set this capability to specify the order and priority for patch code searching. Refining the priorities to only the patch code types of interest to the application can speed the search process.

Source

If this type of search refinement is supported, then report the current values.

If set, then limit future searches to the specified patch codes in the specified priority order.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)

Allowed Values:	`TWPCH_PATCH1`
	`TWPCH_PATCH2`
	`TWPCH_PATCH3`
	`TWPCH_PATCH4`
	`TWPCH_PATCH6`
	`TWPCH_PATCHT`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	`TW_ONEVALUE` `TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ARRAY`
MSG_RESET	`TW_ARRAY`
MSG_SETCONSTRAINT	`TW_ARRAY`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED`	`ICAP_PATCHCODESEARCHMODE`
`ICAP_SUPPORTEDPATCHCODETYPES`	`ICAP_PATCHCODEMAXRETRIES`
`ICAP_PATCHCODEMAXSEARCHPRIORITIES`	`ICAP_PATCHCODETIMEOUT`

ICAP_PATCHCODETIMEOUT

Description

Restricts the total time spent on searching for a patch code on each page.

Application

Refine this value to tune the length of time the search algorithm is allowed to execute before giving up.

Source

If supported, limit the duration of a patch code search to the value specified.

Values

Type:	`TW_UINT32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	1 to 2^{32} –1

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED`	`ICAP_PATCHCODESEARCHPRIORITIES`
`ICAP_SUPPORTEDPATCHCODETYPES`	`ICAP_PATCHCODESEARCHMODE`
`ICAP_PATCHCODEMAXSEARCHPRIORITIES`	`ICAP_PATCHCODEMAXRETRIES`

ICAP_PHYSICALHEIGHT

Description

The maximum physical height (Y-axis) the Source can acquire (measured in units of `ICAP_UNITS`).

This is a read only capability.

Source

For a flatbed scanner, the height of the platen; for a handheld scanner or a sheet fed scanner, the maximum length of a scan.

For dimensionless devices, such as digital cameras, this ICAP is meaningless for all values of `ICAP_UNITS` other than `TWUN_PIXELS`. If the device is dimensionless, the Source should return a value of zero if `ICAP_UNITS` does not equal `TWUN_PIXELS`. This tells the application to inquire with `TWUN_PIXELS`.

Note: The physical acquired area may be different depending on the setting of `CAP_FEEDERENABLED` (if the Source has separate feeder and non-feeder acquire areas).

Values

Type:	`TW_FIX32`
Allowed Values:	0 to 65535 in `ICAP_UNITS`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_FEEDERENABLED`
`ICAP_UNITS`

ICAP_PHYSICALWIDTH

Description

The maximum physical width (X-axis) the Source can acquire (measured in units of `ICAP_UNITS`).

This is a read only capability.

Source

For a flatbed scanner, the width of the platen; for a handheld scanner or a sheet fed scanner, the maximum width of a scan.

For dimensionless devices, such as digital cameras, this ICAP is meaningless for all values of `ICAP_UNITS` other than `TWUN_PIXELS`. If the device is dimensionless, the Source should return a value of zero if `ICAP_UNITS` does not equal `TWUN_PIXELS`. This tells the application to inquire with `TWUN_PIXELS`. The Source should then reply with its X-axis pixel count.

Note: The physical acquired area may be different depending on the setting of `CAP_FEEDERENABLED` (if the Source has separate feeder and non-feeder acquire areas).

Values

Type:	`TW_FIX32`
Allowed Values:	0 to 65535 in `ICAP_UNITS`

Containers

MSG_GET	`TW_ONEVALUE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`CAP_FEEDERENABLED`
`ICAP_UNITS`

ICAP_PIXELFLAVOR

Description

Sense of the pixel whose numeric value is zero (minimum data value). For example, consider a black and white image:

```
If ICAP_PIXELTYPE is TWPT_BW then
   If ICAP_PIXELFLAVOR is TWPF_CHOCOLATE
        then Black = 0
   Else if ICAP_PIXELFLAVOR is TWPF_VANILLA
        then White = 0
```

Application

Sources may prefer a different value depending on ICAP_PIXELTYPE. Set ICAP_PIXELTYPE and do a MSG_GETDEFAULT to determine the Source's preferences.

Source

TWPF_CHOCOLATE means this pixel represents the darkest data value that can be generated by the device (the darkest available optical value may measure greater than 0).

TWPF_VANILLA means this pixel represents the lightest data value that can be generated by the device (the lightest available optical value may measure greater than 0).

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	TWPF_CHOCOLATE
Allowed Values:	TWPF_CHOCOLATE TWPF_VANILLA

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_PIXELTYPE

ICAP_PIXELFLAVORCODES

Description

Used only for CCITT data compression. Specifies whether the compressed codes' pixel "sense" will be inverted from the Current value of `ICAP_PIXELFLAVOR` prior to transfer.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	`TWPF_CHOCOLATE`
Allowed Values:	`TWPF_CHOCOLATE` `TWPF_VANILLA`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_COMPRESSION`

ICAP_PIXELTYPE

Description

The type of pixel data that a Source is capable of acquiring (for example, black and white, gray, RGB, etc.).

Application

- MSG_GET returns a list of all pixel types available from the Source.
- MSG_SET on a TW_ENUMERATION structure requests that the Source restrict the available pixel types to the enumerated list.
- MSG_SET on a TW_ONEVALUE container specifies the only pixel type the application can accept.

If the application plans to transfer data through any mechanism other than Native and cannot handle all possible ICAP_PIXELTYPEs, it *must* support negotiation of this ICAP.

Values

Type:	TW_UINT16
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	TWPT_BW TWPT_GRAY TWPT_RGB TWPT_PALETTE TWPT_CMY TWPT_CMYK TWPT_YUV TWPT_YUVK TWPT_CIEXYZ TWPT_LAB TWPT_SRGB TWPT_SRGB64 TWPT_BGR TWPT_CIELAB TWPT_CIELUV TWPT_YCBCR TWPT_INFRARED

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_BITDEPTH
ICAP_BITDEPTHREDUCTION

ICAP_PLANARCHUNKY

Description

Allows the application and Source to identify which color data formats are available. There are two options, “planar” and “chunky.”

For example, planar RGB data is transferred with the entire red plane of data first, followed by the entire green plane, followed by the entire blue plane (typical for three-pass scanners). “Chunky” mode repetitively interlaces a pixel from each plane until all the data is transferred (R-G-B-R-G-B…) (typical for one-pass scanners).

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	`TWPC_CHUNKY` `TWPC_PLANAR`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`TW_IMAGEINFO.Planar`

ICAP_ROTATION

Description

How the Source can/should rotate the scanned image data prior to transfer. This doesn't use ICAP_UNITS. It is always measured in degrees. Any applied value is additive with any rotation specified in ICAP_ORIENTATION.

Values

Type:	TW_FIX32
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	+/- 360 degrees

Containers

MSG_GET	TW_ONEVALUE TW_RANGE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_RANGE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_ORIENTATION
ICAP_AUTOSIZE
ICAP_MIRROR

ICAP_SHADOW

Description

Specifies which value in an image should be interpreted as the darkest "shadow." All values "darker" than this value will be clipped to this value.

Application

Whether darker values are smaller or larger can be determined by examining the Current value of `ICAP_PIXELFLAVOR`.

Source

If more or less than 8 bits are used to describe the image, the actual data values should be normalized to fit within the 0-255 range. The normalization need not result in a homogeneous distribution if the original distribution was not homogeneous.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	0 to 255

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_PIXELFLAVOR`

ICAP_SUPPORTEDBARCODETYPES

Description

Provides a list of bar code types that can be detected by the current Data Source.

This is a read only capability.

Application

Query this capability to determine if the Data Source can detect bar codes that are appropriate to the particular application.

Source

If bar code detection is supported, report all the bar code types that can be detected.

Values

Type:	TW_UINT16
Allowed Values:	TWBT_2OF5DATALOGIC TWBT_2OF5IATA TWBT_2OF5INDUSTRIAL TWBT_2OF5INTERLEAVED TWBT_2OF5MATRIX TWBT_2OF5NONINTERLEAVED TWBT_3OF9 TWBT_3OF9FULLASCII TWBT_CODABAR TWBT_CODABARWITHSTARTSTOP TWBT_CODE128 TWBT_CODE93 TWBT_EAN13 TWBT_EAN8 TWBT_MAXICODE TWBT_PDF417 TWBT_POSTNET TWBT_QRCODE TWBT_UCC128 TWBT_UPCA TWBT_UPCE

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_BARCODEDETECTIONENABLED`	`ICAP_BARCODESEARCHMODE`
`ICAP_SUPPORTEDPATCHCODETYPES`	`ICAP_BARCODEMAXRETRIES`
`ICAP_BARCODEMAXSEARCHPRIORITIES`	`ICAP_BARCODETIMEOUT`
`ICAP_BARCODESEARCHPRIORITIES`	

ICAP_SUPPORTEDEXTIMAGEINFO

Description

Lists all of the information that the Source is capable of returning from a call to DAT_EXTIMAGEINFO.

This is a read only capability.

Application

This capability mirrors CAP_SUPPORTEDCAPS. The array indicates all of the possible TWEI_ values the Source is capable of returning. It does not guarantee that all of these values will be returned for every call to DAT_EXTIMAGEINFO, because that depends on the negotiated capabilities and on what the device finds.

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.

Source

The Source lists all of the TWEI_ values it is capable of returning in a call to DAT_EXTIMAGEINFO.

Values

Type:	TW_UINT16
Allowed Values:	An array of TWEI_* values

Containers

MSG_GET	TW_ARRAY
MSG_GETCURRENT	TW_ARRAY
MSG_GETDEFAULT	TW_ARRAY
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

None

TWAIN Version Introduced

Version 2.1

See Also

Best Practices

ICAP_EXTIMAGEINFO
DG_IMAGE / DAT_EXTIMAGEINFO / MSG_GET

ICAP_SUPPORTEDPATCHCODETYPES

Description

A list of patch code types that may be detected by the current Data Source.

This is a read only capability.

Application

Query this capability to determine if the Data Source can detect patch codes that are appropriate to the Application.

Source

If patch code detection is supported, report all the possible patch code types that might be detected.

Values

Type:	`TW_UINT16`

Allowed Values:	`TWPCH_PATCH1`
	`TWPCH_PATCH2`
	`TWPCH_PATCH3`
	`TWPCH_PATCH4`
	`TWPCH_PATCH6`
	`TWPCH_PATCHT`

Containers

MSG_GET	`TW_ARRAY`
MSG_GETCURRENT	`TW_ARRAY`
MSG_GETDEFAULT	`TW_ARRAY`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

`ICAP_PATCHCODEDETECTIONENABLED` `ICAP_PATCHCODESEARCHMODE`
`ICAP_PATCHCODEMAXSEARCHPRIORITIES` `ICAP_PATCHCODEMAXRETRIES`
`ICAP_PATCHCODESEARCHPRIORITIES` `ICAP_PATCHCODETIMEOUT`

ICAP_SUPPORTEDSIZES

Description

For devices that support fixed frame sizes. Defined sizes match typical page sizes. This specifies the size(s) the Source can/should use to acquire image data.

Note: TWSS_B has been removed from the specification.

Source

The frame size selected by using this capability should be reflected in the TW_IMAGELAYOUT structure information.

If the Source cannot acquire the exact frame size specified by the application, it should provide the closest possible size (preferably acquiring an image that is larger than the requested frame in both axes).

For devices that support physical dimensions TWSS_NONE indicates that the maximum image size supported by the device is to be used. Devices that do not support physical dimensions should not support this capability.

Note: TWSS_MAXSIZE has been added to simplify negotiating for the entire acquisition area of a device, since TWSS_NONE was overloaded to mean both "a custom frame" and "the maximum image size."

Values

Type:	TW_UINT16	
Value after MSG_OPENDS:	(may be remembered from a previous session)	
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)	
Allowed Values:	TWSS_NONE	TWSS_USLEDGER
	*TWSS_A4LETTER	TWSS_USEXECUTIVE
	*TWSS_B5LETTER	TWSS_A3
	TWSS_USLETTER	*TWSS_B3
	TWSS_USLEGAL	TWSS_A6
	TWSS_A5	TWSS_C4
	*TWSS_B4	TWSS_C5
	*TWSS_B6	TWSS_C6

(*) Constant should not be used in Sources or Applications using TWAIN 1.8 or higher. For instance, use TWSS_A4 instead of TWSS_A4LETTER (note that the values are the same, the reason for the new constants is to improve naming clarification and consistency).

// 1.8 Additions

TWSS_4A0	TWSS_JISB1
TWSS_2A0	TWSS_JISB2
TWSS_A0	TWSS_JISB3
TWSS_A1	TWSS_JISB4
TWSS_A2	WSS_JISB5(TWSS_B5LETTER)

TWSS_A4(TWSS_A4LETTER)	TWSS_JISB6
TWSS_A7	TWSS_JISB7
TWSS_A8	TWSS_JISB8
TWSS_A9	TWSS_JISB9
TWSS_A10	TWSS_JISB10
TWSS_ISOB0	TWSS_C0
TWSS_ISOB1	TWSS_C1
TWSS_ISOB2	TWSS_C2
TWSS_ISOB3(TWSS_B3)	TWSS_C3
TWSS_ISOB4(TWSS_B4)	TWSS_C7
TWSS_ISOB5	TWSS_C8
TWSS_ISOB6(TWSS_B6)	TWSS_C9
TWSS_ISOB7	TWSS_C10
TWSS_ISOB8	TWSS_USSTATEMENT
TWSS_ISOB9	TWSS_BUSINESSCARD
TWSS_ISOB10	TWSS_MAXSIZE
TWSS_JISB0	

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Image Sources that support fixed frame sizes.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_FRAMES
ICAP_AUTOSIZE
TW_IMAGEINFO
TW_IMAGELAYOUT

ICAP_THRESHOLD

Description

Specifies the dividing line between black and white. This is the value the Source will use to threshold, if needed, when `ICAP_PIXELTYPE` = `TWPT_BW`.

The value is normalized so there are no units of measure associated with this ICAP.

Application

Application will typically set `ICAP_BITDEPTHREDUCTION` to `TWBR_THRESHOLD` to use this capability.

Source

Source should fit available values linearly into the defined range such that the lowest available value equals 0 and the highest equals 255.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	128
Allowed Values:	0 to 255

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

When the `ICAP_BITDEPTHREDUCTION` value is `TWBR_THRESHOLD`.

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_BITDEPTHREDUCTION`

ICAP_TILES

Description

This is used with buffered memory transfers. If TRUE, Source can provide application with tiled image data.

Application

If set to TRUE, the application expects the Source to supply tiled data for the upcoming transfer(s). This persists until the application sets it to FALSE. If the application sets it to FALSE, Source will supply strip data.

Source

If Source can supply tiled data and application does not set this ICAP, Source may or may not supply tiled data at its discretion.

In State 6, ICAP_TILES should reflect whether tiles or strips will be used in the upcoming transfer.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	FALSE
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

TW_IMAGEMEMXFER

ICAP_TIMEFILL

Description

Used only with CCITT data compression. Specifies the minimum number of words of compressed codes (compressed data) to be transmitted per line.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	1
Allowed Values:	1 to 2^{16}

Containers

MSG_GET	`TW_ONEVALUE` `TW_RANGE`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_RANGE`
MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_COMPRESSION`

ICAP_UNDEFINEDIMAGESIZE

Description

If TRUE the Source will issue a MSG_XFERREADY before starting the scan.

Note: The Source may need to scan the image before initiating the transfer. This is the case if the scanned image is rotated or merged with another scanned image.

Application

Used by the application to notify the Source that the application accepts -1 as the image width or -length in the TW_IMAGEINFO structure.

Values

Type:	TW_BOOL
Value after MSG_OPENDS:	FALSE
After MSG_RESET/MSG_RESETALL:	FALSE
Allowed Values:	TRUE or FALSE

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION	 **// 2.0 and higher**
MSG_GETCURRENT	TW_ONEVALUE	
MSG_GETDEFAULT	TW_ONEVALUE	
MSG_SET	TW_ONEVALUE	
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION	
MSG_RESET	TW_ONEVALUE	
MSG_QUERYSUPPORT	TW_ONEVALUE	

Required By

When the Source supports any one or more of the following:

- ICAP_AUTOSIZE
- ICAP_AUTOMATICBORDERDETECTION
- ICAP_AUTOMATICLENGTHDETECTION
- ICAP_AUTOMATICROTATE
- ICAP_FLIPROTATION

TWAIN Version Introduced

Version 1.6

See Also

Best Practices

TW_IMAGEINFO

ICAP_UNITS

Description

Unless a quantity is dimensionless or uses a specified unit of measure, ICAP_UNITS determines the unit of measure for all quantities.

Application

Applications should be able to handle TWUN_PIXELS if they want to support data transfers from "dimensionless" devices such as digital cameras.

Values

Type:	TW_UINT16	
Value after MSG_OPENDS:	TWUN_INCHES	
After MSG_RESET/MSG_RESETALL:	TWUN_INCHES	
Allowed Values:	TWUN_INCHES	TWUN_TWIPS
	TWUN_CENTIMETERS	TWUN_PIXELS
	TWUN_PICAS	TWUN_MILLIMETERS
	TWUN_POINTS	

Containers

MSG_GET	TW_ONEVALUE TW_ENUMERATION
MSG_GETCURRENT	TW_ONEVALUE
MSG_GETDEFAULT	TW_ONEVALUE
MSG_SET	TW_ONEVALUE
MSG_SETCONSTRAINT	TW_ONEVALUE TW_ENUMERATION
MSG_RESET	TW_ONEVALUE
MSG_QUERYSUPPORT	TW_ONEVALUE

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

ICAP_FRAMES
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_GETDEFAULT
DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET
DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET

ICAP_XFERMECH

Description

Allows the application and Source to identify which transfer mechanisms the source supports.

Application

The current value of `ICAP_XFERMECH` must match the data argument type when starting the transfer using the triplet: `DG_IMAGE / DAT_IMAGExxxxXFER / MSG_GET`.

Values

Type:	`TW_UINT16`
Value after MSG_OPENDS:	`TWSX_NATIVE`
After MSG_RESET/MSG_RESETALL:	`TWSX_NATIVE`

Allowed Values:	`TWSX_NATIVE`
	`TWSX_FILE`
	`TWSX_MEMORY`
	`TWSX_MEMFILE`

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`	(permitted TWAIN 2.1 and earlier) (required TWAIN 2.2 and later)
MSG_GETCURRENT	`TW_ONEVALUE`	
MSG_GETDEFAULT	`TW_ONEVALUE`	
MSG_SET	`TW_ONEVALUE`	
MSG_SETCONSTRAINT	`TW_ONEVALUE` `TW_ENUMERATION`	
MSG_RESET	`TW_ONEVALUE`	
MSG_QUERYSUPPORT	`TW_ONEVALUE`	

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGEMEMFILEXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET`
`DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

ICAP_XNATIVERESOLUTION

Description

The native optical resolution along the X-axis of the device being controlled by the Source. Most devices will respond with a single value (`TW_ONEVALUE`).

This is NOT a list of all resolutions that can be generated by the device. Rather, this is the resolution of the device's optics. Measured in units of pixels per unit as defined by `ICAP_UNITS` (pixels per `TWUN_PIXELS` yields dimensionless data).

This is a read only capability.

Values

Type:	`TW_FIX32`
Allowed Values:	>0

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All 2.2 Scanner Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_UNITS`
`ICAP_XRESOLUTION`
`ICAP_YNATIVERESOLUTION`

ICAP_XRESOLUTION

Description

All the X-axis resolutions the Source can provide.

Measured in units of pixels per unit as defined by `ICAP_UNITS` (pixels per `TWUN_PIXELS` yields dimensionless data). That is, when the units are `TWUN_PIXELS`, both `ICAP_XRESOLUTION` and `ICAP_YRESOLUTION` shall report 1 pixel/pixel. Some data sources like to report the actual number of pixels that the device reports, but that response is more appropriate in `ICAP_PHYSICALHEIGHT` and `ICAP_PHYSICALWIDTH`.

Application

Setting this value will restrict the various resolutions that will be available to the user during acquisition.

Applications will want to ensure that the values set for this ICAP match those set for `ICAP_YRESOLUTION`.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	>0

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_UNITS` `ICAP_XNATIVERESOLUTION` `ICAP_YRESOLUTION`

ICAP_XSCALING

Description

All the X-axis scaling values available. A value of '1.0' is equivalent to 100% scaling. Do not use values less than or equal to zero.

Application

Applications will want to ensure that the values set for this ICAP match those set for `ICAP_YSCALING`. There are no units inherent with this data as it is normalized to 1.0 being "unscaled."

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	1.0
Allowed Values:	> 0

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_YSCALING`

ICAP_YNATIVERESOLUTION

Description

The native optical resolution along the Y-axis of the device being controlled by the Source.

Measured in units of pixels per unit as defined by `ICAP_UNITS` (pixels per `TWUN_PIXELS` yields dimensionless data).

This is a read only capability.

Application

Most devices will respond with a single value (`TW_ONEVALUE`). This is NOT a list of all resolutions that can be generated by the device. Rather, this is the resolution of the device's optics

Values

Type:	`TW_FIX32`
Allowed Values:	> 0

Containers

MSG_GET	`TW_ONEVALUE` `TW_ENUMERATION`
MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	Not Allowed
MSG_SETCONSTRAINT	Not Allowed
MSG_RESET	Not Allowed
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All 2.2 Scanner Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_UNITS`
`ICAP_XNATIVERESOLUTION`
`ICAP_YRESOLUTION`

ICAP_YRESOLUTION

Description

All the Y-axis resolutions the Source can provide.

Measured in units of pixels per unit as defined by `ICAP_UNITS` (pixels per `TWUN_PIXELS` yields dimensionless data). That is, when the units are `TWUN_PIXELS`, both `ICAP_XRESOLUTION` and `ICAP_YRESOLUTION` shall report 1 pixel/pixel. Some data sources like to report the actual number of pixels that the device reports, but that response is more appropriate in `ICAP_PHYSICALHEIGHT` and `ICAP_PHYSICALWIDTH`.

Application

Setting this value will restrict the various resolutions that will be available to the user during acquisition.

Applications will want to ensure that the values set for this ICAP match those set for `ICAP_XRESOLUTION`.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	(selected by the data source writer)
Allowed Values:	> 0

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

All Image Sources

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_UNITS`   `ICAP_XRESOLUTION`   `ICAP_YNATIVERESOLUTION`

ICAP_YSCALING

Description

All the Y-axis scaling values available. A value of '1.0' is equivalent to 100% scaling. Do not use values less than or equal to zero.

There are no units inherent with this data as it is normalized to 1.0 being "unscaled."

Application

Applications will want to ensure that the values set for this ICAP match those set for `ICAP_XSCALING`.

Values

Type:	`TW_FIX32`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	1.0
Allowed Values:	> 0

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None

TWAIN Version Introduced

Version 1.0

See Also

Best Practices

`ICAP_XSCALING`

ICAP_ZOOMFACTOR

Description

When used with `MSG_GET`, return all camera supported lens zooming range.

Application

Use this capability with `MSG_SET` to select one of the lens zooming value that the Source supports.

Values

Type:	`TW_INT16`
Value after MSG_OPENDS:	(may be remembered from a previous session)
After MSG_RESET/MSG_RESETALL:	0
Allowed Values:	Source dependent.

Containers

MSG_GET	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_GETCURRENT	`TW_ONEVALUE`
MSG_GETDEFAULT	`TW_ONEVALUE`
MSG_SET	`TW_ONEVALUE`

MSG_SETCONSTRAINT	`TW_ONEVALUE`
	`TW_RANGE`
	`TW_ENUMERATION`

MSG_RESET	`TW_ONEVALUE`
MSG_QUERYSUPPORT	`TW_ONEVALUE`

Required By

None. Highly recommended for digital cameras that are equipped with zoom lenses.

TWAIN Version Introduced

Version 1.8

See Also

Best Practices

11

Return Codes and Condition Codes

Chapter Contents

An Overview of Return Codes and Condition Codes

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

Instead, for each call the application makes to `DSM_Entry( )`, whether aimed at the Source Manager or a Source, the Source Manager returns an appropriate Return Code (`TWRC_xxxx`). The Return Code may have originated from the Source if that is where the original operation was destined.

To get more specific status information, the application can use the `DG_CONTROL / DAT_STATUS / MSG_GET` operation to inquire the complimentary Condition Code (`TWCC_xxxx`) from the Source Manager or Source (whichever one originated the Return Code).

The application should always check the Return Code. If the Return Code is `TWRC_FAILURE`, it should also check the Condition Code. This is especially important during capability negotiation.

There are very few, if any, catastrophic error conditions for the application to worry about. Usually, the application will only have to “recover” from low memory errors caused from allocations in the Source. Most error conditions are handled by the Source Manager or, most typically, by the Source (often involving interaction with the user). If the Source fails in a way that is unrecoverable, it will ask to have its user interface disabled by sending the `MSG_CLOSEDSREQ` to the application’s event loop.

The following operations can only return `TWRC_SUCCESS` or `TWRC_FAILURE` / `TWCC_SEQERROR`, if called in the wrong state. This is to avoid a situation where an Application is unable to shutdown a Source because of an error state, like the device being offline. The Source must comply with the request to change states.

```
DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER
DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET
DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS
DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS
DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDSM
```

When an Application receives this condition code, it alerts the user (so they can exit, if they wish). While waiting for the user response the Application polls the value of `CAP_DEVICEONLINE`. The device continues to be offline as long as this call returns `TWCC_SUCCESS`, with a value of `FALSE`.

The state 3 operation `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS` is the only one capable of returning `TWCC_CHECKDEVICEONLINE`. The Application cannot check `CAP_DEVICEONLINE` (since that is a state 4 operation), however, it can retry the `MSG_OPENDS` call, if it chooses.

Currently Defined Return Codes

Code	Description
`TWRC_BUSY`	Scanner busy, please retry your command again later.
`TWRC_CANCEL`	Abort transfer or the Cancel button was pressed.
`TWRC_CHECKSTATUS`	Partially successful operation; request further information.
`TWRC_DSEVENT`	Event (or Windows message) belongs to this Source.
`TWRC_ENDOFLIST`	No more Sources found after `MSG_GETNEXT`.
`TWRC_FAILURE`	Operation failed - get the Condition Code for more information.
`TWRC_NOTDSEVENT`	Event (or Windows message) does not belong to this Source.
`TWRC_SCANNERLOCKED`	Scanner is in use by another application, please retry your command again later.
`TWRC_SUCCESS`	Operation was successful.
`TWRC_XFERDONE`	All data has been transferred.

Currently Defined Condition Codes

The following are the currently defined condition codes:

Code	Description
TWCC_BADCAP*	Capability not supported by Source or operation (get, set) is not supported on capability, or capability had dependencies on other capabilities and cannot be operated upon at this time (Obsolete, see TWCC_CAPUNSUPPORTED, TWCC_CAPBADOPERATION, and TWCC_CAPSEQERROR).
TWCC_BADDEST	Unknown destination in DSM_Entry.
TWCC_BADPROTOCOL	Unrecognized operation triplet.
TWCC_BADVALUE	Data parameter out of supported range.
TWCC_BUMMER	General failure. Unload Source immediately.
TWCC_CAPBADOPERATION*	Operation (i.e., Get or Set) not supported on capability.
TWCC_CAPSEQERROR*	Capability has dependencies on other capabilities and cannot be operated upon at this time.
TWCC_CAPUNSUPPORTED*	Capability not supported by Source.
TWCC_CHECKDEVICEONLINE	Check the device status using CAP_DEVICEONLINE, this condition code can be returned by any TWAIN operation in state 4 or higher, or from the state 3 DG_CONTROL / DAT_IDENTITY / MSG_OPENDS. The state remains unchanged. If in state 4 the Application can poll with CAP_DEVICELINE until the value returns TRUE.
TWCC_DAMAGEDCORNER	Operation failed because the document has a damaged corner.
TWCC_DENIED	File System operation is denied (file is protected).
TWCC_DOCTOODARK	Operation failed because the document is too dark.
TWCC_DOCTOOLIGHT	Operation failed because the document is too light.
TWCC_FILEEXISTS	Operation failed because file already exists.
TWCC_FILENOTFOUND	File not found.
TWCC_FOCUSERROR	Operation failed because of a focusing error during document capture.
TWCC_INTERLOCK	Operation failed because the cover or door is open.
TWCC_LOWMEMORY	Not enough memory to complete operation.
TWCC_MAXCONNECTIONS	Source is connected to maximum supported number of applications.
TWCC_NODS	Source Manager unable to find the specified Source.

Code	Description
TWCC_NOMEDIA	Source has nothing to capture for a transfer. Can be returned by DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS, by any of the DAT_IMAGE*XFER operations, by DAT_IMAGEINFO or DAT_EXTIMAGEINFO.
TWCC_NOTEMPTY	Operation failed because directory is not empty.
TWCC_OPERATIONERROR	Source or Source Manager reported an error to the user and handled the error; no application action required.
TWCC_PAPERDOUBLEFEED	Transfer failed because of a feeder error, this can be returned by any of the DAT_IMAGE*XFER operations. When received the current TWAIN state remains unchanged.
TWCC_PAPERJAM	Transfer failed because of a feeder error, this can be returned by any of the DAT_IMAGE*XFER operations. When received the current TWAIN state remains unchanged.
TWCC_SEQERROR	Illegal operation for current Source Manager or Source state.
TWCC_SUCCESS	Operation worked.

* TWCC_BADCAP has been replaced with three new condition codes that more clearly specify the reason for a capability operation failure. For backwards compatibility applications should also accept TWCC_BADCAP and treat it as a general capability operation failure. No 1.6 Image Data Sources should return this condition code, but use the new ones instead.

Custom Return and Condition Codes

Although probably not necessary or desirable, it is possible to create custom Return Codes and Condition Codes. Refer to the TWAIN.H file for the value of TWRC_CUSTOMBASE for custom Return Codes and TWCC_CUSTOMBASE for custom Condition Codes. All custom values must be numerically greater than these base values. Remember that the consumer of these custom values will look in your TW_IDENTITY.ProductName field to clarify what the identifier's value means. There is no other protection against overlapping custom definitions.

12

Operating System Dependencies

Chapter Contents

This section describes the differences and requirements of developing TWAIN Applications and Sources on various operating systems. This section covers 32-bit and 64-bit Windows, Apple OS X version 10.2 and later, and Linux. Older operating systems are not described in this version of the specification. For older operating systems (16-bit Windows and Apple OS before version 10.2) please refer to version 1.9 of the Specification.

Developing for Windows

Installation of the Data Source Manager

All TWAIN 2.x Applications must use TWAINDSM.DLL. All TWAIN 2.x Sources must be prepared to work with either TWAINDSM.DLL or TWAIN_32.DLL, which may still be used by older Applications.

Applications that wish to use access the Data Source Manager, must install it themselves. Please refer to the TWAIN website http://www.twain.org to obtain this file and for installation instructions. This DSM is fully backwards compatible with all versions of TWAIN. The Application Installer may include a Data Source Manager merge module:
http://sourceforge.net/projects/twain-dsm/files/

The TWAIN DSM is a shared library named TWAINDSM.DLL. There is a 32-bit and a 64-bit version of this file. TWAINDSM.DLL is installed in the Windows System directory (normally C:\Windows\System32). If installing the 32-bit file on a 64-bit system, it needs to end up in the WOW64 System directory (normally C:\Windows\SysWow64). By including the TWAIN DSM merge module in the application installer, the DSM is installed in the correct location.

Load the TWAIN Source Manager and get the DSM_Entry

This process takes a TWAIN application from State 1 to 2.

Load `TWAINDSM.DLL` using the `LoadLibrary( )` routine.

Get the `DSM_Entry` by using the `GetProcAddress( )` call.

Applications must perform a dynamic run-time link to `DSM_Entry( )` by calling `LoadLibrary( )`. If the Application has a dynamic link, however, it will be able to give users a meaningful error message, and perhaps continue with image acquisition facilities disabled.

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

On Windows the official TWAINDSM.dll is signed. This digital signature can be verified to check if an official version is being used. See http://twain-samples.svn.sourceforge.net.

OpenDSM

To Move from State 2 to State 3

The Application must provide a pointer to `hWnd` in `pParent`. It is strongly recommended that all TWAIN calls be made from a single thread.

OpenDS

To Move from State 3 to State 4

The Source Manager does a `LoadLibrary( )` of the Source and passes the `OpenDS` triplet to the Source. When the Source Manager receives a success from a Source, it increases its internal counter for this application having the specified Source open.

CloseDS

To Move from State 4 to State 3

The Source Manager forwards this triplet to the Source. The Source *immediately* prepares to terminate execution. When the Source Manager receives a success from a Source, it decrements its internal counter to see whether this application still has the specified Source open. If not, the Source Manager removes it from memory (It does a `FreeLibrary( )` of the Source).

Unload DSM

To Move from State 2 to State 1

Once the Source Manager has been closed, the application must unload the DLL from memory before continuing.

Use `FreeLibrary(hDSMLib);` where `hDSMLib` is the handle to the Source Manager DLL returned from the call to `LoadLibrary( )` seen earlier (in the State 1 to 2 section).

Function Declaration

The keyword FAR is included in the entry point syntax for legacy reasons. It has no value for any supported operating system, and is defined as an empty value. See Twain.h for details.

Memory Management in TWAIN

When TWAIN 2.x Applications and Sources connect to TWAINDSM.DLL, they must use the memory functions supplied by the DSM.

When a TWAIN Source is connected to a legacy TWAIN 1.x DSM it must use these legacy WIN32 Global Memory functions:

```
GlobalAlloc, GlobalFree, GlobalLock, GlobalUnlock
```

Using DAT_CALLBACK and DAT_NULL for Messages from the Source to the Application

Sources

TWAIN sources use the `DG_CONTROL/DAT_NULL` to return events like `MSG_XFERREADY`. If the callback pointer is supplied by the DSM, then the DS must use it. If not, it must use the Data Source Manager entry point `DSM_ENTRY`. `MSG_INVOKECALLBACK` was immediately deprecated for Windows and Linux Sources after implementing and should not be used.

Alter the Application's Message Loop

Messages include activities such as key clicks, mouse events, periodic events, accelerators, etc. Every TWAIN-compliant application on Windows needs a message loop. These actions are called messages; however, this may be confusing as TWAIN uses the term "messages" to describe the third parameter of an operation triplet. Therefore, we will refer to these key clicks, etc. as events in this section generically. During a TWAIN session, the application opens one or more Sources. However, even if several Sources are open, the application should only have one Source enabled at any given time. That is the Source from which the user is attempting to acquire data.

Altering the message loop is required so that the source can respond to Windows messages.

Message Loop Modification - Passing messages

While a Source is enabled, all messages are sent to the application's message loop. Some of the messages may belong to the application but others belong to the enabled Source. To ensure that the Source receives and processes its messages, the following changes are required: The application must send all Windows messages that it receives in its message loop to the Source, as long as the Source is enabled. The application uses:

```
DG_CONTROL / DAT_EVENT / MSG_PROCESSEVENT
```

The `TW_EVENT` data structure looks like this:

```
typedef struct {
        TW_MEMREF pEvent;      /* Windows pMSG */
```

```
            TW_UINT16 TWMessage;   /* TW message from Source to */
                                         /* the application */
   } TW_EVENT, FAR *pTW_EVENT;
```

The `pEvent` field points to the message structure.

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

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

DAT_EVENT Handling Errors

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

Cannot use TAB or Keyboard Shortcuts to Navigate TWAIN Dialog

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

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

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

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

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

Refer to the function `EnableDS( )`, in the TWAIN application sample file **main.cpp** at http://twain-samples.svn.sourceforge.net, to see an example of how to modify the message loop for Windows.

The Structure of a Source

The following sections describe the structure of a source

Implementation

The Source is implemented as a Dynamic Link Library (DLL). The DLL runs within the calling application's heap, although DLLs may be able to allocate their own heap and stack space. There is only one copy of the DLL's code and data loaded at run-time per application. For more information regarding DLLs please refer to Microsoft documents.

Naming and Location

The DLL's file name must end with a .DS extension. The Source Manager recursively searches for your Source in the TWAIN sub-directory of the Windows directory. The name of the TWAIN directory is "twain_32" for 32-bit Sources and "twain_64" for 64-bit Sources (on 64-bit systems only). To reduce the chance for naming collisions, each Source should create a sub-directory beneath TWAIN giving it a name relevant to their product. The Source DLLs are placed there. Supporting files may be placed there as well, but since this is a system directory that may only be modifiable by a System Administrator, Sources must not write any information into this directory after installation.

Entry Points and Segment Attributes

Every Source is required to have an entry point called DS_Entry (see Chapter 6, "Entry Points and Triplet Components") which must have the stdcall calling style.

General Notes

- **DllMain entry point** - This function is called by the loader when it loads or unloads a DLL. See http://www.microsoft.com for more details.

Sources UI and Handling Windows Messages

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

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

If it does

Set the `Return Code` for the operation to `TWRC_DSEVENT`

Set the `TWMessage` field to `MSG_NULL`

Process the message

Return to the application

Else

Set the `Return Code` to `TWRC_NOTDSEVENT`

Set the `TWMessage` field to `MSG_NULL`

Return to the application immediately

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

On Windows, the code fragment looks like the following:

```
TW_UINT16 FAR PASCAL DS_Entry(pTW_IDENTITY pSrc,
         TW_UINT32 DG,
         TW_UINT16 DAT,
         TW_UINT16 MSG,
         TW_MEMREF pData)
{
   TWMSG twMsg;
   TW_UINT16 twRc;
   //Valid states 5 – 7 (or 4 – 7 if CAP_DEVICEEVENTS has been
   // negotiated to anything other than its default value of an
   // empty TW_ARRAY). As soon as the application has enabled the
   // Source it must being sending the Source events. This allows
   // the Source to receive events to update its user interface and
   // to return messages to the application. The app sends down ALL
   // message, the Source decides which ones apply to it.
if (MSG == MSG_PROCESSEVENT)
{
   if (hImageDlg && IsDialogMessage(hImageDlg,
   (LPMSG)(((pTW_EVENT)pData)->pEvent)))
      {
         twRc = TWRC_DSEVENT;
         // The source should, for proper form, return a MSG_NULL
         // for all Windows messages processed by the Data Source
         ((pTW_EVENT)pData)->TWMessage = MSG_NULL;
      }
      else
      {
               // notify the application that the source did not
               // consume this message
```

```
                    twRc = TWRC_NOTDSEVENT;
                 ((pTW_EVENT)pData)->TWMessage = MSG_NULL;
              }
                 }
                 else
                 {
              // This is a Twain message, process accordingly.
              // The remainder of the Source’s code follows...
                 }
        return twRc;
        }
```

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

If the Source has more than one window it has to check all of them and process the target one.

Native Transfer Mode

Every Source must support Native transfer mode. It is the default mode and is the easiest for an application to implement, however it is restrictive and the format is limited to the Device Independent Bitmap (DIB) when implementing on Windows.

Set `pHandle` pointing to a handle to a DIB in memory. The Source will allocate the image buffer and return the handle to the address specified.

Format the data block as a DIB. Use l `DSM_MemAllocate` (or `GlobalAlloc` for 1.x or less DSM)

The following assignment will work in Windows:

```
*(TW_HANDLE *) pHandle = hDIB;
```

See the Windows SDK documentation under Structures: `BIMAPINFO, BITMAPINFOHEADER, RGBQUAD`.

See also “DIBs and their use” by Ron Gery, in the Microsoft Development Library (MSDN CD).

Note:

- Follow the `BITMAPINFOHEADER` with the color table if required
- Color table entries are RGBQUADs, which are stored in memory as BGR not RGB.
- For 24-bit color DIBs, the “pixels” are also stored in BGR order, not RGB.
- DIBs are stored ‘upside-down’ - the first pixel in the DIB is the lower-left corner of the image, and the last pixel is the upper-right corner.
- Pixels in 1, 4, and 8 bit DIBs are “always” color table indices, you must index through the color table to determine the color value of a pixel.

- Every scanline is DWORD-aligned. The scanline is buffered to alignment; the buffering is not necessarily 0.

Native Audio format is WAV handle

File Transfer

The `TW_SETUPFILEXFER`

`pSetupFile->FileName` = name of file must include the complete path and name

`pSetupFile->VRefNum` = is not used

Developing for Mac

Mac OS X includes high-level native development environment that you can use for your application's graphical user interface: Cocoa. This is full-featured development environment in its own right. Note that Cocoa libraries are 32 and 64-bit. (See http://developer.apple.com). Only the Intel platform is supported, starting from OS X 10.6.

Since Mac OS X 10.7 Apple supports 64-bit Intel Macs only. To allow native, 64-bit TWAIN Applications to access TWAIN Data Sources without any legacy compatibility mode, Data Sources must include 64-bit support. A 32-bit version can and should still be installed to support older Applications and older Mac OS X versions

Because Cocoa changes the event handling mechanism (no `WaitNextEvent` loops), these paragraphs update and extend the section of the previous specification that describes how to modify the application event loop to support TWAIN. Cocoa-based Mac OS X TWAIN applications are required to supply an event handler callback function that the TWAIN DSM will call.

Installation of the Data Source Manager

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

For Mac OS X version 10.2 and later, the Source Manager is installed automatically with the OS and developers should not install or modify TWAIN.framework.

For TWAIN 2.x compatibility vendor's may choose to install the TWAIN Working Group's TWAINDSM.framework into /Library/Frameworks/.

The directory “TWAIN Data Sources” should be created in "/Library/Image Capture" if it does not exist. If you are a scanner vendor, install your scanner data sources into a subdirectory of "/Library/Image Capture/TWAIN Data Sources" directory.

Load the TWAIN Source Manager and get the DSM_Entry

This process takes a TWAIN application from State 1 to 2.

Link against TWAIN.framework.

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

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

For TWAIN 2.x compatibility dynamically load the /Library/Frameworks/TWAINDSM.framework instead of static linking against the legacy TWAIN.framework. In case of failure fall back to loading the system's default TWAIN 1.x DSM from /System/Library/Frameworks/TWAINDSM.framework.

OpenDSM

To Move from State 2 to State 3

Application must set `pParent` to `NULL`

OpenDS

The Source Manager does a `dsOpen( )` of the Source and sends an `OpenDS` triplet to the Source.

CloseDS

Closes the Source and removes it from memory by calling `dsClose`, following receipt of `TWRC_SUCCESS` from the Source.

Unload DSM

To Move from State 2 to State 1

No action is necessary.

Function Declaration

The keyword FAR is included in the entry point syntax for legacy reasons. It has no value for any supported operating system, and is defined as an empty value. See Twain.h for details.

Using DAT_CALLBACK and DAT_NULL for Messages from the Source to the Application

Sources

TWAIN sources that do not detect `DF_DSM2` in `TW_IDENTITY.SupportedGroups` must use `DG_CONTROL/DAT_CALLBACK/MSG_INVOKE_CALLBACK` to return events like `MSG_XFERREADY.READY`.

Memory Management in TWAIN

When TWAIN Applications and Sources must use these memory functions:

Memory allocation - `NewHandle`

Memory free - `DisposeHandle`

Memory lock - `dereference the handle`

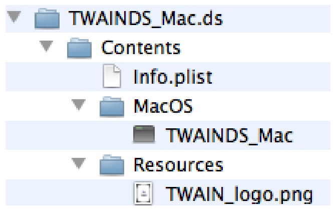
Memory unlock - `is a no op`

The Structure of a Source

The following sections describe the structure of a source

Implementation

A Source on a Macintosh is implemented as a bundle. The Source will not run standalone. A separate copy of the Source's code in memory will be made for each application that opens the Source. The bundle must contain a Contents directory and, inside it, an `Info.plist` file. It is recommended that bundle contains universal 32/64 binary. Bundle structure is shown below:



```
TWAINDS_Mac.ds
  Contents
    Info.plist
    MacOS
      TWAINDS_Mac
    Resources
      TWAIN_logo.png
```

See the sample DS - http://twain-samples.svn.sourceforge.net.

Naming and Location

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

Compatibility with Older Data Sources

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

Sources and Handling Events

Do not receive `TW_EVENTS` from the DSM.

On Mac OS X, the Data source uses Cocoa.

- A Cocoa Data Source gets the UI event handling for free.

Implementing Modal and Modeless User Interfaces

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

When sending `MSG_ENABLEDS` set `hParent` to `NULL`.

Modal (App Modal)

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

This is done by first specifying setting the `ModalUI` field to `TRUE` and second by enabling/ disabling the parent window during the `MSG_ENABLEDS / MSG_DISABLEDS` operations. Use EnableWindow(`NULL`, `FALSE`) to disable the window and EnableWindow(`NULL`, `FALSE`) to re-enable it.

Modeless

If implementing a modeless user interface, set the `ModalUI` field to `FALSE`. Also, it is suggested that you call `BringWindowToTop( )` whenever a second request is made by the same application or another application requesting access to a Source that supports multiple application connections.

Implementing Modal and Modeless User Interfaces

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

Native Transfer Mode

Every Source must support Native transfer mode. It is the default mode and is the easiest for an application to implement, however it is restrictive.

If both the application and the data source are TWAIN 2.4 and later:

The native format is a TIFF.

Set `pHandle` pointing to a TIFF file located in memory. The Source will allocate the image buffer and return the handle to the address specified.

If either the application or the data source is TWAIN 2.3 and earlier:

The native format is a PICT.

The version of PICT to be transferred is the latest version available on the machine on which the application is running (usually PICT II for machines running 32-bit/color QuickDraw and PICT I for machines running black and white QuickDraw).

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

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

Native Audio format is audio handle

`ICAP_XNATIVERESOLUTION` and `ICAP_YNATIVERESOLUTION` are required for Mac ImageCaputre to use TWAIN Data Source. These capabilities are not required by TWAIN.

File Transfer

The `TW_SETUPFILEXFER`

`pSetupFile->FileName` = name of file must only

`pSetupFile->VRefNum` = The volume reference and folder reference number.

Developing for Linux

Installation of the Data Source Manager

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

The TWAIN DSM is a shared library named libtwaindsm.so. There is a 32-bit and a 64-bit version of this file. `libtwaindsm.so` location is: /usr/local/lib. For 32-bit DS running on 64-bit system it is: /usr/local/lib32

Load the TWAIN Source Manager and get the DSM_Entry

This process takes a TWAIN application from State 1 to 2.

Load `TWAINDSM.so` using the `dlopen( )` routine.

Get the `DSM_Entry` by using the `dlsym( )` call.

The Source Manager is a shared library.

OpenDSM

To Move from State 2 to State 3

Application must set `pParent` to `NULL`

OpenDS

The Source Manager does a `dlopen( )` of the Source and sends an `OpenDS` triplet to the Source.

CloseDS

Checks its internal counter to see whether any other applications are accessing the specified Source. If so, the Source Manager takes no other action. If the closing application is the last to be accessing this Source, the Source Manager closes the Source (forwards this triplet to it) and removes it from memory, following receipt of `TWRC_SUCCESS` from the Source.

Upon receiving the request from the Source Manager, the Source immediately prepares to terminate execution.

Unload DSM

To Move from State 2 to State 1

Once the Source Manager has been closed, the application must unload the SO from memory before continuing.

Use `dlclose(hDSMLib);` where `hDSMLib` is the pointer to the Source Manager SO returned from the call to `dlopen( )` seen earlier (in the State 1 to 2 section).

Memory Management in TWAIN

All TWAIN Applications and Sources must use the memory functions supplied by the DSM.

The Structure of a Source

The following sections describe the structure of a source.

Implementation

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

Naming and Location

The extension for a Source is ds. The Source Manager will search for a shared library with this extension in the /usr/local/lib/twain directory. For 32-bit DS running on 64-bit system it is: /usr/local/lib32/twain

It is recommended that each Source is placed in its own directory and any other files it may require placed with it.

Implementing Modal and Modeless User Interfaces

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

Native Transfer Mode

Every Source must support Native transfer mode. It is the default mode and is the easiest for an application to implement, however it is restrictive and the format is limited to TIFF when implementing on Linux.

For Linux, the native format is a TIFF.

Set `pHandle` pointing to a TIFF file located in memory. The Source will allocate the image buffer and return the handle to the address specified.

Native Audio format is WAV

File Transfer

The `TW_SETUPFILEXFER`

`pSetupFile->FileName` = name of file must include the complete path and name

`pSetupFile->VRefNum` = is not used

13

TWAIN Self-Certification Process for Data Sources

Chapter Contents

The TWAIN self-certification system helps developers test their data source’s support of the basic interface described by the TWAIN Specification. Passing the test helps to confirm that the data source’s interface works as expected with applications, leading to a better user experience.

This document provides the Test Plan for TWAIN self-certification for data sources. It also describes how to submit a form affirming successful completion of the test to receive authorization to display the “TWAIN Certified” logo.

Overview

The TWAIN interface operates between an application and a data source. The nature of this interface is described by the TWAIN Specification.

Basic TWAIN self-certification exercises specific portions of the TWAIN interface and behavior of the TWAIN interface that all data sources are required to support. Passing these tests confirms that a data source correctly follows the TWAIN Specification, when responding to commands sent by an application, and that it does not crash or hang.

This is not a comprehensive test of the entire TWAIN interface. It focuses on enforcing basic "good behavior". More stringent tests may be described in future.

The basic self-certification test is limited to the kinds of checks described in this document. Modifications may be made in association with new versions of the TWAIN Specification (for instance, the addition of a new mandatory feature). For this reason self-certification is always done in the context of a particular version of the TWAIN Specification (ex: 2.2).

TWAIN data sources with a protocol version of 1.9 or higher may be self-certified. The version of this document is a measure of the kinds of tests performed on the data source. Running the tests in this document does not certify a TWAIN 1.9 data source as TWAIN 2.2 compliant, rather the data source is self-certified as TWAIN 1.9 compliant using criteria described inside of the TWAIN 2.2 Specification.

TWAIN data sources that have been self-certified will work correctly with any compliant TWAIN Application reporting a TWAIN protocol version of 1.5 or higher.

TWAIN self-certification promotes the creation of 64-bit applications and data sources by requiring simultaneous submissions of native 32-bit and 64-bit data sources for Windows Vista or later, Macintosh OS X or Linux. A native 64-bit data source is one that interfaces with a native 64-bit application. 64-bit applications cannot be run on 32-bit Systems. A 32-bit data source running in any kind of virtual or thunking environment on a 64-bit Operating System does not qualify as a native 64-bit data source.

TWAIN self-certification requires the presence of a TWAIN data source manager corresponding to the version of the TWAIN data source or higher. If one is not pre-installed on the operating system, then the TWAIN data source must install it.

Questions or comments regarding TWAIN self-certification should be referred to the TWAIN Forum www.twainforum.org.

Non-Goals of Basic TWAIN Self-Certification

This is a test of the operation of the interface; it does not test the internals of the data source.

This test is not designed to catch data errors (ex: bad pointers, data corruption, array out of bounds, etc) except in those instances where the error happens to cause the failure of some other test.

Negotiated settings are not confirmed in the meta-data or images they produce (ex: did changing `ICAP_BRIGHTNESS` really result in a brighter or darker image, was the proper print string written on the document).

Constraints for TW_ENUMERATION and TW_RANGE are not tested (ex: limiting the ICAP_PIXELTYPE enumeration to just TWPT_RGB, or limiting ICAP_BRIGHTNESS to a range of -100 to 100).

Mandatory features for accessories are not tested (ex: there is no check to make sure that all of the barcode features are properly supported if any one barcode capability is detected).

Affirmation of Successful Completion of TWAIN Self-Certification

After TWAIN self-certification has been successfully completed the tester may submit an "Acknowledgement of Successful Completion of TWAIN Self-Certification" form to the TWAIN Working Group.

This can be accomplished in more than one way. The preferred method is to access the TWAIN Working Group website (www.twain.org), and access the section titled "Scanner Driver Developers." Under there is the "Certify TWAIN Driver" link.

Alternatively, one can submit a notarized or a digitally signed form of the document

This form includes the following information

Company: The name of the company manufacturing the data source being self-certified, a division within that company may be optionally provided. The submitter may also opt to provide a URL to their company's website which will link off of this name.

Hardware: The model name, model number and revision of the hardware used during self-certification. This is marketing information identifying the device supported by this specific TWAIN data source. In most cases this information can be found printed somewhere on the device.

TWAIN Data Source Identity: Fields from the TWAIN data source's TW_IDENTITY structure, which indicate the manufacturer, family, product, and the version number, uniquely identify the data source to the application. The TW_IDENTITY.ProductName should be unique by itself, since this is the only field displayed by the data source manager's user select dialog on Windows.

TWAIN Data Source Version: The complete version of the TWAIN data source, matching the .DLL version on Windows, and the .so file name on Linux and Mac OS X, this version number matches the MajorNum and MinorNum fields from the data source's TW_IDENTITY.Version structure.

Installation: The name and the version of the installation media that includes this TWAIN data source provides information the user needs to install the self-certified TWAIN driver.

Operating System: The operating system's name and revision (version number or service pack) that was used during self-certification.

Processor: The computer processor of the host machine used during self-certification, examples include: x86, x64, IA64. This should match the native processor supported by the TWAIN data source. For example, if the self-certification is performed for a 32-bit TWAIN data source on Windows XP or Linux without a 64-bit data source, then the x86 processor should be used.

32-Bit / 64-Bit: When performing the self-certification test on Windows Vista or later, or any version of Macintosh OS X, or Linux, the submitted form must indicate successful completion using both a native 32-bit and a native 64-bit data source.

Email: The name and email address of a contact. This is initially used to deliver the Logo, but it will also be used to help manage entries posted by the TWAIN Working Group.

URL: The URL to the Installer for the TWAIN data source is a convenience for users browsing the posted list of self-certified content. It is optional, but recommended.

Self-Certification Method: The submitter may specify the software used to perform self-certification, when indicated this information is made available to users browsing the posted list of self-certified content.

It is expected that multiple versions of the same driver will be submitted over the life of the hardware product. Please be aware of the following:

Email address: The email address specifies the contact who receives the Logo for a successful submission. This same email address must be used when submitting a new instance of a previously submitted TWAIN data source, or when replacing an existing submission. Requests using other email addresses may not be recognized by the TWAIN Working Group.

Signature: There is no requirement for the same signature (notarized or digital) to be used from one submission to the next, but pairing the same signature with the same email address for all submissions for a given driver is appreciated.

Hardware: The model name and model number uniquely identifies the hardware supported by the TWAIN data source. Submissions of new TWAIN data sources for the same hardware must take care to make sure that this information is identical from one version to the next. If there is no exact match with an existing hardware entry, then the entire entry is treated as new.

TWAIN Data Source Identity: The following fields uniquely identify the TWAIN data source: `TW_IDENTITY.Manufacturer`, `TW_IDENTITY.ProductFamily` and `TW_IDENTITY.ProductName`. When updating a previously existing self-certified TWAIN data source it is important to make sure this data is identical from one version to the next. If there is no exact match with an existing TWAIN data source, then the entire entry is treated as new.

TWAIN Data Source Version: Many vendors use a four field versioning system (ex: 1.2.0.1). The first two fields must correspond to the `TW_IDENTITY.Info.Version.Major` and `TW_IDENTITY.Info.Version.Minor` fields. The last two fields vary among vendors, and are not described here. The value zero must be used for any unused field. If a submission has exactly the same email, hardware, data source and version information as a previous submission, it will replace its posting on the TWAIN Working Group website. If there is no exact match with an existing TWAIN data source, then the entire entry is treated as new.

Operating System: The operating system's name and revision (version number or service pack) that was used during self-certification. If there is no exact match with an existing TWAIN data source, then the entire entry is treated as new.

The TWAIN Working Group makes no attempt to enforce continuity of versions. If the submission is correct, the version numbers may change in any way specified by the submitter.

Submission of the form qualifies the data source and its associated hardware to display the TWAIN Certified Logo. Submission information from the form is displayed on the TWAIN Working Group website (www.twain.org).

Contact information is required to deliver the Logo; this includes the name of a contact and an email address. This information will not be shared or made public. The form asks if the email address may be used to occasionally send information relating to TWAIN or the TWAIN Working Group.

The form must be either digitally signed or notarized. This identification is meant to guarantee that the document has not been modified since it was signed. The form includes an address where it can be mailed as a paper copy or emailed. The complete form is on the next two pages.

Form

Affirmation of Successful Completion of TWAIN Self-Certification
Compliance with TWAIN Versions 1.9 through 2.2
Page 1 of 2

Completion and submission of a digitally signed or notarized original of this statement to the TWAIN Working Group authorizes the authorized representative or their company to display the TWAIN Certified Logo on the hardware, software and marketing materials of the TWAIN data source described below. All fields must be filled in, except where otherwise indicated.

The certification mark is intended for use by authorized entities or persons and is intended to certify that this software conforms to standards designated by the TWAIN Working Group. This document indicates compliance with the TWAIN Specification for version TWAIN 2.2 or earlier.

The following information will not be published or shared. The Logo will be sent to the email address.

Name of Contact: ____

Email Address: ____

May the TWAIN Working Group send TWAIN information not related to this submissionto this email address? (circle one) [Yes] [No]

The following fields will be posted on the TWAIN Working Group website.

Company:	________________________
Division: (optional)	________________________
Company/Division URL: (optional)	________________________
Hardware Model Name:	________________________
Hardware Model Number:	________________________
Hardware Model Revision: (optional)	________________________
TW_IDENTITY.Manufacturer:	________________________
TW_IDENTITY.ProductFamily:	________________________
TW_IDENTITY.ProductName:	________________________
TW_IDENTITY.Protocol:	______. ______
TWAIN Data Source Version:	______. ______ . ______ . ______
Installer Version:	________________________
URL to Data Source: (optional)	________________________
Processor:	x86 ___ x64 ___ other ________________
Operating System/Revision:	________________________
Self-Certification Software: (optional)	________________________

May the TWAIN Working Group post the software used to self-certify? (circle one)
[Yes] [No]

<u>Affirmation of Successful Completion of TWAIN Self-Certification</u>
Compliance with TWAIN Versions 1.9 through 2.2
Page 2 of 2

Please confirm that all tests described within the "TWAIN Self-Certification Process for Data Sources" document have been completely and successfully run (check all that apply).

32-bit	64-bit	Test
☐	☐	TWAIN Standard Capability Tests
☐	☐	Vendor Custom Capability Tests
☐	☐	Status Return Tests
☐	☐	Stress Tests
☐	☐	Non-UI Image Transfer Tests
☐	☐	UI Image Transfer Tests
☐	☐	CAP_XFERCOUNT
☐	☐	Version Tests

I attest under penalty of perjury to the fact that the information on this form is true and accurate.

______________________________ ______________
Signature of Authorized Representative Date

____
Printed Name

Subscribed and duly sworn in my presence this ______ day of _______________ 20___.

Country of _________________ State of ________________________

SS ______________________
Notary Public Signature

My commission expires: ______________________

Mail the Notarized Document to:

The TWAIN Working Group
7960 Soquel Drive B113
Aptos, Ca. 95003

- or -

Email the Digitally Signed Document to:

admin@twain.org

Sample Form

Affirmation of Successful Completion of TWAIN Self-Certification
Compliance with TWAIN Versions 1.9 through 2.2
Page 1 of 2

Completion and submission of a digitally signed or notarized original of this statement to the TWAIN Working Group authorizes the authorized representative or their company to display the TWAIN Certified Logo on the hardware, software and marketing materials of the TWAIN data source described below. All fields must be filled in, except where otherwise indicated.

The certification mark is intended for use by authorized entities or persons and is intended to certify that this software conforms to standards designated by the TWAIN Working Group. This document indicates compliance with the TWAIN Specification for version TWAIN 2.2 or earlier.

The following information will not be published or shared. The Logo will be sent to the email address.

Name of Contact: __John Smith________________________________

Email Address: __twainselfcert@notarealcompany.com_________

May the TWAIN Working Group send TWAIN information not related to this submissionto this email address? (circle one) [Yes] [No]

The following fields will be posted on the TWAIN Working Group website.

Company:	Not A Real Company
Division: (optional)	Scanner Group
Company/Division URL: (optional)	www.notarealcompany.com/scanners
Hardware Model Name:	Business Scanner
Hardware Model Number:	123
Hardware Model Revision: (optional)	6.0
TW_IDENTITY.Manufacturer:	Not A Real Company
TW_IDENTITY.ProductFamily:	Business Scanner
TW_IDENTITY.ProductName:	Not A Real Scanner: 123
TW_IDENTITY.Protocol:	__2___. __1___
TWAIN Data Source Version:	___5___. ___3___ . ___0___ . ___0___
Installer Version:	Not A Real Scanner: 123, CD v3.4.0.0
URL to Data Source: (optional)	www.notarealcompany.com/scanners/123
Processor:	x86 _x_ x64 _x_ other ______________________
Operating System/Revision:	Windows Vista / SP2
Self-Certification Software: (optional)	Inspector TWAIN 3.1.14

May the TWAIN Working Group post the software used to self-certify? (circle one)
[Yes] [No]

Affirmation of Successful Completion of TWAIN Self-Certification
Compliance with TWAIN Versions 1.9 through 2.2
Page 2 of 2

Please confirm that all tests described within the "TWAIN Self-Certification Process for Data Sources" document have been completely and successfully run (check all that apply).

32-bit	64-bit	Test
X	X	TWAIN Standard Capability Tests
X	X	Vendor Custom Capability Tests
X	X	Status Return Tests
X	X	Stress Tests
X	X	Non-UI Image Transfer Tests
X	X	UI Image Transfer Tests
X	X	CAP_XFERCOUNT
X	X	Version Tests

I attest under penalty of perjury to the fact that the information on this form is true and accurate.

____ ____________________
Signature of Authorized Representative Date

____
Printed Name

Subscribed and duly sworn in my presence this ______ day of _______________ 20___.

Country of _________________ State of ________________________

SS ________________________
Notary Public Signature

My commission expires: ________________________

Mail the Notarized Document to:

The TWAIN Working Group
7960 Soquel Drive B113
Aptos, Ca. 95003

- or -

Email the Digitally Signed Document to:

admin@twain.org

TWAIN “Congratulations” Webpage

Applications that automate the TWAIN self-certification process are asked to use the “Congratulations” web page to complete the process. Hard coding the “Affirmation of Successful Completion of TWAIN Self-Certification” may require updates to the application if the TWAIN Working Group changes the document. Use of the web page avoids this problem.

The URL of the web page is:

http://www.twain.org/self_certification_congratulations.shtm

TWAIN Self-Certification Tests

The tests are broken down into the following groups:

TWAIN Standard Capability Tests	Exercise `DAT_CAPABILITY` operations for all standard TWAIN capabilities reported by `CAP_SUPPORTEDCAPS`. Confirm use of containers and supported operations.
Vendor Custom Capability Tests	Exercise `DAT_CAPABILITY` operations for any vendor specific custom capabilities reported by `CAP_SUPPORTEDCAPS`.
Status Return Tests	Confirm that the expected status return is reported by certain operations.
Stress Tests	Stress aspects of data sources that have been reported as common problems.
Non-UI Image Transfer Tests	Confirm that multiple `MSG_ENABLEDS` and `MSG_DISABLEDS` calls can be made in the context of one `MSG_OPENDS` / `MSG_CLOSEDS`. This test focuses on image capture with no UI.
UI Image Transfer Tests	Confirm that multiple `MSG_ENABLEDS` and `MSG_DISABLEDS` calls can be made in the context of one `MSG_OPENDS` / `MSG_CLOSEDS`. This test focuses on image capture with the UI.
ICAP_XFERMECH	Test the ability of the data source to transfer the correct number of images based on the value of `ICAP_XFERMECH`.
Version Test	Confirm that the data sources responds correctly to different TWAIN versions of data source manager and application.

TWAIN Standard Capability Tests

Purpose

Exercise all of the TWAIN Standard capabilities exposed by `CAP_SUPPORTEDCAPS` using the standard operations supported by `DG_CONTROL / DAT_CAPABILITY`.

Operations on capabilities (`MSG_*` values specified below) are assumed to be `DG_CONTROL / DAT_CAPABILITY`, unless otherwise stated.

Pre-Test Procedure

Open the data source manager. It is required that when opened the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Confirm Basic Negotiation with CAP_SUPPORTEDCAPS

Make sure that `CAP_SUPPORTEDCAPS` is working properly. Perform basic checks on how well it supports negotiation.

1. **Action**: `MSG_GET CAP_SUPPORTEDCAPS` (get the list of capabilities to be tested)

   1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

   1.2. **Test**: If `TW_CAPABILITY.Cap` is not `CAP_SUPPORTEDCAPS`, then end with error

   1.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ARRAY`, then end with error

   1.4. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

   1.5. **Test**: If `TW_ARRAY.ItemType` is not `TWTY_UINT16`, then end with error

   1.6. **Test**: If `TW_ARRAY.NumItems` is equal to zero, then end with error

   1.7. **Action**: Confirm the presence of the following capabilities in `TW_ARRAY.ItemList`

      1.7.1. **Test**: If `CAP_SUPPORTEDCAPS` not found, then end with error

      1.7.2. **Test**: If `ICAP_PIXELTYPE` not found, then end with error

      1.7.3. **Test**: If `ICAP_XFERMECH` not found, then end with error

Confirm Basic Negotiation with ICAP_PIXELTYPE

Make sure that `ICAP_PIXELTYPE` is working properly. Perform basic checks on how well it supports negotiation.

2. **Action**: `MSG_GET ICAP_PIXELTYPE`

2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

2.2. **Test**: If `TW_CAPABILITY.Cap` is not `ICAP_PIXELTYPE`, then end with error

2.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ENUMERATION`, then end with error

2.4. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

2.5. **Test**: If `TW_ENUMERATION.ItemType` is not `TWTY_UINT16`, then end with error

2.6. **Test**: If `TW_ENUMERATION.NumItems` is equal to zero, then end with error

Confirm Basic Negotiation with ICAP_BITDEPTH

Make sure that `ICAP_BITDEPTH` is working properly, and doesn't include invalid values for commonly used pixel types.

3. **Action**: `MSG_SET ICAP_PIXELTYPE` to `TWPT_BW`

3.1. **Test**: If result is not `TWRC_SUCCESS`, then proceed to the `TWPT_GRAY` test immediately below

3.2. **Action**: `MSG_GET ICAP_BITDEPTH`

3.2.1. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ENUMERATION`, then proceed to the `TWPT_RGB` test below

3.2.2. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

3.2.3. **Test**: If `TW_ENUMERATION.ItemType` is not `TWTY_UINT16`, then end with error

3.2.4. **Test**: If the `TW_ENUMERATION.ItemList` includes a value of `24`, then end with error

4. **Action**: `MSG_SET ICAP_PIXELTYPE` to `TWPT_GRAY`

4.1. **Test**: If result is not `TWRC_SUCCESS`, then proceed to the `TWPT_RGB` test below

4.2. **Action**: `MSG_GET ICAP_BITDEPTH`

4.2.1. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ENUMERATION`, then proceed to the `TWPT_RGB` test below

4.2.2. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

4.2.3. **Test**: If `TW_ENUMERATION.ItemType` is not `TWTY_UINT16`, then end with error

4.2.4. **Test**: If the `TW_ENUMERATION.ItemList` includes a value of `1`, then end with error

4.2.5. **Test**: If the `TW_ENUMERATION.ItemList` includes a value of `24`, then end with error

5. **Action**: MSG_SET ICAP_PIXELTYPE to TWPT_RGB

   5.1. **Test**: If result is not TWRC_SUCCESS, then proceed to the next test section

   5.2. **Action**: MSG_GET ICAP_BITDEPTH

      5.2.1. **Test**: If TW_CAPABILITY.ConType is not TWON_ENUMERATION, then proceed to the TWPT_RGB test below

      5.2.2. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

      5.2.3. **Test**: If TW_ENUMERATION.ItemType is not TWTY_UINT16, then end with error

      5.2.4. **Test**: If the TW_ENUMERATION.ItemList includes a value of 1, then end with error

Confirm Basic Negotiation with ICAP_XFERMECH

Make sure that ICAP_XFERMECH is working properly. Perform basic checks on how well it supports negotiation.

6. **Action**: MSG_GET ICAP_XFERMECH

   6.1. **Test**: If result is not TWRC_SUCCESS, then end with error

   6.2. **Test**: If TW_CAPABILITY.Cap is not ICAP_XFERMECH, then end with error

   6.3. **Test**: If TW_CAPABILITY.ConType is not TWON_ENUMERATION, then end with error

   6.4. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

   6.5. **Test**: If TW_ENUMERATION.ItemType is not TWTY_UINT16, then end with error

   6.6. **Test**: If TW_ENUMERATION.NumItems is less than two, then end with error

Exercise DAT_CAPABILITY

Exercise DAT_CAPABILITY operations for all TWAIN Standard capabilities (ID's with a value less than 0x8000). Ignore Vendor Custom capabilities (ID's with a value of 0x8000 or greater). Confirm correct ConType and ItemType values described in the TWAIN Specification in the chapter titled Chapter 10, "Capabilities".

7. **Action**: MSG_RESETALL

   7.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

   7.2. **Action**: Repeat this section for each enumerated value found inside of ICAP_PIXELTYPE, (testing is done for each value of ICAP_PIXELTYPE, to provide the best chance of exercising every available capability)

   7.3. **Action**: Repeat this section for Standard TWAIN array values found inside of CAP_SUPPORTEDCAPS (each Standard TWAIN capability ID is referred to as #CAP# for the rest of this section)

7.3.1. **Action**: `MSG_QUERYSUPPORT #CAP#`

7.3.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.1.2. **Test**: If `TW_CAPABILITY.Cap` is not `#CAP#`, then end with error

7.3.1.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ONEVALUE`, then end with error

7.3.1.4. **Test**: If `TW_ONEVALUE.ItemType` is not `TWTY_UINT32`, then end with error

7.3.1.5. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

7.3.1.6. **Test**: If the value of `TW_ONEVALUE.Item` doesn't match the `TWQC` values for this capability, then end with error

7.3.1.7. **Test**: If `TWQC_GET`, `TWQC_GETCURRENT` or `TWQC_GETDEFAULT` is detected, then all three must be present, if any are missing end with error

7.3.1.8. **Test**: If `TWQC_RESET` or `TWQC_SET` is detected, then both must be present, plus `TWQC_GET`, `TWQC_GETCURRENT` and `TWQC_GETDEFAULT`, if not true then end with error

7.3.2. **Action**: If `TWQC_GET` is reported, then call `MSG_GET #CAP#`

7.3.2.1. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to the next capability

7.3.2.2. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.2.3. **Test**: If `TW_CAPABILITY.Cap` is not `#CAP#`, then end with error

7.3.2.4. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

7.3.2.5. **Test**: If the value of `TW_CAPABILITY.ConType` doesn't match the Specification's `MSG_GET` container for this capability, then end with error

7.3.2.6. **Test**: If container's `ItemType` doesn't match the Specification's `ItemType` for this capability, then end with error

7.3.3. **Action**: If `TWQC_GETCURRENT` is reported, then call `MSG_GETCURRENT #CAP#`

7.3.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.3.2. **Test**: If `TW_CAPABILITY.Cap` is not `#CAP#`, then end with error

7.3.3.3. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

7.3.3.4. **Action**: If TWQC_GET was reported by MSG_QUERYSUPPORT then do the following:

7.3.3.4.1. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ENUMERATION, TWON_ONEVALUE or TWON_RANGE, then the TW_CAPABILITY.ConType for MSG_GETCURRENT must be TWTY_ONEVALUE, if not then end with error

7.3.3.4.2. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ARRAY then the TW_CAPABILITY.ConType for MSG_GETCURRENT must be TWTY_ARRAY, if not then end with error

7.3.3.4.3. **Test**: If container's ItemType for MSG_GET doesn't match container's ItemType for MSG_GETCURRENT, then end with error

7.3.4. **Action**: If TWQC_GETDEFAULT is reported, then call MSG_GETDEFAULT #CAP#

7.3.4.1. **Test**: If result is not TWRC_SUCCESS, then end with error

7.3.4.2. **Test**: If TW_CAPABILITY.Cap is not #CAP#, then end with error

7.3.4.3. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

7.3.4.4. **Action**: If TWQC_GET was reported by MSG_QUERYSUPPORT then do the following:

7.3.4.4.1. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ENUMERATION, TWON_ONEVALUE or TWON_RANGE, then the TW_CAPABILITY.ConType for MSG_GETDEFAULT must be TWTY_ONEVALUE, if not then end with error

7.3.4.4.2. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ARRAY then the TW_CAPABILITY.ConType for MSG_GETDEFAULT must be TWTY_ARRAY, if not then end with error

7.3.4.4.3. **Test**: If container's ItemType for MSG_GET doesn't match container's ItemType for MSG_GETDEFAULT, then end with error

7.3.5. **Action**: If TWQC_RESET is reported, then call MSG_RESET #CAP#

7.3.5.1. **Test**: If result is not TWRC_SUCCESS, then end with error

7.3.5.2. **Test**: If TW_CAPABILITY.Cap is not #CAP#, then end with error

7.3.5.3. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

7.3.6. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.6.1. **Test**: If `TW_CAPABILITY.ConType` for `MSG_GET` doesn't match `TW_CAPABILITY.ConType` for `MSG_RESET`, then end with error

7.3.6.2. **Test**: If container's `ItemType` for `MSG_GET` doesn't match container's `ItemType` for `MSG_RESET`, then end with error

7.3.7. **Action**: If `TWQC_SET` is reported then do the following:

7.3.7.1. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.7.1.1. **Action**: `MSG_GET #CAP#`

7.3.7.1.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.1.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GET`

7.3.7.1.2.1. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to next capability

7.3.7.1.2.2. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.2. **Action**: If `TWQC_GETCURRENT` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.7.2.1. **Action**: `MSG_GETCURRENT #CAP#`

7.3.7.2.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.2.1.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GETCURRENT`

7.3.7.2.1.3. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to next capability

7.3.7.2.1.4. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.3. **Action**: If `TWQC_GETDEFAULT` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.7.3.1. **Action**: `MSG_GETDEFAULT #CAP#`

7.3.7.3.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.3.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GETDEFAULT`

7.3.7.3.2.1. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to next capability

7.3.7.3.2.2. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.4. **Action**: If `TWQC_RESET` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.7.4.1. **Action**: `MSG_RESET #CAP#`

7.3.7.4.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.4.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_RESET`

7.3.7.4.2.1. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to next capability

7.3.7.4.2.2. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.5. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

7.3.7.5.1. **Action**: `MSG_GET #CAP#`

7.3.7.5.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

7.3.7.5.1.2. **Test**: If the container's `ItemType` is `TWTY_BOOL` and the test application has `DF_APP2` in its `TW_IDENTITY.SupportedGroups`, and the data source has `DF_DS2` in its `TW_IDENTITY.SupportedGroups`, then `TW_CAPABILITY.ConType` must be set to `TW_ENUMERATION`, if not then end with error

7.3.7.5.1.3. **Test**: If the container's `ItemType` is `TWTY_BOOL` and the test application does not have `DF_APP2` in its `TW_IDENTITY.SupportedGroups`, or the data source does not have `DF_DS2` in its `TW_IDENTITY.SupportedGroups`, then `TW_CAPABILITY.ConType` must be set to `TW_ONEVALUE`, if not then end with error

7.3.7.5.2. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ARRAY` then repeat following for each value in the array:

7.3.7.5.2.1. **Action**: `MSG_SET` the value using a `TW_ARRAY` container

7.3.7.5.2.1.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.5.2.2. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ARRAY` then do the following:

7.3.7.5.2.2.1. **Action**: `MSG_SET` the value using a `TW_ARRAY` container, setting the value to 22222 (which is expected to be an illegal value)

7.3.7.5.2.3. **Test**: If result is not `TWRC_BADVALUE` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.5.3. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ENUMERATION` then repeat following for each value in the enumeration:

7.3.7.5.4. **Action**: `MSG_SET` the value using a `TW_ENUMERATION` container

7.3.7.5.4.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

7.3.7.5.5. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ENUMERATION` then do the following:

7.3.7.5.5.1. **Action**: `MSG_SET` the current value using a `TW_ONEVALUE` container, the value must be something that did not appear in the list of valid enumerations

7.3.7.5.5.1.1. **Test**: If result is not `TWRC_BADVALUE`, then end with error

7.3.7.5.6. **Action**: If `TW_CAPABILITY.ConType` is `TWON_RANGE` then repeat the following for the `TW_RANGE.MinValue`, `TW_RANGE.CurrentValue` and `TW_RANGE.MaxValue`:

7.3.7.5.6.1. **Action**: `MSG_SET` the current value using a `TW_RANGE` container

7.3.7.5.6.1.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

Vendor Custom Capability Tests

Purpose

Exercise all of the Vendor Custom capabilities exposed by `CAP_SUPPORTEDCAPS` using the standard operations supported by `DG_CONTROL / DAT_CAPABILITY`.

Operations on capabilities (`MSG_*` values specified below) are assumed to be `DG_CONTROL / DAT_CAPABILITY`, unless otherwise stated.

Pre-Test Procedure

Open the data source manager and the data source that is to be tested. It is recommended that the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Confirm Basic Negotiation with CAP_SUPPORTEDCAPS

Make sure that `CAP_SUPPORTEDCAPS` is working properly. Perform basic checks on how well it supports negotiation.

1. **Action**: `MSG_RESETALL`

1.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

1.2. **Action**: `MSG_GET CAP_SUPPORTEDCAPS` (gets the list of capabilities to be tested)

1.2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.2.2. **Test**: If `TW_CAPABILITY.Cap` is not `CAP_SUPPORTEDCAPS`, then end with error

1.2.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ARRAY`, then end with error

1.2.4. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

1.2.5. **Test**: If `TW_ARRAY.ItemType` is not `TWTY_UINT16`, then end with error

1.2.6. **Test**: If `TW_ARRAY.NumItems` is equal to zero, then end with error

1.2.7. **Action**: Confirm the presence of the following capabilities in `TW_ARRAY.ItemList`

1.2.7.1. **Test**: If `CAP_SUPPORTEDCAPS` not found, then end with error

1.2.7.2. **Test**: If `ICAP_PIXELTYPE` not found, then end with error

Confirm Basic Negotiation with ICAP_PIXELTYPE

Make sure that `ICAP_PIXELTYPE` is working properly. Perform basic checks on how well it supports negotiation.

2. **Action**: `MSG_GET ICAP_PIXELTYPE`

2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

2.2. **Test**: If `TW_CAPABILITY.Cap` is not `ICAP_PIXELTYPE`, then end with error

2.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ENUMERATION`, then end with error

2.4. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

2.5. **Test**: If `TW_ENUMERATION.ItemType` is not `TWTY_UINT16`, then end with error

2.6. **Test**: If `TW_ENUMERATION.NumItems` is equal to zero, then end with error

Exercise DAT_CAPABILITY

Exercise `DAT_CAPABILITY` operations for all Vendor Custom capabilities (ID's with a value of `0x8000` or greater). Ignore TWAIN Standard capabilities (ID's with a value less than `0x8000`).

3. **Action**: Repeat this section for each enumerated value found inside of `ICAP_PIXELTYPE`, (testing is done for each value of `ICAP_PIXELTYPE`, to provide the best chance of exercising every available capability)

3.1. **Action**: Repeat this section for each Vendor Custom TWAIN array value found inside of `CAP_SUPPORTEDCAPS` (each Vendor Custom capability ID is referred to as `#CAP#` for the rest of this section)

3.1.1. **Action**: `MSG_QUERYSUPPORT #CAP#`

3.1.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.1.2. **Test**: If `TW_CAPABILITY.Cap` is not `#CAP#`, then end with error

3.1.1.3. **Test**: If `TW_CAPABILITY.ConType` is not `TWON_ONEVALUE`, then end with error

3.1.1.4. **Test**: If `TW_ONEVALUE.ItemType` is not `TWTY_UINT32`, then end with error

3.1.1.5. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

3.1.2. **Action**: If TWQC_GET is reported, then call MSG_GET #CAP#

3.1.2.1. **Test**: If result is TWRC_FAILURE / TWCC_CAPSEQERROR, then skip to the next capability

3.1.2.2. **Test**: If result is not TWRC_SUCCESS, then end with error

3.1.2.3. **Test**: If TW_CAPABILITY.Cap is not #CAP#, then end with error

3.1.2.4. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

3.1.3. **Action**: If TWQC_GETCURRENT is reported, then call MSG_GETCURRENT #CAP#

3.1.3.1. **Test**: If result is not TWRC_SUCCESS, then end with error

3.1.3.2. **Test**: If TW_CAPABILITY.Cap is not #CAP#, then end with error

3.1.3.3. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

3.1.3.4. **Action**: If TWQC_GET was reported by MSG_QUERYSUPPORT then do the following:

3.1.3.4.1. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ENUMERATION, TWON_ONEVALUE or TWON_RANGE, then the TW_CAPABILITY.ConType for MSG_GETCURRENT must be TWTY_ONEVALUE, if not then end with error

3.1.3.4.2. **Test**: If the TW_CAPABILITY.ConType for MSG_GET was TWON_ARRAY then the TW_CAPABILITY.ConType for MSG_GETCURRENT must be TWTY_ARRAY, if not then end with error

3.1.3.4.3. **Test**: If container's ItemType for MSG_GET doesn't match container's ItemType for MSG_GETCURRENT, then end with error

3.1.4. **Action**: If TWQC_GETDEFAULT is reported, then call MSG_GETDEFAULT #CAP#

3.1.4.1. **Test**: If result is not TWRC_SUCCESS, then end with error

3.1.4.2. **Test**: If TW_CAPABILITY.Cap is not #CAP#, then end with error

3.1.4.3. **Test**: If TW_CAPABILITY.hContainer is not a valid TW_HANDLE value, then end with error

3.1.4.4. **Action**: If TWQC_GET was reported by MSG_QUERYSUPPORT then do the following:

3.1.4.4.1. **Test**: If the `TW_CAPABILITY.ConType` for `MSG_GET` was `TWON_ENUMERATION`, `TWON_ONEVALUE` or `TWON_RANGE`, then the `TW_CAPABILITY.ConType` for `MSG_GETDEFAULT` must be `TWTY_ONEVALUE`, if not then end with error

3.1.4.4.2. **Test**: If the `TW_CAPABILITY.ConType` for `MSG_GET` was `TWON_ARRAY` then the `TW_CAPABILITY.ConType` for `MSG_GETDEFAULT` must be `TWTY_ARRAY`, if not then end with error

3.1.4.4.3. **Test**: If container's `ItemType` for `MSG_GET` doesn't match container's `ItemType` for `MSG_GETDEFAULT`, then end with error

3.1.5. **Action**: If `TWQC_RESET` is reported, then call `MSG_RESET #CAP#`

3.1.5.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.5.2. **Test**: If `TW_CAPABILITY.Cap` is not `#CAP#`, then end with error

3.1.5.3. **Test**: If `TW_CAPABILITY.hContainer` is not a valid `TW_HANDLE` value, then end with error

3.1.5.4. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.5.4.1. **Test**: If `TW_CAPABILITY.ConType` for `MSG_GET` doesn't match `TW_CAPABILITY.ConType` for `MSG_RESET`, then end with error

3.1.5.4.2. **Test**: If container's `ItemType` for `MSG_GET` doesn't match container's `ItemType` for `MSG_RESET`, then end with error

3.1.6. **Action**: If `TWQC_SET` is reported then do the following:

3.1.6.1. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.6.1.1. **Action**: `MSG_GET #CAP#`

3.1.6.1.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.1.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GET`

3.1.6.1.2.1. **Test**: If result is `TWRC_FAILURE / TWCC_CAPSEQERROR`, then skip to next capability

3.1.6.1.2.2. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.2. **Action**: If `TWQC_GETCURRENT` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.6.2.1. **Action**: `MSG_GETCURRENT #CAP#`

3.1.6.2.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.2.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GETCURRENT`

3.1.6.2.2.1. **Test**: If result is `TWRC_FAILURE` / `TWCC_CAPSEQERROR`, then skip to next capability

3.1.6.2.2.2. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.3. **Action**: If `TWQC_GETDEFAULT` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.6.3.1. **Action**: `MSG_GETDEFAULT #CAP#`

3.1.6.3.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.3.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_GETDEFAULT`

3.1.6.3.2.1. **Test**: If result is `TWRC_FAILURE /` `TWCC_CAPSEQERROR`, then skip to next capability

3.1.6.3.2.2. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.4. **Action**: If `TWQC_RESET` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.6.4.1. **Action**: `MSG_RESET #CAP#`

3.1.6.4.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.4.2. **Action**: `MSG_SET` with `TW_CAPABILITY` from `MSG_RESET`

3.1.6.4.2.1. **Test**: If result is `TWRC_FAILURE /` `TWCC_CAPSEQERROR`, then skip to next capability

3.1.6.4.2.2. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.5. **Action**: If `TWQC_GET` was reported by `MSG_QUERYSUPPORT` then do the following:

3.1.6.5.1. **Action**: `MSG_GET #CAP#`

3.1.6.5.1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

3.1.6.5.1.2. **Test**: If the container's `ItemType` is `TWTY_BOOL` and the test application has `DF_APP2` in its `TW_IDENTITY.SupportedGroups`, and the data source has `DF_DS2` in its `TW_IDENTITY.SupportedGroups`, then `TW_CAPABILITY.ConType` must be set to `TW_ENUMERATION`, if not then end with error

3.1.6.5.1.3. **Test**: If the container's `ItemType` is `TWTY_BOOL` and the test application does not have `DF_APP2` in its `TW_IDENTITY.SupportedGroups`, or the data source does not have `DF_DS2` in its `TW_IDENTITY.SupportedGroups`, then `TW_CAPABILITY.ConType` must be set to `TW_ONEVALUE`, if not then end with error

3.1.6.5.2. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ARRAY` then repeat following for each value in the array:

3.1.6.5.2.1. **Action**: `MSG_SET` the value using a `TW_ARRAY` container

3.1.6.5.2.1.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.5.3. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ARRAY` then do the following:

3.1.6.5.3.1. **Action**: `MSG_SET` the value using a `TW_ARRAY` container, setting the value to 22222 (which is expected to be an illegal value)

3.1.6.5.3.1.1. **Test**: If result is not `TWRC_BADVALUE` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.5.4. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ENUMERATION` then repeat following for each value in the enumeration:

3.1.6.5.5. **Action**: `MSG_SET` the value using a `TW_ENUMERATION` container

3.1.6.5.5.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

3.1.6.5.6. **Action**: If `TW_CAPABILITY.ConType` is `TWON_ENUMERATION` then do the following:

3.1.6.5.6.1. **Action**: `MSG_SET` the current value using a `TW_ONEVALUE` container, the value must be something that did not appear in the list of valid enumerations

3.1.6.5.6.1.1. **Test**: If result is not `TWRC_BADVALUE`, then end with error

3.1.6.5.7. **Action**: If `TW_CAPABILITY.ConType` is `TWON_RANGE` then repeat the following for the `TW_RANGE.MinValue`, `TW_RANGE.CurrentValue` and `TW_RANGE.MaxValue`:

3.1.6.5.7.1. **Action**: `MSG_SET` the current value using a `TW_RANGE` container

3.1.6.5.7.1.1. **Test**: If result is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

Status Return Tests

Purpose

Confirm that the expected status return is reported by certain operations.

This is not an exhaustive test of all possible Status Returns.

Pre-Test Procedure

Open the data source manager and the data source that is to be tested. It is recommended that the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Check Status Returns for DAT_IMAGENATIVEXFER and DAT_IMAGEMEMXFER

Confirm that `DAT_IMAGENATIVEXFER` and `DAT_IMAGEMEMXFER` both return the correct status returns in various error conditions.

1. **Action**: In State 4 (after `MSG_OPENDS`, but before calling `MSG_ENABLEDS`)…
   - 1.1. Confirm that the proper statuses are returned for bad protocols and attempts to perform image transfers in State 4.
   - 1.2. **Action**: Call `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_SET`
     - 1.2.1. **Test**: If result is not `TWRC_FAILURE / TWCC_BADPROTOCOL`, then end with error
   - 1.3. **Action**: Call `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`
     - 1.3.1. **Test**: If result is not `TWRC_FAILURE / TWCC_SEQERROR`, then end with error
   - 1.4. **Action**: Call `DG_IMAGE / DAT_IMAGEMEMXFER / MSG_SET`
     - 1.4.1. **Test**: If result is not `TWRC_FAILURE / TWCC_BADPROTOCOL`, then end with error
   - 1.5. **Action**: Call `DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET`
     - 1.5.1. **Test**: If result is not `TWRC_FAILURE / TWCC_SEQERROR`, then end with error

Check Status Returns for DAT_IMAGELAYOUT

Confirm that `DAT_IMAGELAYOUT` returns the correct status returns in various error conditions.

2. **Action**: Call `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = TRUE`
   - 2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error
   - 2.2. **Action**: Call `DG_IMAGE / DAT_IMAGELAYOUT / MSG_GET`
     - 2.2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error
   - 2.3. **Action**: Call `DG_IMAGE / DAT_IMAGELAYOUT / MSG_SET` using the `TW_IMAGELAYOUT` values from the previous `MSG_GET` call
     - 2.3.1. **Test**: If result is not `TWRC_FAILURE / TWRC_SEQERROR`, then end with error
   - 2.4. **Action**: Call `DG_IMAGE / DAT_IMAGELAYOUT / MSG_RESET`
     - 2.4.1. **Test**: If result is not `TWRC_FAILURE / TWCC_SEQERROR`, then end with error

Check Status Returns for DAT_CAPABILITY

Confirm that `DAT_CAPABILITY` returns the correct status returns in various error conditions.

3. **Action**: Call `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = TRUE`

   3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

   3.2. **Action**: `MSG_GET CAP_SUPPORTEDCAPS`

      3.2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

   3.3. **Action**: `MSG_GET CAP_EXTENDEDCAPS`

      3.3.1. **Test**: If result is not `TWRC_SUCCESS` or the `TW_ARRAY` is empty, then skip any checks of `CAP_EXTENDEDCAPS` referenced in the rest of this section

   3.4. **Action**: For each value found in `CAP_SUPPORTEDCAPS` that is not in `CAP_EXTENDEDCAPS` do the following sections (each capability ID is referred to as `#CAP#` for the rest of this section):

      3.4.1. **Action**: `MSG_GET #CAP#`

         3.4.1.1. **Test**: If result is not `TWRC_SUCCESS`, then skip to next capability

      3.4.2. **Action**: `MSG_SET #CAP#` with results of previous `MSG_GET`

         3.4.2.1. **Test**: If result is `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

      3.4.3. **Action**: `MSG_RESET #CAP#`

         3.4.3.1. **Test**: If result is `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, then end with error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

Stress Tests

Purpose

Stress aspects of data sources that have been reported as common problems.

Pre-Test Procedure

Open the data source manager. It is required that when opened the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Open and Close the Data Source Multiple Times

Confirm that the data source can open and close multiple times. This tests for crashes.

1. **Action**: Repeat this section twenty (20) times

    1.1. Confirm that the data source can successfully open and close repeated times from a single instance of an application.

    1.2. **Action**: Call `DG_CONTROL / DAT_IDENTITY / MSG_OPENDS`

        1.2.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.3. **Action**: Call `DG_CONTROL / DAT_IDENTITY / MSG_CLOSEDS`

        1.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

Non-UI Image Transfer Tests

Purpose

Confirm that multiple `MSG_ENABLEDS` and `MSG_DISABLEDS` calls can be made in the context of one `MSG_OPENDS / MSG_CLOSEDS`. This test focuses on image capture with no UI, verifying that the Application does not have to close the driver after capturing images.

Pre-Test Procedure

Open the data source manager and the data source that is to be tested. It is recommended that the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Exercise DAT_IMAGENATIVEXFER

This test issues multiple image transfer sessions using `DAT_IMAGENATIVEXFER`. It is performed for all available image sources (unspecified, flatbed and/or ADF). Only one image is transferred per session.

1. **Action**: `MSG_RESETALL`

    1.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

    1.2. **Action**: `MSG_GET CAP_SUPPORTEDCAPS` (get the list of capabilities to be tested)

    1.3. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_NATIVE`

1.4. **Action**: `MSG_GETCURRENT ICAP_XFERMECH`

1.5. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

1.6. **Test**: If value is not `TWSX_NATIVE`, end with an error.

1.7. **Action**: If `CAP_FEEDERENABLED` is `TRUE`, set `CAP_AUTOFEED` to `TRUE`

1.8. **Action**: `MSG_SET CAP_DUPLEXENABLED` to `FALSE`

1.9. **Action**: `MSG_SET CAP_XFERCOUNT` to 1

1.10. **Action**: Do the following for each supported `ICAP_PIXELTYPE`

- 1.10.1. **Action**: `MSG_SET ICAP_PIXELTYPE`
- 1.10.2. **Action**: `MSG_GET ICAP_BITDEPTH`
- 1.10.3. **Action**: Do the following for each supported `ICAP_BITDEPTH`
  - 1.10.3.1. **Action**: `MSG_SET ICAP_BITDEPTH`
  - 1.10.3.2. **Action**: Do the following for the minimum, maximum and 300 (or nearest) resolution values.
    - 1.10.3.2.1. **Action**: `MSG_SET ICAP_XRESOLUTION` and `ICAP_YRESOLUTION`
    - 1.10.3.2.2. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`
    - 1.10.3.2.3. **Test**: If return code is not `TWRC_SUCCESS`, end with an error
    - 1.10.3.2.4. **Action**: Wait for `MSG_XFERREADY`
    - 1.10.3.2.5. **Action**: `MSG_GET ICAP_XFERMECH`
    - 1.10.3.2.6. **Test**: If return code is not `TWRC_SUCCESS`, end with an error
    - 1.10.3.2.7. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`
    - 1.10.3.2.8. **Test**: If return code is not `TWRC_XFERDONE`, end with an error
    - 1.10.3.2.9. **Test**: If the handle does not point to a valid image, end with an error
    - 1.10.3.2.10. **Test**: If the bit depth of the image is not what was requested, end with an error
    - 1.10.3.2.11. **Action**: Free handle returned by `DAT_IMAGENATIVEXFER`

1.10.3.2.12. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

1.10.3.2.13. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

1.10.3.2.14. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

Exercise DAT_IMAGEMEMXFER

This test issues multiple image transfer sessions using `DAT_IMAGEMEMXFER`. It is performed for all available image sources (unspecified, flatbed and/or ADF). Only one image is transferred per session. The preferred size specified by the data source is used to transfer each strip.

2. **Action**: `MSG_RESETALL`

2.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.2. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_MEMORY`

2.3. **Action**: `MSG_GETCURRENT ICAP_XFERMECH`

2.4. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.5. **Test**: If value is not `TWSX_MEMORY`, end with an error

2.6. **Action**: If `CAP_FEEDERENABLED` is `TRUE`, set `CAP_AUTOFEED` to `TRUE`

2.7. **Action**: `MSG_SET CAP_DUPLEXENABLED` to `FALSE`

2.8. **Action**: `MSG_SET CAP_XFERCOUNT` to `1`

2.9. **Action**: Do the following for each supported `ICAP_PIXELTYPE`

2.9.1. **Action**: `MSG_SET ICAP_PIXELTYPE`

2.9.2. **Action**: `MSG_GET ICAP_BITDEPTH`

2.9.3. **Action**: Do the following for each supported `ICAP_BITDEPTH`

2.9.3.1. **Action**: `MSG_SET ICAP_BITDEPTH`

2.9.3.2. **Action**: `MSG_GET ICAP_COMPRESSION`

2.9.3.3. **Action**: Do the following for each supported `ICAP_COMPRESSION`

2.9.3.3.1. **Action**: `MSG_SET ICAP_COMPRESSION`

2.9.3.3.2. **Action**: Do the following for the minimum, maximum and 300 (or nearest) resolution values.

2.9.3.3.2.1. **Action**: `MSG_SET ICAP_XRESOLUTION` and `ICAP_YRESOLUTION`

2.9.3.3.2.2. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

2.9.3.3.2.3. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.9.3.3.2.4. **Action**: Wait for `MSG_XFERREADY`

2.9.3.3.2.5. **Action**: `MSG_GET ICAP_XFERMECH`

2.9.3.3.2.6. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.9.3.3.2.7. **Action**: `DG_CONTROL / DAT_SETUPMEMXFER / MSG_GET`

2.9.3.3.2.8. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.9.3.3.2.9. **Action**: `DG_IMAGE / DAT_IMAGEMEMXFER / MSG_GET` with the preferred buffer size

2.9.3.3.2.10. **Test**: if the return code is `TWRC_SUCCESS`, repeat previous step

2.9.3.3.2.11. **Test**: if the return code is not `TWRC_XFERDONE`, end with an error

2.9.3.3.2.12. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.9.3.3.2.13. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

2.9.3.3.2.14. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

Exercise DAT_IMAGEFILEXFER

This test issues multiple image transfer sessions using `DAT_IMAGEFILEXFER`. It is performed for all available image sources (unspecified, flatbed and/or ADF). Only one image is transferred per session. The preferred size specified by the data source is used to transfer each strip.

3. **Action**: `MSG_RESETALL`

   3.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

   3.2. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_MEMORY`

   3.3. **Test**: If return code is `TWRC_SUCCESS / TWCC_BADVALUE`, skip to section 4

3.4. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

3.5. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_FILE`

3.6. **Action**: If `CAP_FEEDERENABLED` is `TRUE`, set `CAP_AUTOFEED` to `TRUE`

3.7. **Action**: `MSG_SET CAP_DUPLEXENABLED` to `FALSE`

3.8. **Action**: `MSG_SET CAP_XFERCOUNT` to `1`

3.9. **Action**: `MSG_GET ICAP_IMAGEFILEFORMAT`

3.10. **Action**: Do the following for each supported `ICAP_IMAGEFILEFORMAT`

- 3.10.1. **Action**: `MSG_SET ICAP_IMAGEFILEFORMAT`
- 3.10.2. **Action**: `MSG_GET ICAP_PIXELTYPE`
- 3.10.3. **Action**: Do the following for each supported `ICAP_PIXELTYPE`
  - 3.10.3.1. **Action**: `MSG_SET ICAP_PIXELTYPE`
  - 3.10.3.2. **Action**: `MSG_GET ICAP_BITDEPTH`
  - 3.10.3.3. **Action**: Do the following for each supported `ICAP_BITDEPTH`
    - 3.10.3.3.1. **Action**: `MSG_SET ICAP_BITDEPTH`
    - 3.10.3.3.2. **Action**: `MSG_GET ICAP_COMPRESSION`
    - 3.10.3.3.3. **Action**: Do the following for each supported `ICAP_COMPRESSION`
      - 3.10.3.3.3.1. **Action**: `MSG_SET ICAP_COMPRESSION`
      - 3.10.3.3.3.2. **Action**: Do the following for the minimum, maximum and 300 (or nearest) resolution values.
        - 3.10.3.3.3.2.1. **Action**: `MSG_SET ICAP_XRESOLUTION` and `ICAP_YRESOLUTION`
        - 3.10.3.3.3.2.2. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`
        - 3.10.3.3.3.2.3. **Test**: If return code is not `TWRC_SUCCESS`, end with an error
        - 3.10.3.3.3.2.4. **Action**: Wait for `MSG_XFERREADY`

3.10.3.3.3.2.5. **Action**: `MSG_GET ICAP_XFERMECH`

3.10.3.3.3.2.6. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

3.10.3.3.3.2.7. **Action**: `DG_CONTROL / DAT_SETUPFILEXFER / MSG_SET`

3.10.3.3.3.2.8. **Action**: `DG_IMAGE / DAT_IMAGEFILEXFER / MSG_GET`

3.10.3.3.3.2.9. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

3.10.3.3.3.2.10. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

3.10.3.3.3.2.11. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

3.10.3.3.3.2.12. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

UI Image Transfer Tests

Purpose

Confirm that multiple `MSG_ENABLEDS` and `MSG_DISABLEDS` calls can be made in the context of one `MSG_OPENDS / MSG_CLOSEDS`. This test focuses on image capture with the UI, verifying that the Application does not have to close the driver after capturing images.

Procedure

These tests are identical to the “Non-UI Image Transfer Tests”, except that the value of `ShowUI` is set to `TRUE` instead of `FALSE`.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

CAP_XFERCOUNT Tests

Purpose

Confirm that when the data source accepts various values for `CAP_XFERCOUNT`, that it returns the specified number of images. Test both flatbed and document feeders.

Pre-Test Procedure

Open the data source manager and the data source that is to be tested. It is recommended that the data source is in the state it would be in after being installed (e.g., no saved settings from previous sessions), to make the test more reproducible.

When performing this test on Windows Vista or later, Macintosh OS X or Linux it must be successfully completed using both a native 32-bit and a native 64-bit data source.

Test Flatbed Scanning

This test sets `CAP_XFERCOUNT` to `0`, `1` and `-1` for a flatbed scanner. It expects an error for the value `0`, and only one image to be transferred per scanning session for the values `1` and `-1`.

1. **Action**: `MSG_RESETALL`

    1.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

    1.2. **Action**: `MSG_SET CAP_FEEDERENABLED` to `FALSE`

    1.3. **Test**: If return is `TWRC_FAILURE / TWCC_BADVALUE`, then scanner does not have a flatbed, proceed to the Test Document Feeder Scanning section

    1.4. **Test**: If return is not `TWRC_SUCCESS` and not `TWRC_FAILURE / TWCC_CAPUNSUPPORTED`, end with error

    1.5. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_NATIVE`

    1.5.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

    1.6. **Action**: `MSG_SET CAP_XFERCOUNT` to `0`

    1.6.1. **Test**: If return code is not `TWRC_FAILURE / TWCC_BADVALUE`, end with an error

    1.7. **Action**: `MSG_SET CAP_XFERCOUNT` to `1`

    1.7.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

    1.8. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

1.8.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

1.9. **Action**: Wait for MSG_XFERREADY

1.10. **Action**: DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

1.10.1. **Test**: If return code is not TWRC_XFERDONE, end with an error

1.11. **Action**: DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER

1.11.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

1.11.2. **Test**: If TW_PENDINGXFERS.Count is not 0, end with error

1.12. **Action**: DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS

1.12.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

1.13. **Action**: MSG_SET CAP_XFERCOUNT to -1

1.13.1. **Test**: If return is not TWRC_SUCCESS, end with error

1.14. **Action**: DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS with ShowUI = FALSE and ModalUI = FALSE

1.14.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

1.15. **Action**: Wait for MSG_XFERREADY

1.16. **Action**: DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET

1.16.1. **Test**: If return code is not TWRC_XFERDONE, end with an error

1.17. **Action**: DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER

1.17.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

1.17.2. **Test**: If TW_PENDINGXFERS.Count is not 0, end with error

1.18. **Action**: DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS

1.18.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

Test Document Feeder Scanning

This test issues multiple image transfer sessions using DAT_IMAGENATIVEXFER. It is performed for all available image sources (unspecified, flatbed and/or ADF). Only one image is transferred per session.

2. **Action**: MSG_RESETALL

2.1. **Test**: If return code is not TWRC_SUCCESS, end with an error

2.2. **Action**: MSG_SET CAP_FEEDERENABLED to TRUE

2.3. **Test**: If return is TWRC_FAILURE / TWCC_BADVALUE or TWRC_FAILURE / TWCC_CAPUNSUPPORTED, then scanner does not have a Document Feeder, skip the rest of this section

2.4. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.5. **Action**: `MSG_SET ICAP_XFERMECH` to `TWSX_NATIVE`

2.5.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.6. **Action**: `MSG_SET CAP_XFERCOUNT` to `3`

2.6.1. **Test**: If return is not `TWRC_SUCCESS` or `TWRC_CHECKSTATUS`, end with error

2.7. **Action**: `MSG_GET CAP_XFERCOUNT`

2.7.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.7.2. **Test**: If value is not equal to `3` do this section

2.7.2.1. **Action**: `MSG_SET CAP_XFERCOUNT` to `0`

2.7.2.1.1. **Test**: If return code is not `TWRC_FAILURE / TWCC_BADVALUE`, end with an error

2.7.2.2. **Action**: `MSG_SET CAP_XFERCOUNT` to `1`

2.7.2.2.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.7.2.3. **Action**: Ask user to place one sheet of paper in the document feeder

2.7.2.4. `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

2.7.2.4.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.2.5. **Action**: Wait for `MSG_XFERREADY`

2.7.2.6. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

2.7.2.6.1. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

2.7.2.7. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.7.2.7.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.2.7.2. **Test**: If `TW_PENDINGXFERS.Count` is not `0`, end with error

2.7.2.8. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

2.7.2.8.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.2.9. **Action**: `MSG_SET CAP_XFERCOUNT` to `-1`

2.7.2.9.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.7.2.10. **Action**: Ask user to place one sheet of paper in the document feeder

2.7.2.11. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

2.7.2.11.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.2.12. **Action**: Wait for `MSG_XFERREADY`

2.7.2.13. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

2.7.2.13.1. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

2.7.2.14. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.7.2.14.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.2.14.2. **Test**: If `TW_PENDINGXFERS.Count` is not `0`, end with error

2.7.2.15. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

2.7.2.15.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3. **Test**: If value is equal to `3` do this section

2.7.3.1. **Action**: Ask user to place three sheets of paper in the document feeder

2.7.3.2. **Action**: `MSG_SET CAP_DUPLEXENABLED` to `FALSE`

2.7.3.2.1. **Test**: If return code is not `TWRC_SUCCESS` or `TWRC_FAILURE / TWCC_CAPUNSUPPORTED`, end with error

2.7.3.3. **Action**: `MSG_SET CAP_XFERCOUNT` to `0`

2.7.3.3.1. **Test**: If return code is not `TWRC_FAILURE / TWCC_BADVALUE`, end with an error

2.7.3.4. **Action**: `MSG_SET CAP_XFERCOUNT` to `1`

2.7.3.4.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.7.3.5. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

2.7.3.5.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.6. **Action**: Wait for `MSG_XFERREADY`

2.7.3.7. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

2.7.3.7.1. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

2.7.3.8. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.7.3.8.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.8.2. **Test**: If `TW_PENDINGXFERS.Count` is not `0`, end with error

2.7.3.9. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

2.7.3.9.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.10. **Action**: `MSG_SET CAP_XFERCOUNT` to `-1`

2.7.3.10.1. **Test**: If return is not `TWRC_SUCCESS`, end with error

2.7.3.11. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS` with `ShowUI = FALSE` and `ModalUI = FALSE`

2.7.3.11.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.12. **Action**: Wait for `MSG_XFERREADY`

2.7.3.13. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

2.7.3.13.1. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

2.7.3.14. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.7.3.14.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.14.2. **Test**: If `TW_PENDINGXFERS.Count` is not `1` or `-1`, end with error

2.7.3.15. **Action**: `DG_IMAGE / DAT_IMAGENATIVEXFER / MSG_GET`

2.7.3.15.1. **Test**: If return code is not `TWRC_XFERDONE`, end with an error

2.7.3.16. **Action**: `DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER`

2.7.3.16.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

2.7.3.16.2. **Test**: If `TW_PENDINGXFERS.Count` is not `0`, end with error

2.7.3.17. **Action**: `DG_CONTROL / DAT_USERINTERFACE / MSG_DISABLEDS`

2.7.3.17.1. **Test**: If return code is not `TWRC_SUCCESS`, end with an error

Post-Test Procedure

When testing is completed, close the data source and the data source manager.

Version Tests

Purpose

Confirm that the data sources responds correctly to different TWAIN versions of data source manager and application.

Pre-Test Procedure

Close the data source manager.

Attempt to scan Multiple Times

Confirm that the data source can respond correctly to different TWAIN version of application and data source manager by attempting to scan using different setups. This tests for hangs and crashes. Use Memory transfer if available. Scan one image in simplex without UI. Testing with old DSM is only for 32-bit data sources only.

1. **Action**: `MSG_OPENDSM` using old DSM as TWAIN version 1.9 application, with `DF_APP2` set,

   1.1. **Action**: Attempt to scan

   1.2. **Test**: Confirm that the scan succeeds without hanging.

   1.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

   1.4. **Action**: `MSG_CLOSEDSM`

2. **Action**: `MSG_OPENDSM` using old DSM as TWAIN version 2.x application, with `DF_APP2` not set,

   2.1. **Action**: Attempt to scan

2.2. **Test**: Confirm that the scan succeeds without hanging.

2.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

2.4. **Action**: `MSG_CLOSEDSM`

3. **Action**: `MSG_OPENDSM` using old DSM as TWAIN version 2.x application, with `DF_APP2` set,

   3.1. **Action**: Attempt to scan

   3.2. **Test**: Confirm that the scan succeeds without hanging.

   3.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

   3.4. **Action**: `MSG_CLOSEDSM`

4. **Action**: `MSG_OPENDSM` using TWAIN 2 DSM as TWAIN version 1.9 application, with `DF_APP2` set,

   4.1. **Action**: Attempt to scan

   4.2. **Test**: Confirm that the scan succeeds without hanging.

   4.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

   4.4. **Action**: `MSG_CLOSEDSM`

5. **Action**: `MSG_OPENDSM` using TWAIN 2 DSM as TWAIN version 2.x application, with `DF_APP2` not set,

   5.1. **Action**: Attempt to scan

   5.2. **Test**: Confirm that the scan succeeds without hanging.

   5.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

   5.4. **Action**: `MSG_CLOSEDSM`

6. **Action**: `MSG_OPENDSM` using TWAIN 2 DSM as TWAIN version 1.9 application, with `DF_APP2` not set,

   6.1. **Action**: Attempt to scan

   6.2. **Test**: Confirm that the scan succeeds without hanging.

   6.3. **Test**: If the application does not receive `MSG_XFERREADY`, then end with error

   6.4. **Action**: `MSG_CLOSEDSM`

Post-Test Procedure

Nothing to do.

Verify Values For MSG_RESETALL and MSG_RESET

Purpose

Confirm that the indicated capabilities have the values required by the Specification after a `DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL` is applied to the entire driver, or a `DG_CONTROL / DAT_CAPABILITY / MSG_RESET` is applied to a single capability.

Pre-Test Procedure

Open the data source manager and the data source that is to be tested.

Test MSG_RESETALL and MSG_RESET

Make sure that `MSG_RESETALL` results in the following values for the indicated capabilities.

1. **Action**: `DG_CONTROL / DAT_CAPABILITY / MSG_RESETALL`

    1.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.2. **Action**: `MSG_GETCURRENT ACAP_XFERMECH`

        1.2.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

        1.2.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWSX_NATIVE`, then end with error

        1.2.3. **Action**: `MSG_RESET ACAP_XFERMECH`

            1.2.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

            1.2.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWSX_NATIVE`, then end with error

    1.3. **Action**: `MSG_GETCURRENT CAP_AUTHOR`

        1.3.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

        1.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_STRING128`, or the value is not an empty string, then end with error

        1.3.3. **Action**: `MSG_RESET CAP_AUTHOR`

            1.3.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

            1.3.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_STRING128`, or the value is not an empty string, then end with error

    1.4. **Action**: `MSG_GETCURRENT CAP_AUTOFEED`

        1.4.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.4.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.4.3. **Action**: `MSG_RESET CAP_AUTOFEED`

1.4.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.4.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.5. **Action**: `MSG_GETCURRENT CAP_AUTOMATICCAPTURE`

1.5.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.5.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.5.3. **Action**: `MSG_RESET CAP_AUTOMATICCAPTURE`

1.5.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.5.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.6. **Action**: `MSG_GETCURRENT CAP_CAMERSIDE`

1.6.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.6.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWCS_BOTH`, then end with error

1.6.3. **Action**: `MSG_RESET CAP_CAMERSIDE`

1.6.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.6.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWCS_BOTH`, then end with error

1.7. **Action**: `MSG_GETCURRENT CAP_CAPTION`

1.7.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.7.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_STRING255`, or the value is not an empty string, then end with error

1.7.3. **Action**: `MSG_RESET CAP_CAPTION`

1.7.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.7.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_STRING255`, or the value is not an empty string, then end with error

1.8. **Action**: `MSG_GETCURRENT CAP_CLEARPAGE`

1.8.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.8.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.8.3. **Action**: `MSG_RESET CAP_CLEARPAGE`

1.8.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.8.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.9. **Action**: `MSG_GETCURRENT CAP_DEVICEEVENT`

1.9.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.9.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.9.3. **Action**: `MSG_RESET CAP_DEVICEEVENT`

1.9.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.9.3.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.10. **Action**: `MSG_GETCURRENT CAP_DOUBLEFEEDDETECTION`

1.10.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.10.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.10.3. **Action**: `MSG_RESET CAP_DOUBLEFEEDDETECTION`

1.10.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.10.3.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.11. **Action**: `MSG_GETCURRENT CAP_ENDORSER`

1.11.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.11.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT32`, or the value is not `1`, then end with error

1.11.3. **Action**: `MSG_RESET CAP_ENDORSER`

1.11.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.11.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT32`, or the value is not `1`, then end with error

1.12. **Action**: `MSG_GETCURRENT CAP_FEEDERPREP`

1.12.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.12.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.12.3. **Action**: `MSG_RESET CAP_FEEDERPREP`

1.12.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.12.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.13. **Action**: `MSG_GETCURRENT CAP_FEEDPAGE`

1.13.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.13.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.13.3. **Action**: `MSG_RESET CAP_FEEDPAGE`

1.13.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.13.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.14. **Action**: `MSG_GETCURRENT CAP_INDICATORS`

1.14.1. **Test**: If the result is not `TWRC_SUCCESS`, then end with error

1.14.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.14.3. **Action**: `MSG_RESET CAP_INDICATORS`

1.14.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.14.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.15. **Action**: `MSG_GETCURRENT CAP_INDICATORS`

1.15.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.15.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.15.3. **Action**: `MSG_RESET CAP_INDICATORS`

1.15.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.15.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.16. **Action**: `MSG_GETCURRENT CAP_JOBCONTROL`

1.16.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.16.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWJC_NONE`, then end with error

1.16.3. **Action**: `MSG_RESET CAP_JOBCONTROL`

1.16.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.16.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWJC_NONE`, then end with error

1.17. **Action**: `MSG_GETCURRENT CAP_MICRENABLED`

1.17.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.17.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.17.3. **Action**: `MSG_RESET CAP_MICRENABLED`

1.17.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.17.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.18. **Action**: `MSG_GETCURRENT CAP_PAPERHANDLING`

1.18.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.18.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPH_NORMAL`, then end with error

1.18.3. **Action**: `MSG_RESET CAP_PAPERHANDLING`

1.18.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.18.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPH_NORMAL`, then end with error

1.19. **Action**: `MSG_GETCURRENT CAP_PRINTERENABLED`

1.19.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.19.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.19.3. **Action**: `MSG_RESET CAP_PRINTERENABLED`

1.19.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.19.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.20. **Action**: `MSG_GETCURRENT CAP_PRINTERINDEX`

1.20.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.20.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT32`, or the value is not `1`, then end with error

1.20.3. **Action**: `MSG_RESET CAP_PRINTERINDEX`

1.20.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.20.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT32`, or the value is not `1`, then end with error

1.21. **Action**: `MSG_GETCURRENT CAP_REACQUIREALLOWED`

1.21.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.21.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.21.3. **Action**: `MSG_RESET CAP_REACQUIREALLOWED`

1.21.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.21.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.22. **Action**: `MSG_GETCURRENT CAP_SEGMENTED`

1.22.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.22.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWSG_NONE`, then end with error

1.22.3. **Action**: `MSG_RESET CAP_SEGMENTED`

1.22.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.22.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWSG_NONE`, then end with error

1.23. **Action**: `MSG_GETCURRENT CAP_TIMEBEFOREFIRSTCAPTURE`

1.23.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.23.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.23.3. **Action**: `MSG_RESET CAP_TIMEBEFOREFIRSTCAPTURE`

1.23.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.23.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.24. **Action**: `MSG_GETCURRENT CAP_TIMEBETWEENCAPTURES`

1.24.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.24.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.24.3. **Action**: `MSG_RESET CAP_TIMEBETWEENCAPTURES`

1.24.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.24.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT32`, or the value is not `0`, then end with error

1.25. **Action**: `MSG_GETCURRENT CAP_THUMBNAILSENABLED`

  1.25.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

  1.25.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

  1.25.3. **Action**: `MSG_RESET CAP_THUMBNAILSENABLED`

    1.25.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.25.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.26. **Action**: `MSG_GETCURRENT CAP_XFERCOUNT`

  1.26.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

  1.26.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT16`, or the value is not `-1`, then end with error

  1.26.3. **Action**: `MSG_RESET CAP_XFERCOUNT`

    1.26.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.26.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT16`, or the value is not `-1`, then end with error

1.27. **Action**: `MSG_GETCURRENT ICAP_AUTOBRIGHT`

  1.27.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

  1.27.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

  1.27.3. **Action**: `MSG_RESET ICAP_AUTOBRIGHT`

    1.27.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.27.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.28. **Action**: `MSG_GETCURRENT ICAP_AUTODISCARDBLANKPAGES`

  1.28.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

  1.28.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBP_DISABLE`, then end with error

  1.28.3. **Action**: `MSG_RESET ICAP_AUTODISCARDBLANKPAGES`

    1.28.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

    1.28.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBP_DISABLE`, then end with error

1.29. **Action**: `MSG_GETCURRENT ICAP_AUTOMATICCOLORENABLED`

1.29.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.29.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.29.3. **Action**: `MSG_RESET ICAP_AUTOMATICCOLORENABLED`

1.29.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.29.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.30. **Action**: `MSG_GETCURRENT ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE`

1.30.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.30.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPT_BW`, then end with error

1.30.3. **Action**: `MSG_RESET ICAP_AUTOMATICCOLORNONCOLORPIXELTYPE`

1.30.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.30.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPT_BW`, then end with error

1.31. **Action**: `MSG_GETCURRENT ICAP_AUTOMATICROTATE`

1.31.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.31.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.31.3. **Action**: `MSG_RESET ICAP_AUTOMATICROTATE`

1.31.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.31.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.32. **Action**: `MSG_GETCURRENT ICAP_AUTOSIZE`

1.32.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.32.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWAS_NONE`, then end with error

1.32.3. **Action**: `MSG_RESET ICAP_AUTOSIZE`

1.32.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.32.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWAS_NONE`, then end with error

1.33. **Action**: `MSG_GETCURRENT ICAP_BARCODEDETECTIONENABLED`

1.33.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.33.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.33.3. **Action**: `MSG_RESET ICAP_BARCODEDETECTIONENABLED`

1.33.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.33.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.34. 1.35 **Action**: `MSG_GETCURRENT ICAP_BITORDER`

1.34.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.34.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBO_MSBFIRST`, then end with error

1.34.3. **Action**: `MSG_RESET ICAP_BITORDER`

1.34.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.34.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBO_MSBFIRST`, then end with error

1.35. **Action**: `MSG_GETCURRENT ICAP_BITORDERCODES`

1.35.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.35.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBO_LSBFIRST`, then end with error

1.35.3. **Action**: `MSG_RESET ICAP_BITORDERCODES`

1.35.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.35.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWBO_LSBFIRST`, then end with error

1.36. **Action**: `MSG_GETCURRENT ICAP_BRIGHTNESS`

1.36.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.36.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.36.3. **Action**: `MSG_RESET ICAP_BRIGHTNESS`

1.36.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.36.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.37. **Action**: `MSG_GETCURRENT ICAP_CCITTKFACTOR`

1.37.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.37.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `4`, then end with error

1.37.3. **Action**: `MSG_RESET ICAP_CCITTKFACTOR`

1.37.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.37.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `4`, then end with error

1.38. **Action**: `MSG_GETCURRENT ICAP_COLORMANAGEMENTENABLED`

1.38.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.38.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.38.3. **Action**: `MSG_RESET ICAP_COLORMANAGEMENTENABLED`

1.38.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.38.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.39. **Action**: `MSG_GETCURRENT ICAP_COMPRESSION`

1.39.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.39.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWCP_COMPRESSION`, then end with error

1.39.3. **Action**: `MSG_RESET ICAP_COMPRESSION`

1.39.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.39.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWCP_COMPRESSION`, then end with error

1.40. **Action**: `MSG_GETCURRENT ICAP_CONTRAST`

1.40.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.40.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.40.3. **Action**: `MSG_RESET ICAP_CONTRAST`

1.40.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.40.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.41. **Action**: `MSG_GETCURRENT ICAP_EXTIMAGEINFO`

1.41.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.41.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.41.3. **Action**: `MSG_RESET ICAP_EXTIMAGEINFO`

1.41.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.41.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `TRUE`, then end with error

1.42. **Action**: `MSG_GETCURRENT ICAP_FILTER`

1.42.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.42.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.42.3. **Action**: `MSG_RESET ICAP_FILTER`

1.42.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.42.3.2. **Test**: If the container is not `TW_ARRAY`, or the value is not an empty array, then end with error

1.43. **Action**: `MSG_GETCURRENT ICAP_FLIPROTATION`

1.43.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.43.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWFR_BOOK`, then end with error

1.43.3. **Action**: `MSG_RESET ICAP_FLIPROTATION`

1.43.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.43.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWFR_BOOK`, then end with error

1.44. **Action**: `MSG_GETCURRENT ICAP_GAMMA`

1.44.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.44.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `2.2`, then end with error

1.44.3. **Action**: `MSG_RESET ICAP_GAMMA`

1.44.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.44.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `2.2`, then end with error

1.45. **Action**: `MSG_GETCURRENT ICAP_HIGHLIGHT`

1.45.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.45.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `255`, then end with error

1.45.3. **Action**: `MSG_REEST ICAP_HIGHLIGHT`

1.45.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.45.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `255`, then end with error

1.46. **Action**: `MSG_GETCURRENT ICAP_IMAGEMERGE`

1.46.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.46.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWIM_NONE`, then end with error

1.46.3. **Action**: `MSG_RESET ICAP_IMAGEMERGE`

1.46.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.46.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWIM_NONE`, then end with error

1.47. **Action**: `MSG_GETCURRENT ICAP_IMAGEMERGEHEIGHTTHRESHOLD`

1.47.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.47.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.47.3. **Action**: `MSG_GETCURRENT ICAP_IMAGEMERGEHEIGHTTHRESHOLD`

1.47.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.47.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.48. **Action**: `MSG_GETCURRENT ICAP_MIRROR`

1.48.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.48.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWMR_NONE`, then end with error

1.48.3. **Action**: `MSG_RESET ICAP_MIRROR`

1.48.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.48.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWMR_NONE`, then end with error

1.49. **Action**: `MSG_GETCURRENT ICAP_ORIENTATION`

1.49.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.49.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWOR_PORTRAIT`, then end with error

1.49.3. **Action**: `MSG_RESET ICAP_ORIENTATION`

1.49.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.49.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWOR_PORTRAIT`, then end with error

1.50. **Action**: `MSG_GETCURRENT ICAP_OVERSCAN`

1.50.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.50.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWOV_NONE`, then end with error

1.50.3. **Action**: `MSG_RESET ICAP_OVERSCAN`

1.50.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.50.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWOV_NONE`, then end with error

1.51. **Action**: `MSG_GETCURRENT ICAP_PATCHCODEDETECTIONENABLED`

1.51.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.51.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.51.3. **Action**: `MSG_RESET ICAP_PATCHCODEDETECTIONENABLED`

1.51.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.51.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.52. **Action**: `MSG_GETCURRENT ICAP_PIXELFLAVOR`

1.52.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.52.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPF_CHOCOLATE`, then end with error

1.52.3. **Action**: `MSG_RESET ICAP_PIXELFLAVOR`

1.52.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.52.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPF_CHOCOLATE`, then end with error

1.53. **Action**: `MSG_GETCURRENT ICAP_PIXELFLAVORCODES`

1.53.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.53.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPF_CHOCOLATE`, then end with error

1.53.3. **Action**: `MSG_RESET ICAP_PIXELFLAVORCODES`

1.53.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.53.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWPF_CHOCOLATE`, then end with error

1.54. **Action**: `MSG_GETCURRENT ICAP_ROTATION`

1.54.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.54.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.54.3. **Action**: `MSG_RESET ICAP_ROTATION`

1.54.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.54.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.55. **Action**: `MSG_GETCURRENT ICAP_SHADOW`

1.55.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.55.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.55.3. **Action**: `MSG_RESET ICAP_SHADOW`

1.55.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.55.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `0`, then end with error

1.56. **Action**: `MSG_GETCURRENT ICAP_THRESHOLD`

1.56.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.56.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `128`, then end with error

1.56.3. **Action**: `MSG_RESET ICAP_THRESHOLD`

1.56.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.56.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `128`, then end with error

1.57. **Action**: `MSG_GETCURRENT ICAP_TILES`

1.57.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.57.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.57.3. **Action**: `MSG_RESET ICAP_TILES`

1.57.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.57.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.58. **Action**: `MSG_GETCURRENT ICAP_TIMEFILL`

1.58.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.58.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `1`, then end with error

1.58.3. **Action**: `MSG_RESET ICAP_TIMEFILL`

1.58.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.58.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `1`, then end with error

1.59. **Action**: `MSG_GETCURRENT ICAP_UNDEFINEDIMAGESIZE`

1.59.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.59.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.59.3. **Action**: `MSG_RESET ICAP_UNDEFINEDIMAGESIZE`

1.59.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.59.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_BOOL`, or the value is not `FALSE`, then end with error

1.60. **Action**: `MSG_GETCURRENT ICAP_UNITS`

1.60.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.60.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWUN_INCHES`, then end with error

1.60.3. **Action**: `MSG_RESET ICAP_UNITS`

1.60.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.60.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `TWUN_INCHES`, then end with error

1.61. **Action**: `MSG_GETCURRENT ICAP_XFERMECH`

1.61.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.61.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `-1`, then end with error

1.61.3. **Action**: `MSG_RESET ICAP_XFERMECH`

1.61.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.61.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_UINT16`, or the value is not `-1`, then end with error

1.62. **Action**: `MSG_GETCURRENT ICAP_XSCALING`

1.62.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.62.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `1`, then end with error

1.62.3. **Action**: `MSG_RESET ICAP_XSCALING`

1.62.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.62.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `1`, then end with error

1.63. **Action**: `MSG_GETCURRENT ICAP_YSCALING`

1.63.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.63.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `1`, then end with error

1.63.3. **Action**: `MSG_RESET ICAP_YSCALING`

1.63.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.63.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_FIX32`, or the value is not `1`, then end with error

1.64. **Action**: `MSG_GETCURRENT ICAP_ZOOMFACTOR`

1.64.1. **Test**: If result is not `TWRC_SUCCESS`, then skip down to the next capability

1.64.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT16`, or the value is not `0`, then end with error

1.64.3. **Action**: `MSG_RESET ICAP_ZOOMFACTOR`

1.64.3.1. **Test**: If result is not `TWRC_SUCCESS`, then end with error

1.64.3.2. **Test**: If the container is not `TW_ONEVALUE`, or the data type is not `TWTY_INT16`, or the value is not `0`, then end with error

A

TWAIN Articles

Chapter Contents

The articles in this appendix provide additional information about some of the features described in this specification.

Device Events

TWAIN 1.8 expands upon asynchronous event notification. Previous versions provided the `DG_CONTROL / DAT_NULL` messages: `MSG_CLOSEDSOK`, `MSG_CLOSEDSREQ` and `MSG_XFERREADY` to permit the Source to alert the Application that it needed to exit, or that an image was ready to be processed. With the addition of Digital Cameras, and the burgeoning interest in Push Technologies, it has become desirable to enhance TWAIN in this area.

An event begins when the Source needs to alert the Application to some change that has occurred within the device. For example, the owner of a Digital Camera (which is tethered to a host machine) has changed the setting for flash from on to off. The Source wants to alert the Application of this change: first, it records the event in a FIFO queue; second, it sends a `DG_CONTROL / DAT_NULL / DAT_DEVICEEVENT` to the Source Manager, which forwards the message to the Application.

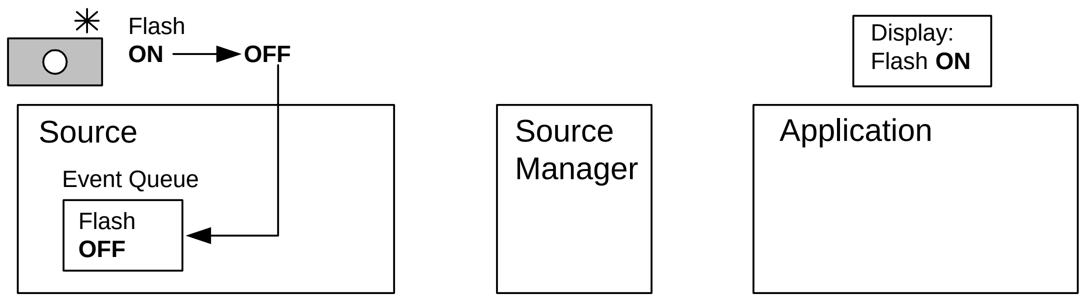
The Application receives the DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT, and immediately issues a DG_CONTROL / DAT_DEVICEEVENT / MSG_GET request to the Source. The Source delivers the information about the event, and pops it off the queue. The process concludes with the Application examining the information and acting upon it, in this case by alerting the user that the flash setting on the camera has been changed.

Notes:

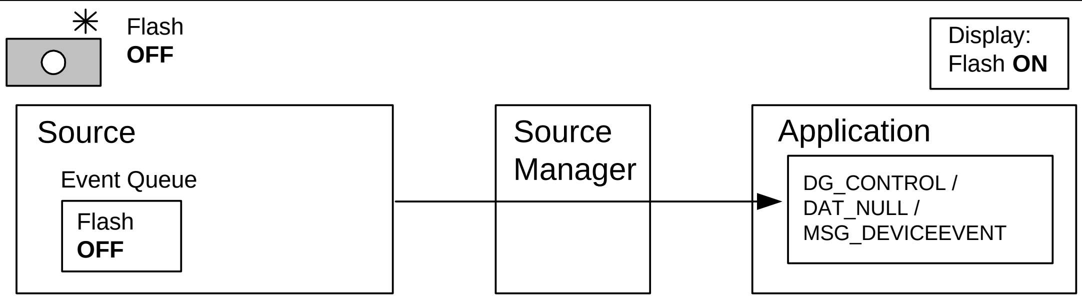
- Sources must start up in a mode with device events turned off (an empty array for CAP_DEVICEEVENTS), this is for the benefit of pre-1.8 applications which may not be able to process this new event.
- Device events are never generated by an Application setting a value within a Source (such as Application changing ICAP_FLASHUSED2). Device events are only generated in response to some outside change within the Source or the Device (such as the User changing the flash setting on the camera).
- Sources must maintain an internal Event Queue, so that they can report each and every device event to the Application in the order of their occurrence.
- Device events are supported in State 4. Windows Sources must use the main window handle supplied with the DG_CONTROL / DAT_PARENT / MSG_OPENDS if they issue device events in State 4. In States 5 through 7 Sources must use the pTW_USERINTERFACE->hParent supplied in the DG_CONTROL / DAT_USERINTERFACE / MSG_ENABLEDS triplet.
- Since device events may occur in State 4, Applications that enable them using CAP_DEVICEEVENTS must be ready to receive and process them.
- When the Application receives a device event, it must immediately collect the information about it. The Application must not issue the DG_CONTROL / DAT_DEVICEEVENT / MSG_GET, except when it has received a DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT message.
- The Application must process events without User intervention, this is to prevent situations where the device event queue builds up because a User is not responding to the system.
- Applications may sometimes fail to respond to a Source's device events. A maximum queue size should be selected so that the Source does not exhaust memory. If the queue fills, the Source must do the following:
  - Turns off device events (resets CAP_DEVICEEVENT to an empty array).
  - Refuse to set CAP_DEVICEEVENT until the queue is emptied, return TWCC_SEQERROR.
  - Process DG_CONTROL / DAT_DEVICEEVENT / MSG_GET requests for each item on the device event queue.

After the last device event is read by the Application, return TWRC_FAILURE / TWCC_DEVICEEVENTOVERFLOW for the next call to DG_CONTROL / DAT_DEVICEEVENT / MSG_GET.

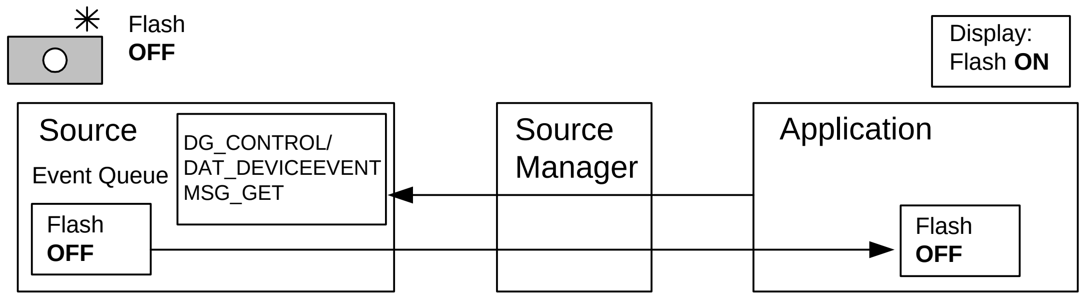
  - After TWCC_DEVICEEVENTOVERFLOW has been reported, permit the Source to set CAP_DEVICEEVENT again.



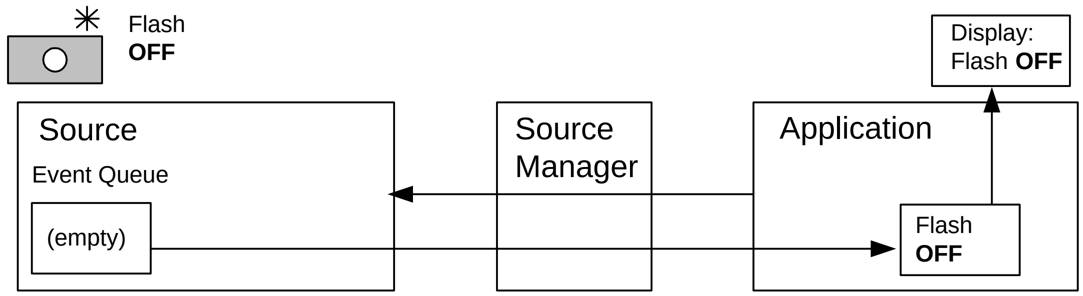
Step 1: The Source senses that the device has changed from ON to OFF and stores this information in an Event Queue. A Queue must be used because the Source may generate multiple events before the Application can respond.



Step 2: The Source sends a DG_CONTROL / DAT_NULL / MSG_DEVICEEVENT to the Application. The Application only knows that some Event has taken place.



Step 3: The Application sends a DG_CONTROL / DAT_DEVICEEVENT / MSG_GET to the Source to learn about the Event. The Source informs the Application that the flash is OFF and it clears the Event from its Queue.



Step 4: The Application informs the User that the flash is now OFF.

Figure A-1 Device Events

This section details the various event types and how Sources and Applications should make use of them.

TWDE_CHECKAUTOMATICCAPTURE

The automatic capture settings on the device have been changed.

TWDE_CHECKBATTERY

Status of the battery has changed. Sources will report BatteryMinutes or BatteryPercentage depending on which capabilities say they support.

TWDE_CHECKDEVICEONLINE

The device has been powered off. If an Application receives this device event, it should call `CAP_DEVICEONLINE` to verify the state of the Source, and then proceed as seems appropriate.

TWDE_CHECKFLASH

The flash setting on the device has been changed.

TWDE_CHECKPOWERSUPPLY

The power supply has changed, for example this event would be generated if AC was removed from a device, putting it on battery. Scanners may also provide this event to notify that a power on reset has taken place, indicating that the device has been power cycled.

TWDE_CHECKRESOLUTION

The resolution on the device has changed.

TWDE_DEVICEADDED

A device has been added to the Source. See `DG_CONTROL` / `DAT_FILESYSTEM` / `MSG_CHANGEDIRECTORY` and `DG_CONTROL` / `DAT_FILESYSTEM` / `MSG_GETINFO` to get more information about the new device.

TWDE_DEVICEOFFLINE

A device has become unavailable. This is different from `TWDC_DEVICEREMOVED`, since the device is assumed to be connected.

TWDE_DEVICEREADY

A device is ready to capture another image. Applications should be careful when negotiating this event, especially in situations where images are gathered quickly, as with automatic capture.

TWDE_DEVICEREMOVED

A device has been removed from the Source. This is different from `TWDE_DEVICEOFFLINE`. As soon as this event is received an Application should re-negotiate its current device, since that may have been the one that was removed. Sources must default to the `TWFY_CAMERA` device if the current device is removed.

TWDE_PAPERDOUBLEFEED

Report double feeds to the Application. Because of the asynchronous nature of device events there may still be images waiting to be transferred, applications need to decide if they want to recover these images or discard them.

TWDE_PAPERJAM

Report paper jams to the Application. Because of the asynchronous nature of device events there may still be images waiting to be transferred, applications need to decide if they want to recover these images or discard them.

Supported Sizes

Typical uses for `ICAP_SUPPORTEDSIZES` include, but are not limited to the following:

A0, A1	Technical drawings, posters
A2, A3	Drawings, diagrams, large tables
A4	Letters, magazines, forms, catalogs, laser printer and copying machine output
A5	Note pads
A6	Postcards
B5, A5, B6, A6	Books
C4, C5, C6	Envelopes for A4 letters: unfolded (C4), folded once (C5), folded twice (C6)
B4, A3	Newspapers, supported by most copying machines in addition to A4

The following table details the physical dimensions associated with `ICAP_SUPPORTEDSIZES`. Multiply millimeters by 0.03937 to get the approximate inches. Multiply inches by 25.4 to get the approximate millimeters.

ICAP_SUPPORTEDSIZES	Description
`TWSS_NONE`	Images will match the maximum scanning dimensions of the device. This setting is only applicable to devices that have fixed measurable dimensions, such as most scanners. Devices that do not support physical dimensions should not support `ICAP_SUPPORTEDSIZES`.

`TWSS_A4LETTER`
`TWSS_B5LETTER`
`TWSS_B3`
`TWSS_B4`
`TWSS_B6`
	These values are preserved for backward compatibility. TWAIN 1.8+ enabled Applications should not use these settings.

TWSS_B	This value is obsolete, and no longer supported by the specification. Do not use it.	
TWSS_USLETTER	8.5″ x 11.0″	(216mm x 280mm)
TWSS_USLEGAL	8.5″ x 14.0″	(216mm x 356mm)
TWSS_USLEDGER	11.0″ x 17.0″	(280mm x 432mm)
TWSS_USEXECUTIVE	7.25″ x 10.5″	(184mm x 267mm)
TWSS_USSTATEMENT	5.5″ x 8.5″	(140mm x 216mm)

TWSS_BUSINESSCARD	90mm x 55mm
TWSS_4A0	1682mm x 2378mm
TWSS_2A0	1189mm x 1682mm
TWSS_A0	841mm x 1189mm
TWSS_A1	594mm x 841mm
TWSS_A2	420mm x 594mm
TWSS_A3	297mm x 420mm
TWSS_A4	210mm x 297mm
TWSS_A5	148mm x 210mm
TWSS_A6	105mm x 148mm
TWSS_A7	74mm x 105mm
TWSS_A8	52mm x 74mm
TWSS_A9	37mm x 52mm
TWSS_A10	26mm x 37mm
TWSS_ISOB0	1000mm x1414mm
TWSS_ISOB1	707mm x1000mm
TWSS_ISOB2	500mm x 707mm
TWSS_ISOB3	353mm x 500mm
TWSS_ISOB4	250mm x 353mm
TWSS_ISOB5	176mm x 250mm
TWSS_ISOB6	125mm x 176mm
TWSS_ISOB7	88mm x 125mm
TWSS_ISOB8	62mm x 88mm

TWSS_ISOB9	44mm x 62mm
TWSS_ISOB10	31mm x 44mm
TWSS_JISB0	1030mm x1456mm
TWSS_JISB1	728mm x1030mm
TWSS_JISB2	515mm x 728mm
TWSS_JISB3	364mm x 515mm
TWSS_JISB4	257mm x 364mm
TWSS_JISB5	182mm x 257mm
TWSS_JISB6	128mm x 182mm
TWSS_JISB7	91mm x 128mm
TWSS_JISB8	64mm x 91mm
TWSS_JISB9	45mm x 64mm
TWSS_JISB10	32mm x 45mm

TWSS_C0	917mm x1297mm
TWSS_C1	648mm x 917mm
TWSS_C2	458mm x 648mm
TWSS_C3	324mm x 458mm
TWSS_C4	229mm x 324mm
TWSS_C5	162mm x 229mm
TWSS_C6	114mm x 162mm
TWSS_C7	81mm x 114mm
TWSS_C8	57mm x 81mm
TWSS_C9	40mm x 57mm
TWSS_C10	28mm x 40mm

Automatic Capture

Automatic image capture is intended for Digital Cameras, although there may be opportunities for other kinds of devices. The intention is to allow an Application to control when pictures are taken, how many pictures are taken, and the interval of time between picture taking. All that is required is that the device be able to perform capture on command from the Source, the timing control and storage of pictures may reside in the Source or the device; the Application does not care.

There are three capabilities needed to control automatic capture:

- CAP_AUTOMATICCAPTURE
- CAP_TIMEBEFOREFIRSTCAPTURE
- CAP_TIMEBETWEENCAPTURES

And one triplet:

- DG_CONTROL/DAT_FILESYSTEM/MSG_AUTOMATICCAPTUREDIRECTORY

CAP_AUTOMATICCAPTURE selects the number of images to be captured. A value of zero (0), the default, disables it. CAP_TIMEBEFOREFIRSTCAPTURE selects how many milliseconds are to pass before the first picture is taken by the device. If this value is 0, then picture taking begins immediately. CAP_TIMEBETWEENCAPTURES selects the milliseconds of elapsed time between pictures. If this value is 0, then the pictures are taken as fast as the device can go.

DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIRECTORY selects the directory that will receive the images as they are captured.

Automatic capture expects the device (or Source) to manage the storage of images until the Application is ready to collect them. Applications may choose to retrieve images as they are captured by the Source (using the DAT_FILESYSTEM triplets to browse the storage directory), but must realize that this may affect the performance of the device.

The nature of automatic capture suggests that an Application should be able to disconnect from a Source and expect that if it returns after CAP_TIMEBEFOREFIRSTCAPTURE has passed, there may be images available for it to collect. Because of this Sources should remember their automatic capture settings from session to session, so that a Source starting up does not inadvertently clear them.

Applications need to remember that since the capture of images may occur outside of their control that the settings may be changed directly on the device by the user, resulting in alternations in any of the automatic capture settings. Applications that cannot support this uncertainty should clear the Source's automatic capture settings prior to shutdown (and after notifying the User).

Camera Preview

Some digital cameras offer a way to preview the intended shot through either a continuous flow of low-resolution frames or streaming video. TWAIN exposes two methods for a Source to present this information to an Application, both in association with the TWFY_CAMERAPREVIEW device.

The TWFY_CAMERAPREVIEW Device

Sources that wish to provide access to their preview camera must do so through DAT_FILESYSTEM. A minimum configuration includes a single TWFY_CAMERA and a single TWFY_CAMERAPREVIEW. The Application discovers what devices are available by using the

DAT_FILESYSTEM commands MSG_GETFIRSTFILE and MSG_GETNEXTFILE. It can then switch from the startup default TWFY_CAMERA to the TWFY_CAMERAPREVIEW using the MSG_CHANGEDIRECTORY command.

Performance

It is important when taking a picture from preview mode that the switch from TWFY_CAMERAPREVIEW to TWFY_CAMERA happens as quickly as possible. Applications can minimize the switch over time by negotiating the settings of the TWFY_CAMERA before changing to the TWFY_CAMERAPREVIEW device to collect real-time images.

Sources can help by optimizing their communication with the TWFY_CAMERA, perhaps downloading its values when the user sends MSG_ENABLEDS to the TWFY_CAMERAPREVIEW device so that when the switch back occurs all that needs to happen is a command sent to the camera to take a picture.

Another matter of importance is the transfer mechanism. If the camera is capable of sending a run of continuous snapshots to the application (as opposed to real video streaming), then it is recommended that the TWFY_CAMERAPREVIEW device only support an ICAP_XFERMECH of TWSX_NATIVE.

Entering Preview Mode

An application should do the following before entering preview mode.

1. The application sends MSG_OPENDS to the Source.
2. The application determines that the Source TWFY_CAMERAPREVIEW device.
3. The user/application negotiates values for the TWFY_CAMERA device.
4. The user/application decides to enter preview mode. The application uses MSG_CHANGEDIRECTORY to change to the TWFY_CAMERAPREVIEW device.
5. The application uses MSG_ENABLEDS to enter preview mode. Note that the value of ShowUI should depend on which of the next two sections the application decides to use to control the Source (GUI mode or programmatic).

Previewing with the Source's GUI (ShowUI == TRUE)

If the application relies solely on the Source's GUI for its control of the camera, then it shouldn't have to worry about preview mode issues, since it is hoped that a Source that supports preview will provide access to it from its GUI. This section is concerned with a more limited area, where an application has opted to control the Source programmatically, except for the use of preview. One reason an application might need to do this is to provide preview support for cameras that output streaming video. TWAIN does not have a mechanism for handling this kind of data, so if the only way that a TWAIN application will be able to show this kind of preview data, is if the Source provides a GUI that can show it.

If the Source has CAP_CAMERAPREVIEWUI set to TRUE, then it is possible for the application to use this to preview the images coming from the camera. In this mode the application does not have to concern itself with the kind of data that the Source is providing, since the Source takes the responsibility of displaying the preview images to the user. However, the application does have to wait for the triggers that indicates that the user wishes to take a picture, or that they wish to exit from preview mode. To help standardize this behavior, the preview GUI should be able to indicate two things.

1. **Take a picture** – if the user selects to take a picture, perhaps by pressing a button labeled CAPTURE, then the Source should send the DAT_NULL command MSG_CLOSEDSOK back to the application.
2. **Cancel preview** – if the user decides to exit from preview mode, then the Source should send the DAT_NULL command MSG_CLOSEDSREQ back to the application. The application should then send MSG_DISABLEDS to the Source, change back to the TWFY_CAMERA device, and resume its programmatic control of the Source.

Previewing under Programmatic Control (ShowUI == FALSE):

TWAIN provides programmatic support for TWFY_CAMERAPREVIEW devices that operate by taking a continuous flow of low-resolution snapshots. An application learns that a Source is capable of this by changing to TWFY_CAMERAPREVIEW and testing ICAP_XFERMECH. If the capability is supported, then the TWFY_CAMERAPREVIEW device is capable of transferring these low-resolution images fast enough to simulate real-time video. The way the application obtains these images is similar to how scanners work. The application sets CAP_XFERCOUNT to –1 and enables the Source. The Source sends a MSG_XFERREADY to the application, and the application begins transferring and displaying the low-resolution images as fast as it can. These steps are repeated to aid understanding…

1. The application negotiates any capabilities with the TWFY_CAMERAPREVIEW device, including setting CAP_XFERCOUNT to –1, indicating that the application wishes to receive an unlimited number of images.
2. The application send MSG_ENABLEDS (ShowUI == FALSE) to the Source.
3. The Source sends back MSG_XFERREADY and transitions to State 6.
4. The application uses MSG_IMAGENATIVEXFER to transfer the image and the Source transitions to State 7.
5. The application displays the image.
6. The application uses DAT_PENDINGXFERS / MSG_ENDXFER to transition the Source to State 6. The application needs to pay attention to the TW_PENDINGXFERS.Count, but it is expected that it should remain at –1.
7. Go to step (4).

As long as the application and Source are looping from steps (4) through (7) the application should be displaying a continuous run of snapshots.

Since the application is in complete control, it is implementation dependent on how the user indicates that a picture should be taken. However, once the decision to take a picture is made, the steps to do it are as follows…

Taking a Picture:

The application should do the following when it is told to take a picture while in preview mode.

1. The application sends DAT_PENDINGXFERS / MSG_ENDXFER to the Source, transitioning from State 7 to State 6 (if necessary).
2. The application sends DAT_PENDINGXFERS / MSG_RESET to the Source, transitioning from State 6 to State 5.
3. The application sends MSG_DISABLEDS to the Source, transitioning from State 5 to State 4.

4. The application uses `MSG_CHANGEDIRECTORY` to switch from the `TWFY_CAMERAPREVIEW` device to the `TWFY_CAMERA` device.
5. The application uses `MSG_ENABLEDS (ShowUI == FALSE)` to enable the `TWFY|_CAMERA` device.
6. The application sends one of the `MSG_IMAGExxxxXFER` commands to the Source.
7. The source takes the full resolution picture and transfers it back to the application

File System

This section consists of the following:

- Overview
- Rules for path and file names
- File system components
- Rule for root directory
- Rules for image directory
- File Types
- `DAT_FILESYSTEM` operations
- Thumbnails and Sound snippets
- Context variable
- Condition Codes

Note: The term ‘camera’ is used generically in the specification to describe a device that captures an image, and is not limited to just devices that employ a camera to accomplish this.

Overview

Digital cameras and some scanners have the ability to capture images to their own local storage. When Automatic Capturing is being used an Application need not collect the captured images until long after their acquisition. A file system is a good representation for the storage of images (since it is a model that is familiar to most programmers), so TWAIN exposes a simple file system interface that Applications may browse through in a random fashion.

There is also a need in TWAIN to expose multiple devices through a single Source. Single pass duplex scanners have multiple cameras that accept different settings. Digital cameras come with disks and memory expansion cards, and many are able to provide a stream of preview images. The file system offers a way for a Source to maintain in its root directory a list of the devices available to an Application.

Rules for Path and File Names

There are two main grouping of files supported by TWAIN; devices, which are associated with real-time capture, which accept image capture settings, and which are of the form:

`/DeviceName`

And image path and file names, which are images on local storage which have been previously captured by the device, and which are of the form (bracketed items are optional):

`[/DomainName] [/HostName] /TopDirectory [/Sub-Directory…] /ImageFile`

1. A filename consists of any characters *except*: NUL (0), either of the slashes '/' or '\' and the colon ':'.
2. Sources should at a minimum support the characters: "A-Z a-z 0-9 _ ."
3. The file system should not be case sensitive, though it may show upper and lowercase.
4. Applications should take into consideration that internationalized Sources may construct filenames from characters within UNICODE.
5. The forward slash '/' and backward slash '\' may be used interchangeably in the creation of path names. Sources and Applications must support the use of both slashes. (ex: /abc\xyz).
6. Multiple adjacent slashes reduce to a single slash. (ex: ///\\abc///xyz == /abc/xyz).
7. The root directory is designated as a solitary slash (ex: / or \).
8. The `MSG_CHANGEDIRECTORY` and `MSG_AUTOMATICCAPTUREDIRECTORY` operations are the only ones that accepts absolute or relative directory paths. All other operations occur within the current directory.
9. `MSG_CHANGEDIRECTORY` and `MSG_AUTOMATICCAPTUREDIRECTORY` can use dot '.' to address the current directory (ex: ./abc).
10. `MSG_CHANGEDIRECTORY` and `MSG_AUTOMATICCAPTUREDIRECTORY` can use dot-dot '..' to address the parent directory (ex: ../abc).
11. In the root directory a `MSG_CHANGEDIRECTORY` or `AUTOMATICCAPTUREDIRECTORY` to dot-dot '..' is the same as dot '.' (ex: /. == /..).

Examples:

`\Camera` is the same as `/Camera`

`//Camera` is the same as `/Camera`

`./Camera` is the same as `/Camera`

`../Camera` is the same as `/Camera`

File System Components

A file system consists of the following.

1. A root directory.
2. A camera device (`TWFY_CAMERA`), which must be the default device when the Source starts.
3. Zero or more additional devices (`TWFY_CAMERATOP`, `TWFY_CAMERATOP`, `TWFY_CAMERAPREVIEW`).
4. It is possible for a Source to support multiples of a given device type, for instance a scanner may support two devices of type `TWFY_CAMERA`, both with a supporting `TWFY_CAMERATOP` and `TWFY_CAMERABOTTOM`. Use `pTW_FILESYSTEM->DeviceGroupMask` to uniquely identify a camera or to group it with its associated top and bottom cameras. For example:

Name	Type	Group
/camera_1	TWFY_CAMERA	0x0001
/camera_1_top	TWFY_CAMERATOP	0x0001
/camera_1_bottom	TWFY_CAMERABOTTOM	0x0001
/camera_2	TWFY_CAMERA	0x0002
/camera_2_top	TWFY_CAMERATOP	0x0002
/camera_2_bottom	TWFY_CAMERABOTTOM	0x0002

5. Zero or more directories for storing images (on memory cards, disks, etc…). These are organized in a hierarchical structure that permits, but does not require the ability to browse in a network:

   A `TWFY_DOMAIN` directory contains only `TWFY_HOST` directories

   A `TWFY_HOST` directory contains only `TWFY_DIRECTORY` directories

   A `TWFY_DIRECTORY` contains `TWFY_IMAGE` files and/or `TWFY_DIRECTORY` directories.

   Sources that provide image storage must provide at least one `TWFY_DIRECTORY`. `TWFY_DOMAIN` and `TWFY_HOST` are optional.

Rules for Root Directory

1. The root directory can only contain devices or directories, not images.
2. The application cannot create, delete, copy into or rename files in the root directory.
3. Files in a directory are not ordered in any fashion (for instance, an Application may not assume that they are alphabetically sorted). There is one exception to this rule: when an Application issues a `DG_CONTROL` / `DAT_FILESYSTEM` / `MSG_GETFIRSTFILE` on the root directory, the Source must return a `TWFY_CAMERA` device. This device is the designated default capture camera. If an Application begins capability negotiation, or image capture *without* accessing `DAT_FILESYSTEM`, then this is the device that will be used.

Rules for Image Directory

1. A `TWFY_DIRECTORY` can contain 0 or more `TWFY_DIRECTORYs` (sub-directories).
2. Can contain 0 or more `TWFY_IMAGE` (image files).
3. May be fully accessible, read or write protected.
4. May be created or deleted by an Application, given that it is not in the root directory, and that it is not protected by the Source.

Context Variable

The reason for the Context variable is that it allows for unconditional mingling of `DAT_FILESYSTEM` operations. If there was no Context variable, then Applications would be more limited in the order of operations that could be performed. For instance, the recursive directory walk in the code sample would be much harder to accomplish without a Context to help the Source identify the current directory being accessed by a call to `MSG_GETNEXTFILE`.

This value is provided solely for the benefit of Source writers. When MSG_GETFIRSTFILE is called, the Source should record the current directory and the current file and store those values internally, using Context as a reference to their location. The nature or value of the Context is dependent on the implementation of the Source, Applications must never attempt to use or modify the Context. A call to MSG_GETINFO must use this Context to identify the file being reported. Calls to any of the file transfer methods (MSG_IMAGENATIVEXFER, MSG_IMAGEFILEXFER, MSG_IMAGEMEMXFER, MSG_AUDIONATIVEXFER, MSG_AUDIOFILEXFER) must use this Context to determine the data being sent to the Application. A call to MSG_GETNEXTFILE must use this Context to help obtain the next file from the directory (this will result in a change in the context as it references the new file). And, finally, a call to MSG_GETCLOSE releases the memory in the Source associated with this Context.

Condition Codes

These are some condition codes that apply specifically to file system operations:

TWCC_DENIED	File system operation is denied. A Source should report this condition code if an attempt is made to access a protected file. Examples of such protection include: any attempt to delete, rename or copy into the root directory; protected files that are on the network; and any file that the Source feels it needs to protect.
TWCC_FILEEXISTS	The operation failed because the file already exists. A Source should report this condition code if an attempt is made to create a sub-directory with a name that already exists in the targeted directory; or if an attempt is made to copy or rename over an existing file or directory.
TWCC_FILENOTFOUND	The file was not found. This can occur for a variety of reasons: attempts to change directory to a path that does not exist; attempts to delete, rename or copy files that do not exist; as the condition code from MSG_GETFIRSTFILE for an empty directory; or MSG_GETNEXTFILE when it finds no more files in the current directory; and, finally, from MSG_GETINFO if it is requested to provide information on a file that has been deleted.
TWCC_NOTEMPTY	Operation failed because the directory is not empty. This condition code is used by the Source if an attempt is made with the Recursion flag set to FALSE to delete a non-empty directory.

File Types

The DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY operation is used to make either a device or a directory current. If a camera device is the target, then all capability negotiation is with that device and all images come from that device, until a new MSG_CHANGEDIRECTORY command is issued. If an image directory is selected then the current device is set to be the root level directory name (i.e., changing to /abc/mno/xyz means that the current device is /abc).

TWFY_CAMERA	Every TWAIN file system must support at least one camera, which must be the default device on startup. This is for

compatibility with pre 1.8 applications as well as post 1.8 applications that do not choose to make use of the file system. On single pass duplex scanners, this camera device is used to simultaneously set values for the top and bottom cameras. During the capturing of images (in duplex mode) it sends a stream of images in the order: TOP, BOTTOM, TOP…

`TWFY_CAMERATOP` / `TWFY_CAMERABOTTOM`

Single pass duplex scanners may opt to provide independent access to the top and bottom cameras. A device with one of these file types controls the settings for the specified camera. If this device is the current device at the time image capture commences, then only images from that camera will be passed to the Application. This means that even if a device is set for duplex scanning, if the current device has a file type of `TWFY_CAMERATOP`, then only top images will be passed to the Application.

`TWFY_CAMERAPREVIEW`

A logical device that performs camera live preview functionality. When implementing the Source for this logical device, related capabilities must be negotiated to perform preview specific functions. Among them, `ICAP_XRESOLUTION` and `ICAP_YRESOLUTION` must be implemented to specify the preview image sizes. Other capabilities may be available in some sources, such as `ICAP_ZOOMFACTOR` and `ICAP_FLASHUSED2`.

`TWFY_DIRECTORY`	At the root directory level files of this type should correspond to a physical piece of hardware (a memory card or a disk). The root directory is only allowed to contain devices. Sub-directories may only contain image files or more sub-directories. Access to files and directories is controlled by the Source, so Applications should check all operations and watch out for condition codes such as `TWCC_DENIED`.
`TWFY_IMAGE`	Any directory, except root, may contain image files. The `DAT_FILESYSTEM` messages `MSG_GETFIRSTFILE` and `MSG_GETNEXTFILE` select the current image. Once an image has been selected, it may be transferred in the same fashion used to acquire images from a camera. Note: this file type is reserved for full resolution images, see the section on Thumbnails for information on how to acquire them.

DAT_FILESYSTEM Operations

`MSG_AUTOMATICCAPTUREDIRECTORY`	Selects the directory to be used to store images acquired by automatic capture.
`MSG_CHANGEDIRECTORY`	Selects the device or image subdirectory. Use this to select between direct camera (scanner) control, and browsing of stored images. All capabilities negotiated and triplet operations are with the current device (directory), until this value is changed by the Application.
`MSG_COPY`	Copies the specified file from one directory to another. If the Recursive flag is `TRUE` and the file type specified is `TWFY_DIRECTORY` then that directory and all the files and directories under it are copied. The Application cannot copy files into the root directory.
`MSG_CREATEDIRECTORY`	Creates a new image subdirectory. The Application cannot create files in the root directory.
`MSG_DELETE`	Deletes the specified file. If the Recursive flag is `TRUE` and the file type specified is `TWFY_DIRECTORY`, then all the files under that directory are deleted. The Application cannot delete files in the root directory.
`MSG_FORMATMEDIA`	Formats the currently selected storage device. Use with caution.

`MSG_GETCLOSE`	Closes the Context created by `MSG_GETFIRSTFILE`.
`MSG_GETFIRSTFILE`	Creates a Context that points to the first file in a directory. This Context is used by `MSG_GETINFO`, `MSG_GETNEXTFILE`, `MSG_GETCLOSE`; and for files of type `TWFY_IMAGE` all image transfer related operations performed in states 6 and 7 use the image pointed to by this Context (i.e., `DAT_IMAGEINFO`, `DAT_IMAGEMEMXFER`, etc…).
`MSG_GETINFO`	Returns information about a device, directory or image file.
`MSG_GETNEXTFILE`	Updates the Context to point to the next file in the directory.
`MSG_RENAME`	Renames a directory or an image file. If the directories differ, then it moves the file as well, creating it in the new location and deleting it from the old location. Files in the root directory cannot be renamed by the Application.

Thumbnails and Sound Snippets

TWAIN is primary concerned with the acquisition of images, so the file system does not contain thumbnail files or sound files, since these kinds of data are expected to be associated with image files. This simplifies an Application's browsing of the file system, since it need only concern itself with one type of data file (`TWFY_IMAGE`), and does not have to trace associated data files.

Sources must filter out non-image files, if the device stores thumbnail and sound data independent of the image files. For instance, if a device stores the following files:

```
IMAGE001.TIF
IMAGE001_THUMBNAIL.TIF
IMAGE001_SOUND.WAV
```

The file system must only report the existence of `IMAGE001.TIF`

An Application obtains the thumbnail for an image by setting `ICAP_THUMBNAILSENABLED` to `TRUE`; the same filename is used for both the full resolution and thumbnail versions of an image. By setting `ICAP_THUMBNAILSENABLED`, the Application decides which version of the image it receives.

Sound snippets are also associated with image files, unlike thumbnails it is possible for a single image file to own several sound snippets. An Application can get the number of snippets that an image owns, and then, during image transfer, the Application has the option to transfer any number of those snippets. It is also possible to collect the snippets for an image without transferring the image data.

Sample Recursive Directory Walk

The following is a sample recursive directory walk.

```
// This Application function walks through all the files in a Source's
// file system, counting the file types file system, counting the file
// types it finds.  It is intended only as a sample, error checking is
// omitted to simplify the code.
typedef struct {
   int Devices;
   int Directories;
   int Images;
} t_Counters;
TW_UINT16 DirectoryWalk(TW_FILESYSTEM *fsArg, t_Counters *Counters)
{
   TW_UINT16 rc;  TW_FILESYSTEM fs;
   // Caller has set fsArg->InputFile to some value, such as "/"…
   rc = (*DS_Entry) (&app,&src,DG_CONTROL,DAT_FILESYSTEM,
      MSG_CHANGEDIRECTORY, fsArg);

   // We do GETFIRSTFILE first in each new directory, GETNEXTFILE for all
   // subsequent calls…
   for (rc = (*DS_Entry)(&app,&src,DG_CONTROL,DAT_FILESYSTEM,
      MSG_GETFIRSTFILE,&fs);
       rc == TWRC_SUCCESS;
      rc = (*DS_Entry)(&app,&src,DG_CONTROL,DAT_FILESYSTEM
         ,MSG_GETNEXTFILE,&fs)) {

         // Count the appropriate file type…
         switch (fs.FileType) {
             default:  Counters->Devices += 1;  break;
             case TWFY_IMAGE:  Counters->Images  += 1;  break;
             case TWFY_DOMAIN:
             case TWFY_HOST:
             case TWFY_DIRECTORY:
                  Counters->Directories += 1;
                 // Recursively step into this directory, looking for more
                 // stuff…
                  rc = DirectoryWalk(&fs,&Counters);
                  if (rc != TWRC_SUCCESS) {
                    rc = (*DS_Entry)(&app,&src,DG_CONTROL,DAT_FILESYSTEM,
                       MSG_GETCLOSE,&fs);
                     return(rc);
                  }
                  break;
         }
   }
```

```
      // Cleanup and return…
      rc = (*DS_Entry)(&app,&src,DG_CONTROL,DAT_FILESYSTEM,MSG_GETCLOSE,&fs);
      return(TWRC_SUCCESS);
   }

   // Using this function…
   TW_UINT16 rc;
   TW_FILESYTEM fs;
   t_Counters Counters;
   memset(&fs,0,sizeof(fs));
   memset(&Counters,0,sizeof(Counters));
   strcpy(fs.InputFile,"/"); // start at root…
   rc = DirectoryWalk(&fs,&Counters);
```

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…
```

Audio Snippets

Digital Cameras have the ability to acquire audio snippets along with an image. To support this TWAIN 1.8 provides a new data group, `DG_AUDIO`. Because TWAIN is image-centric, `DG_AUDIO` operations are dependent on an image context, audio snippets must be associated with an image. When a Source enters into state 6, the Application can opt to transfer any and all audio snippets. The steps required to obtain audio snippets deliberately parallel the steps required to transfer images, to reduce the effort to learn how to access this new kind of data.

The following Data Argument Types (DATs) are supported by `DG_AUDIO`:

DAT_AUDIOFILEXFER	transfer audio in file format
DAT_AUDIOINFO	info about an audio snippet
DAT_AUDIONATIVEXFER	transfer audio in native format

The following DG_CONTROL (DATs) are supported when DAT_XFERGROUP is set to DG_AUDIO, DATs not mentioned in this list must return TWRC_FAILURE / TWCC_BADPROTOCOL:

DAT_CAPABILITY	no changes to its operation
DAT_EVENT	no changes to its operation
DAT_IDENTITY	no changes to its operation
DAT_NULL	no changes to its operation
DAT_PASSTHRU	no changes to its operation
DAT_PENDINGXFERS	reports number of snippets remaining to be transferred, MSG_ENDXFER and MSG_RESET do not cause the Source to drop to State 5.
DAT_SETUPFILEXFER	selects the audio file format
DAT_STATUS	no changes to its operation
DAT_USERINTERFACE	no changes to its operation
DAT_XFERGROUP	MSG_SET, MSG_GETDEFAULT and MSG_GETCURRENT added to allow switching between data groups. The default value for MSG_GETDEFAULT must be DG_IMAGE. And when the Source starts up, MSG_GETCURRENT must report DG_IMAGE as the current data group, to maintain compatibility with pre-TWAIN 1.8 Applications.

The following capabilities support audio; all capabilities are negotiable at all times (at least in state 4), independent of the current setting of DAT_XFERGROUP:

ACAP_XFERMECH	negotiate audio snippet transfer mechanism

Collecting Audio Snippets

The transfer of an audio snippet was designed to be used when an Application is browsing through a selection of stored images. There is nothing to prevent the transfer of audio when an image is captured in real-time, though TWAIN does require that any audio snippets be transferred before the image is transferred.

A typical transfer may occur in the following way: An Application is browsing through storage managed by the TWAIN Source using MSG_GETFILEFIRST / MSG_GETFILENEXT (see DAT_FILESYSTEM), and finds an image that it wants to work with. The Application enters state 6 by calling DG_CONTROL / DAT_IDENTITY / MSG_ENABLEDS. If the Application wants to find out if there are any audio snippets associated with the image, it can call DG_AUDIO / DAT_AUDIOINFO / MSG_GET. In this example it finds in the TW_AUDIOINFO structure that this image file has three audio snippets associated with it. The Application wants the second audio snippet, so it calls DG_CONTROL / DAT_XFERGROUP / MSG_SET and sets the data group to DG_AUDIO. This call changes the context of the Source, it is now set up to transfer audio data. One effect of this is that a call to DG__CONTROL / DAT_PENDINGXFERS / MSG_GET will report the number of audio snippets (for this image) that remain to be transferred. Because the Application wants the second audio snippet, it must discard the first one, and does this by making a call to DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER. The snippet that it wants is now available to be transferred, and it does this with a call to DG_AUDIO / DAT_AUDIONATIVEXFER / MSG_GET. The Source moves up into state 7. The Application ends the transfer with a call to DG_CONTROL / DAT_PENDINGXFERS / MSG_ENDXFER.

Because the Application only wanted the second audio snippet, it can return to DG_IMAGE by making a call to DG_CONTROL / DAT_XFERGROUP / MSG_SET. Once this is done, all other commands work in a traditional TWAIN fashion. The Application can opt to transfer or discard the image, even though it did not transfer all of the audio snippets.

There is one more thing to note, if the Application had read the third audio snippet, or if it had issued the DG_CONTROL / DAT_PENDINGXFERS / MSG_RESET command while in DG_AUDIO, the state of the Source would remain at state 6. TWAIN works this way because it is image-centric, the only way to transition from state 6 to state 5 is when it is determined that there are no more images to transfer.

Notes

1. TWAIN 1.8 supports native and file transfers of audio snippets. Buffered mode transfers are not supported, because TWAIN does not have the necessary infrastructure to describe audio data, and it was decided that adding that structure in this release would be overly complex, and probably incomplete.
2. As a general rule, even though many operations are possible with DAT_XFERGROUP set to DG_AUDIO, Applications are encouraged to only change to DG_AUDIO for the length of time it takes to collect an audio snippet, and to stay in DG_IMAGE mode at all other times.
3. Though TWAIN is image-centric, it is possible to envision a TWAIN Source that is only capable of supporting DG_AUDIO. The TWAIN Working Group feels that any such notion is a bad idea, and encourages anyone thinking of doing this to pick on some other API.

How to use the Preview Device

Application Switch to the Preview Logical Device

The application first tries to switch to the preview logical device using the DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY triplet with TWFY_CAMERAPREVIEW set in InputName field of TW_FILENAME structure. If the returned value is TW_SUCCESS, the application can proceed.

1. After the application successfully switches to the preview device, all subsequent capability negotiation is with the preview device.
2. The application queries the Source with capability CAP_CAMERAPREVIEWUI. If it returns SUCCESS, then the Source is able to assume the responsibility of displaying preview images. The application can choose to use the Source's UI or not when it issues the MSG_ENABLEDS. If the application uses the Source's UI, it will do nothing but wait to issue MSG_DISABLEDS, or wait for a MSG_CLOSEDSREQ from the Source to stop the preview mode. If the application does not use the Source's UI or the Source does not provide a UI, then the application should follow the following steps.

Setting Up Environments for Preview Mode

1. The application starts negotiation on the Preview size using the ICAP_XRESOLUTION and ICAP_YRESOLUTION capabilities with MSG_GET first. With the returned supported sizes from

the Source, the application can set the selected preview sizes using the ICAP_XRESOLUTION and ICAP_YRESOLUTION capabilities with MSG_SET.

2. Optionally, the application can negotiate the zoom lens value, camera flash state during previewing, etc, with available capabilities such as ICAP_ZOOMFACTOR, ICAP_FLASHUSED2. If application queries for capabilities that are not related to preview device, Source will return TWRC_FAILURE.

Start Getting and Displaying Preview Thumbnails

1. The application can use the automatic capture feature with CAP_XFERCOUNT to -1 (Application is willing to transfer multiple images).
2. Application issues MSG_ENABLEDS to move to state 5. Upon receiving this message, the Source should start capturing images[1].
3. Source issues MSG_XFERREADY, indicating that an image is present, and state moves to 6.

LOOP:

4. Application issues DAT_IMAGENATIVEXFER to get image and goes to state 7.
5. Application issues MSG_ENDXFER to return to state 6, and it displays the image. Then if it wants the next preview image, examines pTW_PENDINGXFERS->Count to verify that there is another image, and it goes to LOOP. Source, upon receiving the MSG_ENDXFER message, takes the next picture and returns -1 in the pTW_PENDINGXFERS->Count.

END LOOP

6. If the application wants to end preview mode, it issues DAT_PENDINGXFERS / MSG_RESET. This forces the Source to go to state 5 (CAP_XFERCOUNT is set to 0). If the Source is unable to deliver preview images, it sets pTW_PENDINGXFERS->Count to 0 in reply to the application's MSG_ENDXFER command, and returns to state 5.
7. The application can then issue MSG_DISABLEDS, which returns it to state 4, and now the application can use DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY to change directory to the camera device to take a full resolution picture.

How to Take a Snapshot from Preview Scene

1. The application could provide a button or menu item for the user to take a snapshot from the preview scene, for example, a "Take Picture" button. In response to this, the application should use the triplet DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRECTORY with TWFS_CAMERA set in the TW_FILENAME structure to stop the preview mode.
2. Subsequently, the application can use the automatic capture feature with CAP_XFERCOUNT to 1, CAP_TIMEBEFOREFIRSTCAPTURE to 0 and CAP_AUTOMATICCAPTURE set to 1 to initiate the capture of preview snapshot.
3. When the Source receives the CAP_AUTOMATICCAPTURE, it should capture the preview snapshot, and inform the application with MSG_XFERREADY when it is ready to transfer.
4. After receiving the MSG_XFERREADY, the application should use one of the three standard image transfer methods to transfer the captured image from the Source to the application.

1. The Source takes a picture as soon as it receives MSG_ENABLEDS and each time it receives MSG_ENDXFER

5. At the end of this operation, the application has the option of going back to the preview thumbnail loop.

Imprinter / Endorser

Scanners intended for document imaging sometimes include accessories that let the scanner print data on the documents as it scans them. TWAIN provides basic functionality to negotiate capabilities for imprinter / endorser devices. An imprinter is a general term for any document-printing device. An endorser is more specialized, and is primarily intended as proof of scanning. In addition to the type of printing device, TWAIN offers ways to locate the printer on the scanning path: top or bottom of the sheet of paper, before or after the paper has been scanned. It is the responsibility of the Source to provide the available combinations to the Application. It is the responsibility of the Application to enable the printers that it wants to use, and to establish seed values prior to scanning.

This is a context sensitive scheme, Applications use `CAP_PRINTER` to discover what printers are available to the Source, and to select each of those printers for negotiation.

`CAP_PRINTERENABLED` determines whether or not a given printer will be used when scanning begins; a value of `TRUE` indicates that it will be used, a value of `FALSE` that it will not be used. Applications must enable a printer before negotiating the seed values.

`CAP_PRINTERINDEX` describes an index that counts by ones for every image seen by a given printer.

`CAP_PRINTERMODE` selects one of three options: print one line of text from `CAP_PRINTERSTRING`, or multiple lines from `CAP_PRINTERSTRING`, or a compound string constructed (in order) from `CAP_PRINTERSTRING`, `CAP_PRINTERINDEX` and `CAP_PRINTERSUFFIX`.

`CAP_PRINTERSTRING` specifies the base message to be printed. For compound strings, the `CAP_PRINTERSTRING` serves as the prefix to the `CAP_PRINTERINDEX`.

`CAP_PRINTERSUFFIX` is only available for compound strings, and describes the text (if any) that is to follow the `CAP_PRINTERINDEX`.

`CAP_PRINTERVERTICALOFFSET` specifies Y-Offset for current `CAP_PRINTER` device.

Example of Use:

Consider a Source that supports two `CAP_PRINTER`s:

`TWPR_IMPRINTERTOPBEFORE`

`TWPR_IMPRINTERBOTTOMBEFORE`

The Application then:

- uses `CAP_PRINTER` to discover the two printers
- sets `CAP_PRINTER` to `TWPR_IMPRINTERTOPBEFORE`

    - sets CAP_PRINTERENABLED to TRUE (turning this printer on)
    - sets CAP_PRINTERMODE to TWPM_SINGLESTRING
    - sets CAP_PRINTERSTRING to a string containing today's date
- sets CAP_PRINTER to TWPR_IMPRINTERBOTTOMBEFORE
    - sets CAP_PRINTERENABLED to FALSE (turning this printer off)

Note that the value of CAP_PRINTER is not important at the time of scanning, it is the other capabilities that control the imprinter, like CAP_PRINTERENABLED; CAP_PRINTER only selects the current printer under negotiation.

Capability Ordering

As the number of capabilities described by TWAIN has increased it has become clear that there are dependencies between many of them. The purpose of this section is to point out connections between certain capabilities. The way one capability can affect another is not always obvious and failure to recognize this interdependence is often the reason for unexpected TWAIN Scanning results. Using this as a guideline, an Application Developer can code capability negotiation with confidence, and Data Source developers can refer back to this section to make sure they have not introduced an unusual dependency.

In some cases these dependencies are not likely to be critical, for example if ICAP_CCITTKFACTOR is set to some non-zero value, and ICAP_COMPRESSION is not TWCP_GROUP32D, most scanners will not see this as a problem. On the other hand, if ICAP_COMPRESSION is set to TWCP_JPEG and ICAP_XFERMECH is set to TWSX_NATIVE then it is extremely unlikely that the Application will get useable image data.

It is the responsibility of the Source to properly constrain itself according to the current settings of all of its capabilities. Doing so has the following benefits:

- The Source protects itself from illegal configurations.
- The Source reports to the Application through constraints and the TWCC_CAPSEQERROR condition code which capabilities are fully, partially or currently not negotiable.

It is the responsibility of the Application to negotiate capabilities in the proper order. Doing so has the following benefits:

- The Application protects itself from illegal configurations.
- The Application can use constraints and occurrences of TWCC_CAPSEQERROR to modify the behavior of its user interface, better representing the Source's capabilities to the user.

The reset of this article is written in the order of negotiation that an Application should use to control a Source.

One other note about interpreting this section, the entire list of capabilities is in the context of the Current File System Device. If the Current File System Device was changed using the DAT_FILESYSTEM triplets, the context of these capabilities is expected to change and re-negotiation must occur. It is much easier to deal with if the File System operations are completed first and Capability negotiation on a large scale is left until just before scanning from a particular device.

Language Support

The first thing the Source and Application should negotiate is the language. This negotiation best occurs as part of the DG_CONTROL / DAT_PARENT / MSG_OPENDS call. The Application reports the language it is using in appIdentity->Version.Language. The Source should attempt to try to match this language. If it cannot, it should attempt to match the language that the user logged in with. If this fails then is should pick the best language that it can. For those Sources that support CAP_LANGUAGE the Application has a further opportunity to try and get a good language match, and this should be done as soon as the Source is successfully opened.

Duplex Control

If an Application finds that CAP_DUPLEX exists and it indicates that duplex scanning is supported, then the Application should negotiate CAP_DUPLEXENABLED. If CAP_DUPLEXENABLED is set to FALSE, then DAT_FILESYSTEM capable Sources should not report any TWFY_CAMERABOTTOM devices in the root directory. If the Source is set to a TWFY_CAMERABOTTOM device at the time that CAP_DUPLEXENABLED is set to FALSE, then it should automatically change itself to the corresponding TWFY_CAMERATOP device.

Device Negotiation

If the Source supports DAT_FILESYSTEM, then the Application needs to walk through the root directory to determine what devices are available, if it wants to independently control the individual devices. Sources are required to default to the TWFY_CAMERA device (the implied default for Sources that do not support DAT_FILESYSTEM). If an Application negotiates capabilities using this device, then the Source is expected to apply the settings to as many of its applicable devices as possible. For instance, in a duplex scanner changing the value of ICAP_BRIGHTNESS for the default TWFY_CAMERA device will change the settings of its corresponding TWFY_CAMERATOP and TWFY_CAMERABOTTOM. Once the list of devices has been identified, the Application may optionally change to one of them using DG_CONTROL / DAT_FILESYSTEM / MSG_CHANGEDIRCTORY.

Supported Capabilities

Applications are encouraged to use this call to get the capabilities supported by a Source, since this information can be used to quickly characterize the device. For instance, a Source that supports ICAP_FLASH2 is more likely to be a digital camera than a scanner. Or in another case, a single-pass duplex scanner that supports DAT_FILESYSTEM access to both of its cameras might not support all the same capabilities for both cameras.

Extended Capabilities

Beginning with TWAIN 2.3 the Data Source always sets CAP_EXTENDEDCAPS to the array of capabilities that are negotiable in States 5, 6 and 7. The application reads this array, or (for legacy purposes) it can set the array to the desired values, and, if TWRC_CHECKSTATUS is returned, follow up to see which values were accepted.

Feeder Control

CAP_FEEDERENABLED is the key capability to determine if a Source supports an automatic document feeder (ADF). Once this value has been determined no special ordering is required to test most of the other values, although there are groupings worth noting. Some ADFs provide control over individual sheets of paper: CAP_CLEARPAGE, CAP_FEEDPAGE, CAP_REWINDPAGE. Some ADFs are supported by memory buffers built into the device: CAP_AUTOSCAN, CAP_MAXBATCHBUFFERS. Some ADFs are capable of detecting the presence of paper in the input bin: CAP_PAPERDETECTABLE, CAP_FEEDERLOADED. ICAP_FEEDERTYPE reports either the types of feeders available (in the case where there is a general type feeder only) or the scan types supported through the feeder. Some ADFs provide control over paper handling: CAP_PAPERHANDLING.

Frame Management

Before negotiating frame information an Application should first establish the unit of measurement using ICAP_UNITS. It should establish the ICAP_XRESOLUTION and ICAP_YRESOLUTION of the image, especially if TWUN_PIXEL is supported, since the reported values should vary with the pixel density. After that the Application should determine the physical limits of the Source using ICAP_MINIMUMHEIGHT, ICAP_MINIMUMWIDTH, ICAP_PHYSICALHEIGHT and ICAP_PHYSICALWIDTH.

Note: ICAP_MINIMUMHEIGHT, ICAP_MINIMUMWIDTH, ICAP_PHYSICALHEIGHT, and ICAP_PHYSICALWIDTH may vary depending on the source of the document. For example, when using a Flatbed / ADF combination scanner, the ADF path may permit longer documents to be scanned. In this case, values for these extents would be expected to differ for different values of CAP_FEEDERENABLED.

DAT_IMAGELAYOUT is required by all Sources. Most scanners support ICAP_SUPPORTEDSIZES (unlike digital cameras, which tend to not support physical measurements like inches and centimeters).

ICAP_SUPPORTEDSIZES is required to set itself to TWSS_NONE if frames are negotiated using either DAT_IMAGELAYOUT or ICAP_FRAMES.

ICAP_MAXFRAMES will report how many frames ICAP_FRAMES is capable of delivering per captured item.

ICAP_ORIENTATION is intended to tell a Source how the orientation of a sheet of paper fed into the scanner varies from the settings of its frame information. 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_OVERSCAN is intended as a way to capture image data beyond the usual boundaries of a scanned sheet of paper, and is primarily intended as an aid in deskewing images. The additional scan area is only reported with DAT_IMAGEINFO. The reason for negotiating this value after setting the other values listed above is that some Sources may reject overscan if certain dimensions are exceeded.

ICAP_AUTOMATICDESKEW will correct the rotation of an image, it may also affect the dimensions of the image as reported by DAT_IMAGEINFO. ICAP_AUTOMATICBORDERDETECTION reduces or removes the border generated around an image by the scanner scanning its own platen (the area not covered by the paper).

ICAP_UNDEFINEDIMAGESIZE may be set to TRUE by a Source depending on one or more of the previously negotiated capabilities. Applications need to remember that it is possible for images to exceed the width and height dimensions, which can impact the amount of allocated memory. It is also important to note that if the width is undefined and ICAP_XFERMECH is set to TWSX_MEMORY, then the Source is required to also set ICAP_TILES to TRUE.

Bar Code Negotiation

ICAP_BARCODEDETECTIONENABLED must be set before any of the other, related capabilities are made available. ICAP_SUPPORTEDBARCODETYPES should then be tested, to determine what bar-code values are supported by the Source. After that the bar-code capabilities may be negotiated in any order.

Patch Code Negotiation

ICAP_PATCHCODEDETECTIONENABLED must be set before any of the other, related capabilities are made available. ICAP_SUPPORTEDPATCHCODETYPES should then be tested, to determine what patch-code values are supported by the Source. After that the patch-code capabilities may be negotiated in any order.

Imprinter/Endorser Negotiation

CAP_PRINTER establishes what (if any) printer/endorsers are supported by the Source. Selecting one establishes a context for that printer/endorser that is used by all related capabilities. CAP_PRINTERENABLED turns the printer on or off; the printer must be on in order for the other settings to be negotiated. A Source may opt to refuse to enable a printer if ICAP_SUPPORTEDSIZES selects a document with a size that is not within the area of the printer.

CAP_PRINTERINDEX should be negotiated next. CAP_PRINTERMODE can then be determined, followed by CAP_PRINTERSTRING and CAP_PRINTERSUFFIX.

Scaling

ICAP_XSCALING should be negotiated before the ICAP_YSCALING.

Image Manipulation

ICAP_ROTATION is a specific request to the scanner to rotate the scanned image the indicated number of degrees. ICAP_MIRROR is another specific request to mirror the scanned image. ICAP_FLIPROTATION is used to properly orient images that flip orientation every other image.

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.

```
// Then negotiate these capabilities…
ICAP_PIXELTYPE
     or
ICAP_JPEGPIXELTYPE
       // Use of flash may affect other values…
       ICAP_FLASHUSED
       ICAP_FLASHUSED2
              ICAP_AUTOBRIGHT
                   ICAP_BRIGHTNESS
              ICAP_BITDEPTH
              ICAP_BITDEPTHREDUCTION
                   ICAP_CUSTHALFTONE
                   ICAP_HALFTONES
                   ICAP_THRESHOLD
              ICAP_BITORDER
              ICAP_COMPRESSION
                   ICAP_BITORDERCODES
                   ICAP_CCITTKFACTOR
                   ICAP_PIXELFLAVORCODES
                   ICAP_TIMEFILL
              ICAP_CONTRAST
              ICAP_EXPOSURETIME
              ICAP_FILTER
              ICAP_GAMMA
              ICAP_IMAGEFILTER
              ICAP_NOISEFILTER
              ICAP_PIXELFLAVOR
                   ICAP_HIGHLIGHT
                   ICAP_SHADOW
              ICAP_PLANARCHUNKY
              ICAP_XRESOLUTION
                   ICAP_XNATIVERESOLUTION
                   ICAP_YRESOLUTION
                         ICAP_YNATIVERESOLUTION
```

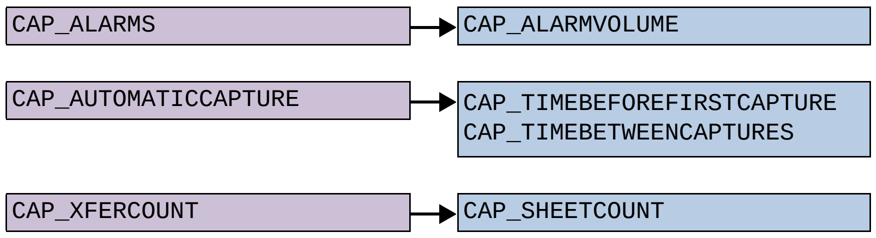
Independent Capabilities

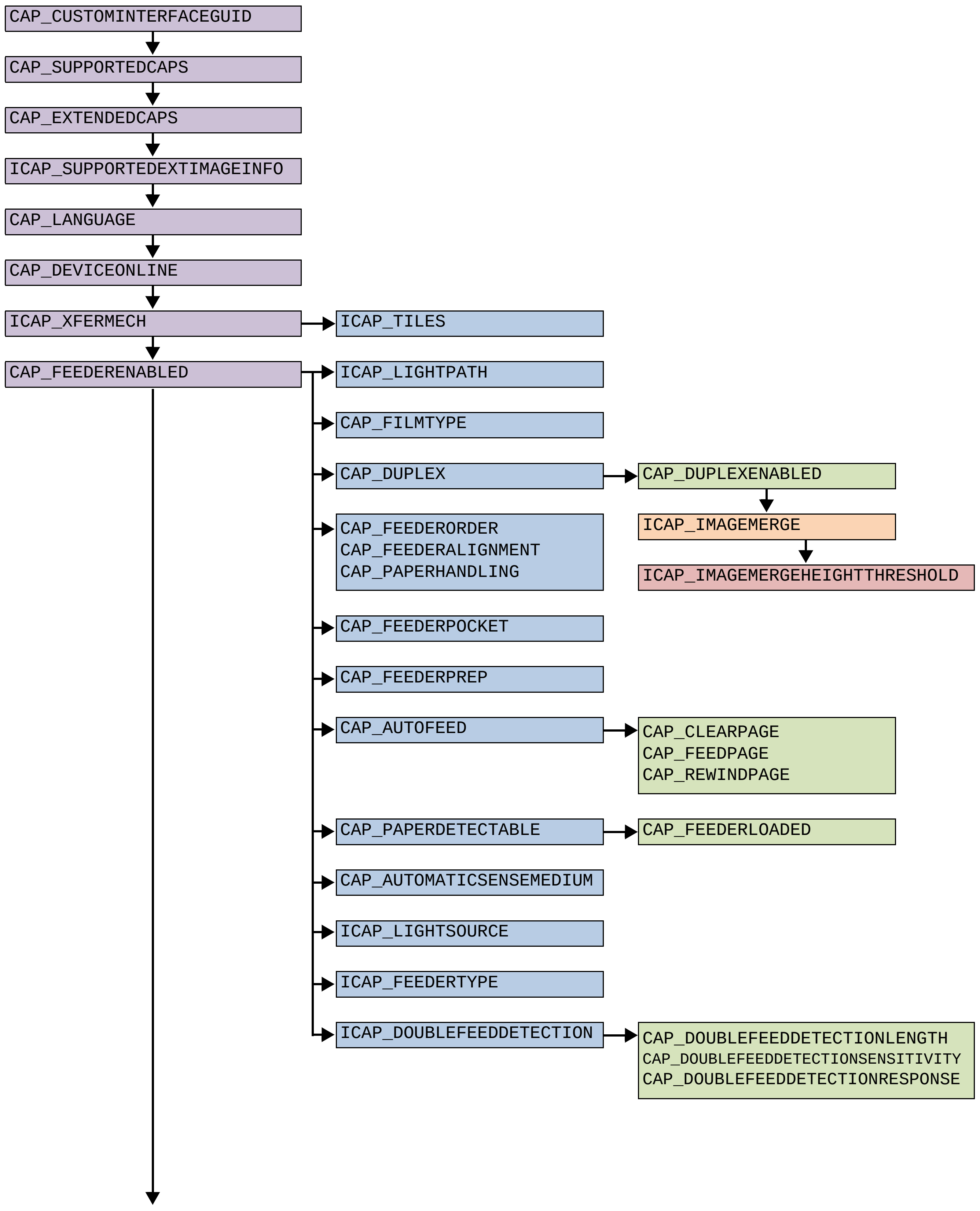
These capabilities are considered independent because they do not affect other capabilities and they are not affected by changes in other capabilities.

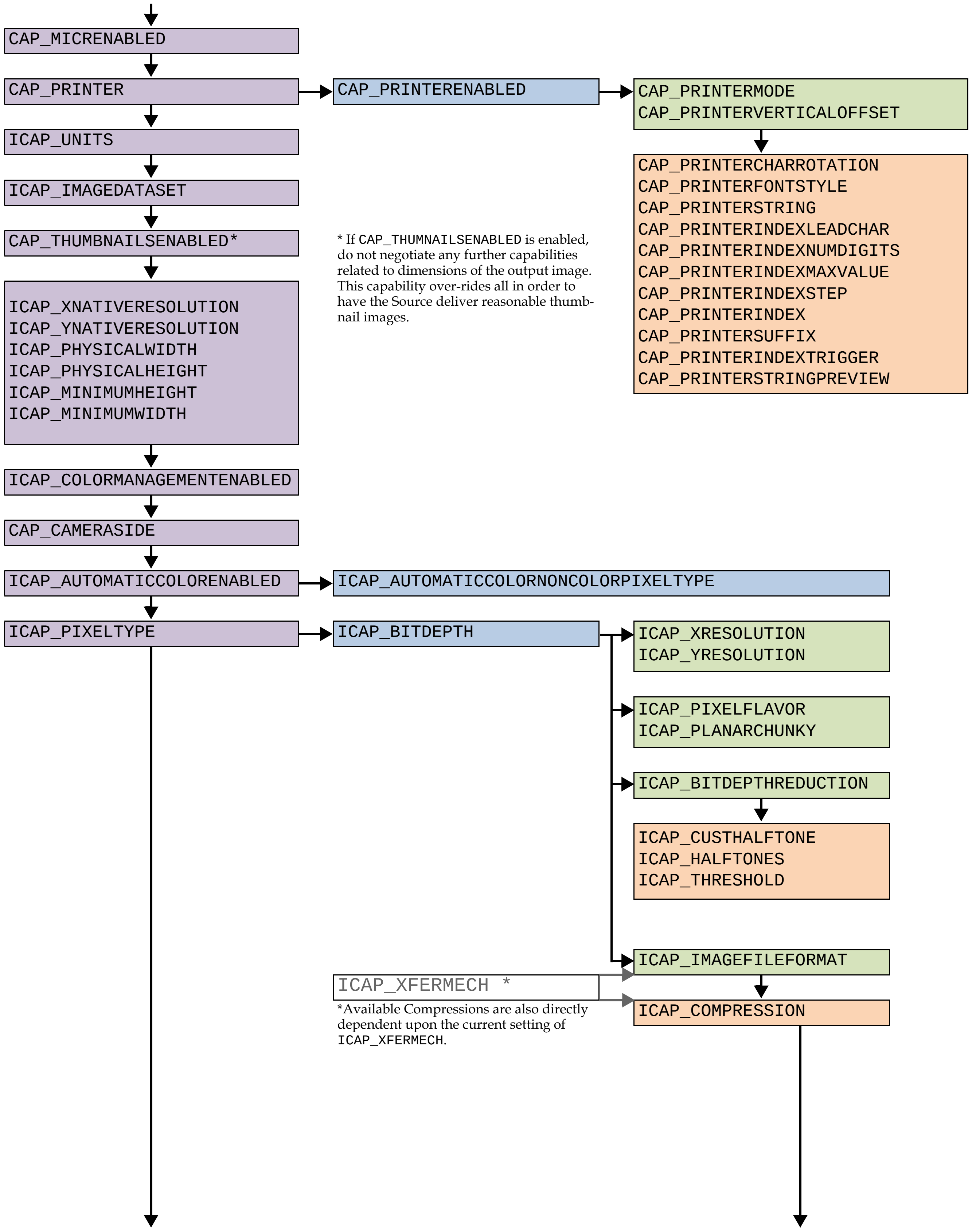
```
CAP_ENABLEDSUIONLY
CAP_CUSTOMDSDATA
CAP_INDICATORS
CAP_INDICATORSMODE
CAP_UICONTROLLABLE
CAP_SERIALNUMBER
ICAP_LAMPSTATE
CAP_BATTERYMINUTES
CAP_BATTERYPERCENTAGE
CAP_POWERSUPPLY
ICAP_BITORDER
CAP_DEVICETIMEDATE
CAP_DEVICEEVENT
CAP_CAMERAPREVIEWUI
CAP_POWERSAVETIME
ICAP_AUTODISCARDBLANKPAGES
ACAP_XFERMECH
```

Semi-Independent Capabilities

Semi Independent Capabilities are small groups that have no effect on the big picture, but do have their own pockets of dependencies.

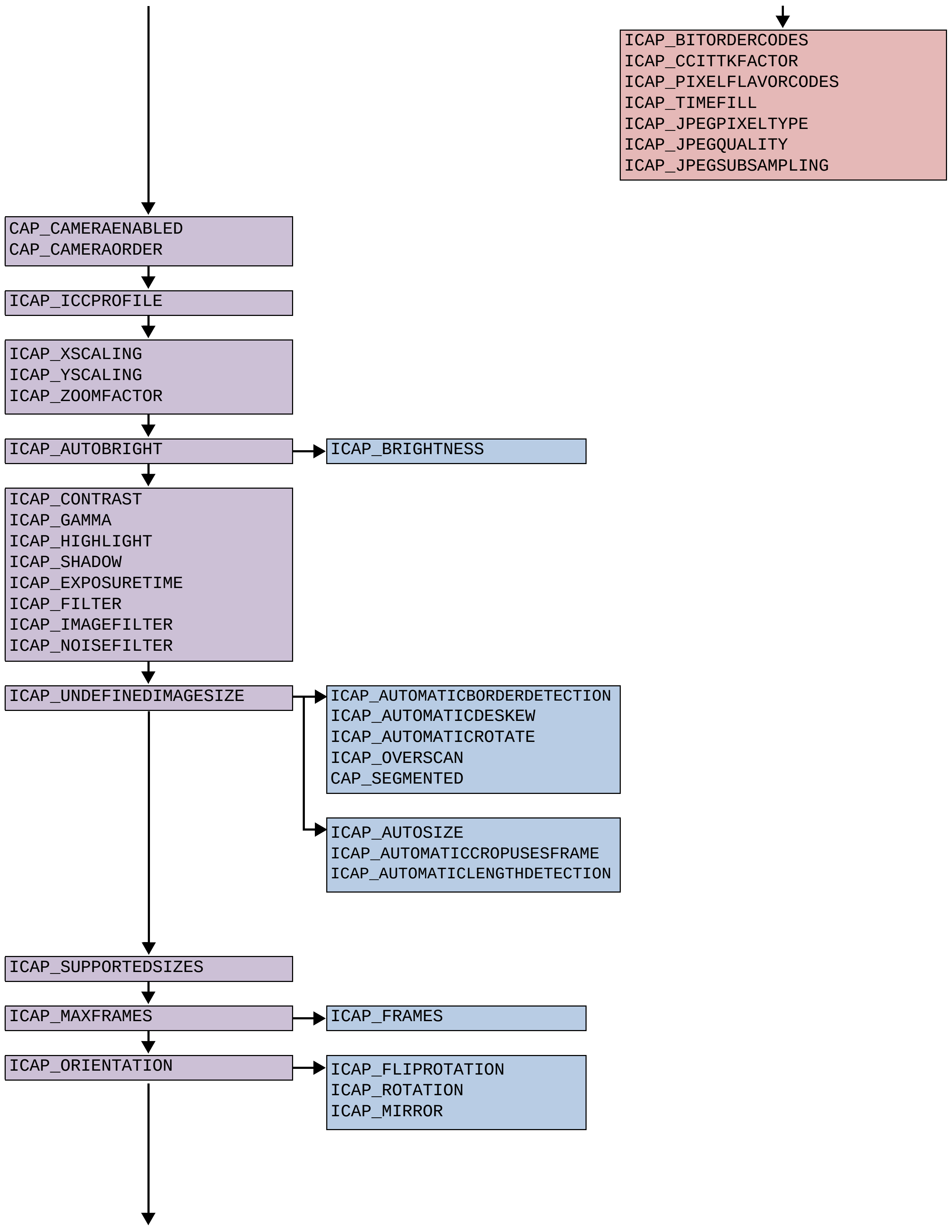


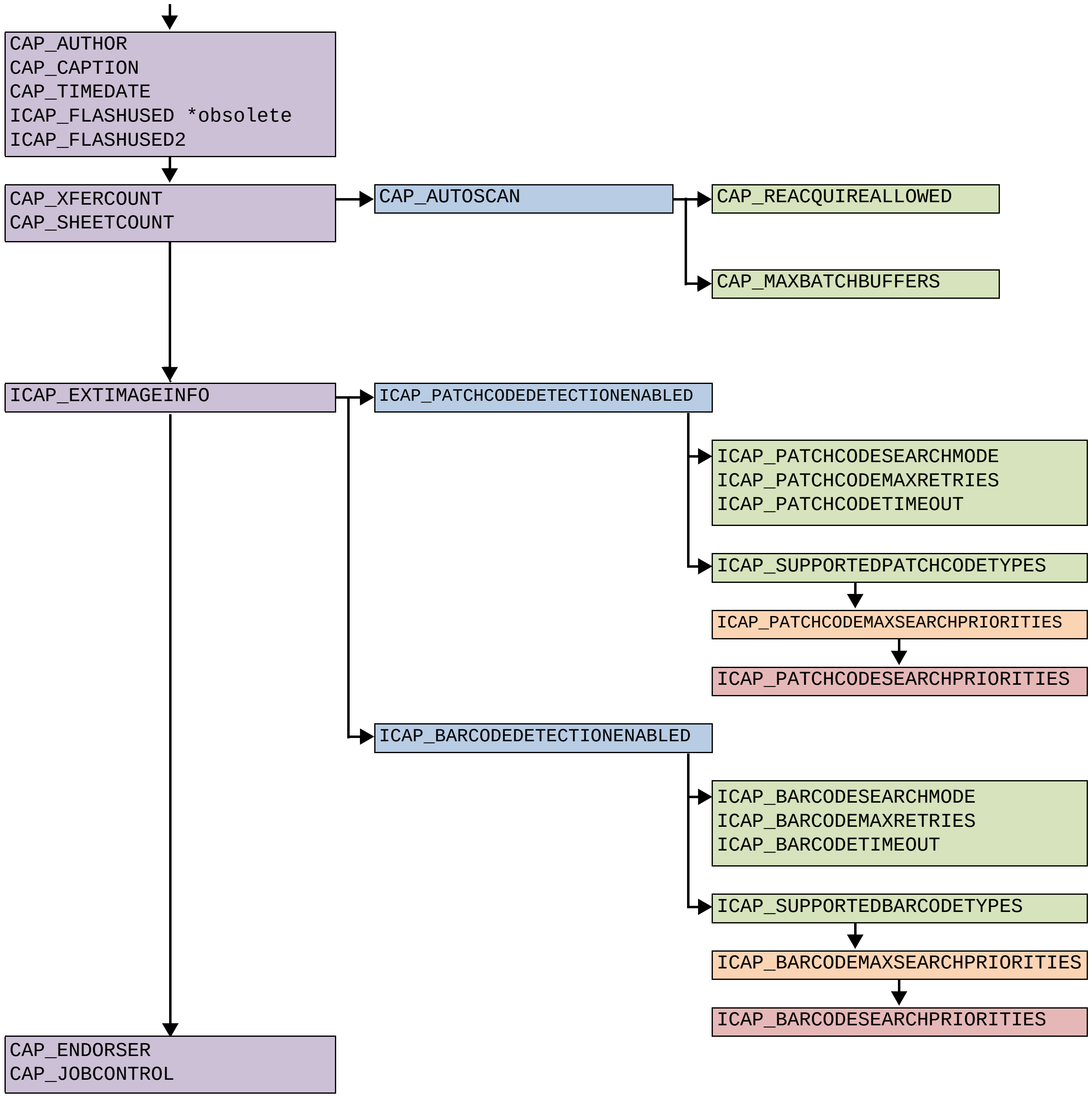

CAP_CUSTOMINTERFACEGUID
CAP_SUPPORTEDCAPS
CAP_EXTENDEDCAPS
ICAP_SUPPORTEDEXTIMAGEINFO
CAP_LANGUAGE
CAP_DEVICEONLINE
ICAP_XFERMECH
ICAP_TILES
CAP_FEEDERENABLED
ICAP_LIGHTPATH
CAP_FILMTYPE
CAP_DUPLEX
CAP_DUPLEXENABLED
ICAP_IMAGEMERGE
ICAP_IMAGEMERGEHEIGHTTHRESHOLD
CAP_FEEDERORDER
CAP_FEEDERALIGNMENT
CAP_PAPERHANDLING
CAP_FEEDERPOCKET
CAP_FEEDERPREP
CAP_AUTOFEED
CAP_CLEARPAGE
CAP_FEEDPAGE
CAP_REWINDPAGE
CAP_PAPERDETECTABLE
CAP_FEEDERLOADED
CAP_AUTOMATICSENSEMEDIUM
ICAP_LIGHTSOURCE
ICAP_FEEDERTYPE
ICAP_DOUBLEFEEDDETECTION
CAP_DOUBLEFEEDDETECTIONLENGTH
CAP_DOUBLEFEEDDETECTIONSENSITIVITY
CAP_DOUBLEFEEDDETECTIONRESPONSE



* If CAP_THUMNAILSENABLED is enabled, do not negotiate any further capabilities related to dimensions of the output image. This capability over-rides all in order to have the Source deliver reasonable thumbnail images.

*Available Compressions are also directly dependent upon the current setting of ICAP_XFERMECH.


ICAP_BITORDERCODES
ICAP_CCITTKFACTOR
ICAP_PIXELFLAVORCODES
ICAP_TIMEFILL
ICAP_JPEGPIXELTYPE
ICAP_JPEGQUALITY
ICAP_JPEGSUBSAMPLING
CAP_CAMERAENABLED
CAP_CAMERAORDER
ICAP_ICCPROFILE
ICAP_XSCALING
ICAP_YSCALING
ICAP_ZOOMFACTOR
ICAP_AUTOBRIGHT
ICAP_BRIGHTNESS
ICAP_CONTRAST
ICAP_GAMMA
ICAP_HIGHLIGHT
ICAP_SHADOW
ICAP_EXPOSURETIME
ICAP_FILTER
ICAP_IMAGEFILTER
ICAP_NOISEFILTER
ICAP_UNDEFINEDIMAGESIZE
ICAP_AUTOMATICBORDERDETECTION
ICAP_AUTOMATICDESKEW
ICAP_AUTOMATICROTATE
ICAP_OVERSCAN
CAP_SEGMENTED
ICAP_AUTOSIZE
ICAP_AUTOMATICCROPUSESFRAME
ICAP_AUTOMATICLENGTHDETECTION
ICAP_SUPPORTEDSIZES
ICAP_MAXFRAMES
ICAP_FRAMES
ICAP_ORIENTATION
ICAP_FLIPROTATION
ICAP_ROTATION
ICAP_MIRROR



Audio Negotiation

The availability of the audio capabilities can be inferred from the presence of DG_AUDIO. If it is available then the Application should negotiate ACAP_XFERMECH. Note that these operations occur independently of the current value of DAT_XFERGROUP. The actual selection of an audio file format takes place in State 6 using DAT_SETUPFILEXFER, and must be preceded by a call to DAT_XFERGROUP / MSG_SET to DG_AUDIO to change the Source over to the audio data group. Sources that transfer audio data need to set the Source back to DG_IMAGE when they are done with the audio data, and ready to get image data, or exit back to State 4.

Alarms

CAP_ALARMS selects the kind of audio alerts provided by a Source. CAP_ALARMVOLUME is only available if an alarm is selected, and controls the volume for all alarms with a single value.

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.

Asynchronous Device Events

CAP_DEVICEEVENT may be used to activate device events.

Automatic Capture

DG_CONTROL / DAT_FILESYSTEM / MSG_AUTOMATICCAPTUREDIREDCTORY should be negotiated first, since it selects the destination for the images. CAP_TIMEBEFOREFIRSTCAPTURE and CAP_TIMEBETWEENCAPTURES should be negotiated next. CAP_AUTOMATICCAPTURE must be negotiated last, because it is the trigger that starts the timer.

Camera-Dependent Capabilities

The following list covers capabilities have no interdependencies, but which may be dependent on the currently selected device (for Sources that support DAT_FILESYSTEM).

CAP_AUTHOR

CAP_CAMERASIDE

CAP_CAPTION

CAP_DEVICETIMEDATE

CAP_ENDORSER

CAP_JOBCONTROL

CAP_PASSTHRU

CAP_SERIALNUMBER

CAP_SHEETCOUNT

CAP_TIMEDATE

CAP_XFERCOUNT

CAP_INDICATORSMODE

ICAP_EXTIMAGEINFO

ICAP_IMAGEDATASET

ICAP_LAMPSTATE

ICAP_LIGHTPATH

ICAP_LIGHTSOURCE

ICAP_ZOOMFACTOR

Camera-Independent Capabilities

The following list covers capabilities that are free of any dependencies. Applications can negotiate these in any order (during state 4), and regardless of the current device in effect (for Sources that support DAT_FILESYSTEM):

CAP_CAMERAPREVIEWUI

CAP_CUSTOMDSDATA

CAP_DEVICEONLINE

CAP_DEVICETIMEDATE

CAP_ENABLEDSUIONLY

CAP_INDICATORS

CAP_PASSTHRU

CAP_SEGMENTED

CAP_SHEETCOUNT

CAP_THUMBNAILSENABLED

CAP_TIMEDATE

CAP_UICONTROLLABLE

CAP_XFERCOUNT

Defaults

TWAIN describes defaults for capabilities, unfortunately, this information is spread throughout the specification, and in some cases is ambiguous. This article discusses how Sources and Applications should use and manage defaults values. It covers the three main kinds of defaults supported by TWAIN. It discusses the DG_CONTROL / DAT_CAPABILITY / MSG_xxxx functions and how they relate to defaults. Finally, it offers a section that describes the expected default settings for each capability within TWAIN.

Default Mechanisms

Defaults in TWAIN serve three main functions:

Mandatory Defaults	Protect Applications from incompatible settings.
Preferred Defaults	Permit Source providers to expose preferred settings for capabilities.
User Defaults	Create consistency in a Source's user interface; values selected in one session are preserved for the next session.

Mandatory defaults are established by the TWAIN specification. Preferred defaults may be selected by a Source for any capability that does not have a mandatory default. User defaults may replace any preferred default with a value selected by the user through the Source's user interface. These three functions are not intrinsically compatible, which creates ambiguity; Applications cannot make assumptions about the initial values of all capabilities.

Mandatory Defaults

Some capabilities must reflect certain values when a Source is opened. These defaults are selected because Applications must be allowed to expect certain kinds of behavior without being forced to negotiate all capabilities (not only would this be tedious, but it is impossible in situations where a Source and Application are derived from different versions of TWAIN). For example, the 1.8 capability `CAP_PRINTERENABLED` must default to `FALSE`, otherwise a 1.6 Application might find itself printing data on scanned documents, and unable to do anything about it.

Preferred Defaults

TWAIN permits a Source to provide its own defaults. These settings are assumed to produce the most favorable results possible, whether they are measured in terms of processing speed, memory usage, or some other criteria. For instance, a Source will select a preferred value for `ICAP_PIXELFLAVOR` that keeps it from having to invert the bits in an image. In some cases the preferred defaults are gleaned from the current state of the device. For instance, `CAP_FEEDERENABLED` depends on the presence of a feeder on the scanner device. A Source is expected to determine if the feeder is truly present, not assume that the value saved from the last session is valid.

User Defaults

Prior to TWAIN 1.7 Applications generally relied on Sources to provide user interfaces that controlled image capture. Since one of the tenants of TWAIN is to make things easier for Applications, it became common for Sources to save state, preserving the values selected by a user, so the next time the Application started the same values would be displayed. This mechanism continues to be desirable, but Source writers should bear in mind that user defaults values are a convenience that can create problems for users who access their Source from more than one Application. `CAP_AUTOSCAN` is an example of a capability that should never have its state saved, since Applications that do not negotiate will also not be able to handle the results if it happens to be set to `TRUE`.

DAT_CAPABILITY Operations

There are five methods of negotiating values with a Source, this section discusses how Sources and Applications should relate them to the various kinds of default values:

- `MSG_GET` returns the current value of the capability, along with the allowed values (if any). At startup, this value will reflect the mandatory default, if there is one. If there is no mandatory default, then this call will return the user default, if supported and if one is available; otherwise it will return the preferred default value for the Source. It is up to the Application to understand the possible sources of a value, and override it if desired. Note that the allowed values are always reset when a Source starts up. Sources must never save the constraints created by an Application.
- `MSG_GETCURRENT` returns the current value of the capability, it does not return the allowed list. At startup, this value will reflect the mandatory default, if there is one. If there is no

mandatory default, then this call will return the user default, if supported and if one is available; otherwise it will return the preferred default value for the Source.

- `MSG_GETDEFAULT` always returns either the mandatory or preferred default, whichever is appropriate for the capability. It never returns a user default. Like `MSG_GETCURRENT` it only returns the value, not the allowed list.
- `MSG_RESET` resets a capability's allowed list to all permitted values, and sets the current value to the mandatory or preferred default, never the user default. The container returned by `MSG_RESET` must be the same kind of container returned for a `MSG_GETDEFAULT` operation, this preserves legacy behavior; however, Applications should follow `MSG_RESET` with `MSG_GET` if they wish to determine how the constraints for the capability have been reset. This message is a good one for Applications to use, since it is easy to code, and can be used to get a Source to some kind of a known state.

A simple mechanism for resetting a Source uses the following steps (*Applications that use the Source's UI should not use this method*): for each device supported by the Source (pre-1.8 Sources only have one implicit device) the Application calls `CAP_SUPPORTEDCAPS`; for each capability the Application calls `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_QUERYSUPPORT` to see if it supports `TWQI_RESET`; if it does, then the Application sends `DG_CONTROL` / `DAT_CAPABILITY` / `MSG_RESET` which resets the capability.

Performing these steps will protect an Application from any user defaults created by a previous Application. Please note, not all Sources may support `MSG_QUERYSUPPORT`. For those that don't, an Application would have to issue `MSG_RESET` on all capabilities (perhaps excluding those it knows to be read-only) and trust that the Source is robust enough to report `TWRC_FAILURE` for those capabilities that do not support `MSG_RESET`.

- `MSG_SET` sets the current value. Therefore, it's possible for a capability set in State 4 to find its way into the user defaults.
- `MSG_SETCONSTRAINT` sets the current value and optionally sets the constraints on a capability. Sources must never save the constraints negotiated by an application. However, it's possible for a capability set in State 4 to find its way into the user defaults.

B

TWAIN Technical Support

If you have a question or technical issue that isn't answered on the TWAIN.org website, ask the TWAIN community at http://twainforum.org

The TWAIN Forum is an online discussion platform where you can exchange ideas with the TWAIN community. We suggest reviewing previous messages in the forum or posting a new message for best results to technical questions

As an alternative, contact TWAIN directly at admin@twain.org.