PHOTOGRAPHIC AND IMAGING MANUFACTURERS ASSOCIATION, INC.

PIMA 15740:2000

Approved 2000-07-05

FIRST EDITION

Photography – Electronic still picture imaging - Picture Transfer Protocol (PTP) for Digital Still Photography Devices

Published by:
Photographic and Imaging
Manufacturers Association, Inc.
550 Mamaroneck Avenue, Suite 307
Harrison, NY 10528-1612 USA
Phone: (914) 698-7603
FAX: (914) 698-7609
E-mail: natlstds@pima.net (Standards Office)

The Association for Manufacturers of Image Technology Products

Foreword

The technical content of this PIMA standard is closely related to ISO 15740, which is currently in the working draft stage while work on multiple transports is being completed. The main difference is that PIMA 15740 includes an informative annex describing a USB implementation of ISO 15740. This information is not included in ISO 15740, which instead references the USB still device class document developed by the Device Working Group of the USB Implementers Forum. The USB annex in PIMA 15740 provides the same technical approach as the USB still device class specification. The reason for developing PIMA 15740 is to immediately provide a complete, fully documented, stable, publicly available specification for USB implementations of the Picture Transfer Protocol defined in ISO 15740. This will enable hardware and software manufacturers to immediately produce product implementations, without waiting for the ISO and USB documents to complete the approval process. This PIMA 15740 standard may be withdrawn once the ISO 15740 and USB still device class documents have been approved and are publicly available.

Introduction

For the purposes of this standard, digital still photography devices (DSPDs) are defined as devices with persistent storage that capture a digital two-dimensional image at a discrete point in time. Most DSPDs include interfaces that can be used to connect to a host computer or other imaging devices, such as a printer. A number of new, high speed interface transports have recently been developed, including IrDA, USB, and IEEE1394 (Firewire). This standard is designed to provide requirements for communicating with DSPDs. This includes communications with any type of device, including host computers, direct printers and other DSPDs over a suitable transport. The requirements include standard image referencing behavior, operations, responses, events, device properties, datasets, and data formats to ensure interoperability. This standard also provides optional operations and formats, as well as extension mechanisms.

This standard has been designed to appropriately support popular image formats used in digital still cameras, including the EXIF and TIFF/EP formats defined in ISO 12234-1 and ISO 12234-2, as well as the Design Rule for Camera File System (DCF) and the Digital Print Order Format (DPOF).

Purpose

Numerous DSPDs have been developed in order to allow direct capture of digital images for both consumer and professional photography applications. Most of these devices can interface to digital computers or other imaging devices, such as printers, using various standardized electrical digital interfaces. However, there is currently no standard operation set to enable data transfer or control of DSPDs over these different interfaces. As a result, proprietary software is typically developed to control image capture devices from specific manufacturers, and different DSPD drivers are needed to support each interface. Standardizing the operations and data requirements for DSPDs will assist transport implementers, platform aggregation of conforming devices across all transports, and device manufacturers by providing a common ground for interface support. It will also assist developers of host software and image receiving devices by ensuring that their products can interface to many different DSPDs from different manufacturers, and assist users by ensuring that the DSPDs they purchase will inter-operate with those of different manufacturers.

The primary purpose of this standard is to provide a common protocol for any device, including DSPDs, to exchange images with a DSPD, either by retrieving images from a DSPD or by sending images to or from a DSPD. Secondary purposes include a mechanism for devices to control DSPDs (e.g. a PC can request that a DSPD change its shutter duration setting and capture a new picture) and the ability to transfer auxiliary information such as non-image data files and associated information, such as a digital print order file (DPOF).

Contributors

The following experts have contributed to this technical specification:

Steinberg, Eran	FotoNation, Inc. (Project Leader)
Looney, Timothy	Eastman Kodak Company (Editor)
Whitcher, Timothy	Eastman Kodak Company (USB Annex D Editor)
Anderson, Blair	Ajilon Information Technology Services
Armstrong, Frank	Eastman Kodak Company
Bitz, Mike	Apple Computer, Inc.
Coppola, Steven	Eastman Kodak Company
Edwards, Eric	Sony Corporation
Fontani, Paolo	Hewlett Packard Company
Foshee, Scott	Adobe Systems, Inc.
Fujisaki, Hirohisa	Eastman Kodak, Japan
Hong, Fang	Seiko Epson Corporation
Hsieh, William	Microsoft Corporation
Kazunori, Suenaga	Seiko Epson Corporation
Kuo, David	Flashpoint Technology, Inc.
Lawrence, David	Smart Technology Enablers, Inc.
Lyon, Lonne	Eastman Kodak Company
Melville, John	Eastman Kodak Company
Myers, Paul	Questra Corporation
Parsons, Dave	Microsoft Corporation
Parulski, Ken	Eastman Kodak Company
Reus, Edward	Microsoft Corporation
Sadovsky, Vladimir	Microsoft Corporation
Shidate, Ichiro	Nikon Corporation

Table of Contents

Index of Figures

Annex Figures

Index of Tables

Annex Tables

1 Scope

It is the objective of this standard to provide a common communication mechanism for exchanging images with and between digital still photography devices (DSPDs). This includes communication between digital still photography devices and host computers, printers, other digital still devices, telecommunications kiosks, and image storage and display devices.

This standard presents a protocol that is intended to be transport and platform independent. The purpose of this intent is to enable standard behavior by allowing implementation of the protocol in a variety of standard transports. In this document three transports are referenced: USB (Universal Serial Bus), IEEE 1394, and IrDA (Infrared Data Association). The protocol is by no means limited to these transports. This standard specifies the following:

- Behavior requirements for DSPDs. This includes the baseline features a device needs to support to provide interoperability over conforming transports.
- Functional requirements needed by a transport to enable the creation of a transport-dependent implementation specification that conforms to this standard.
- A high-level protocol for communicating with and between DSPDs consisting of operation, data, and response phases.
- Sets of suggested data codes and their usages including:
  - OperationCodes
  - ResponseCodes
  - ObjectFormatCodes
  - DevicePropCodes
  - EventCodes
- Required datasets and their usages.
- A means for describing data object associations and filesystems.
- Mechanisms for implementing extensibility.

This standard does not attempt to define any of the following:

- Any sort of device discovery, enumeration, or transport aggregation methods. Implementation of this functionality is left to the transports and the platforms upon which support for this standard is implemented.
- An application programming interface. This is left to the platforms upon which support for this standard is implemented.

2 Normative References

The following standards contain provisions, which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid standards.

IEC 61966-2-1:1999, *Multimedia systems and equipment – Colour measurement and management – Part 2-2: Colour management - Default RGB colour space - sRGB*

ISO 8601: 1988 *Data elements and interchange formats - Information interchange - Representation of Dates and times*

ISO 12234-1 *Photography - Electronic still picture cameras - Removable memory, Part 1: Basic removable memory reference model*[1]

ISO 12234-2 *Photography - Electronic still picture cameras - Removable memory, Part 2: Image data format - TIFF/EP*[1]

ISO/IEC 10918-1: 1994 *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*

ISO/IEC 10646-1:1993 *Information technology -- Universal multiple-octet coded character set (UCS) -- Part 1: Architecture and basic multilingual plane*

[1] To be published

3 Terms & Definitions

For the purpose of this standard, the following definitions shall apply:

album: End-user created object that is used to logically group data objects according to some user-defined criteria. An album may or may not be a physical folder in a filesystem. In this specification, album is a type of association.

Application Programming Interface: (API). High-level functional description of a software interface that is typically language dependent.

association: Logical construct used to expose a relationship between discrete objects. In this specification, associations are used to indicate that separate data objects are related. Examples include a time sequence, or user-defined groupings by content or capture session. Associations are represented like folders, and may be nested using a standard branched hierarchical tree structure.

connection: Transport-provided mechanism for establishing paths for transferring data between devices.

datacode: 16-bit unsigned integer whose Most Significant Nibble (4 bits) is used to indicate the category of code, and whether the code value is standard or vendor-extended.

data object: Image or other type of data that typically exists in persistent storage of a DSPD or other device**.**

dataset: Transport-independent collection of one or more individual data items with known interpretations. Data sets are not necessarily opaque nor atomic to transport implementations.

DCF: *See Design rule for Camera File system***.**

Design rule for Camera File system (DCF): Standard convention for camera file systems that specifies the file format, foldering, and naming conventions in order to promote file interoperability between conforming digital photography devices.

Device discovery: Act of determining the set of all devices present on a particular transport or platform that are physically or logically accessible.

Digital Still Photography Device (DSPD): Device with persistent storage that captures a two-dimensional digital still image.

Digital Print Order Format (DPOF): Standardized ASCII file stored on removable media along with the image files that indicates how many copies of which images should be printed. It also allows index prints, cropping, and text overlays to be specified.

DSPD: *See Digital Still Photography Device***.**

enumeration: Act of creating an ordered increasing numerical list that contains one representative element for each member of a set.

Exif/JPEG: Compressed file format for digital cameras in which the images are compressed using the baseline JPEG standard, described in ISO 12234, and metadata and thumbnail images are stored using TIFF tags within an application segment at the beginning of the JPEG file.

folder: Optional sub-structure in a hierarchical storage area that can contain data objects.

FlashPix: Image file format, defined in *FlashPix Format Specification*, using a structured storage file containing metadata and a tiled, hierarchical image representation. The tiles are normally baseline JPEG images, and individual image tiles of a particular resolution can be easily accessed for rapid display and editing.

ICC profile: Data file that characterises the colour characteristics of an image capture or image output device

IEEE 1394: High-speed serial bus standardised by the IEEE (Institute of Electrical and Electronics Engineers) currently having clock rates of 100, 200, and 400 Mbits/sec.

image aspect ratio: Ratio of the image width to the image height.

image capture device: Device, such as a digital still picture camera or a scanner, for converting a scene or a fixed image such as a print, film, or transparency, to digital image data.

image output device: Device, such a printer or film recorder that can render a digital image to hardcopy media.

in-band events: Transport only provides one logical connection, so events are transmitted on the same logical connection as operations and responses. Events are therefore only asynchronous to the degree of data granularity for which the transport implementation allows event interleaving.

Initiator: Device that initiates a conversation by opening a session, and issues all formal operations to the Responder. The Initiator is analogous to the client in the client/server paradigm.

IrDA: Infrared Data Association. Infrared wireless communication system that currently supports wireless communication at data rates between 9600bps and 4Mbps.

JPEG: Joint Photographic Experts Group. Image compression method defined in ISO/IEC 10918-1.

LogicalStorageID: Least significant sixteen bits of a StorageID. This value uniquely identifies one logical storage area within the physical store indicated in the PhysicalStorageID.

Most Significant Nibble (MSN): Most-significant four bits of the most-significant byte.

MSN: *See Most Significant Nibble.*

object aggregation: Act of taking one or more location-specific lists of objects that exists on a particular device and grouping them together into one set.

ObjectHandle: Device-unique 32-bit unsigned integer assigned by a device to each data object in local persistent storage. This handle is provided to external devices, which must use it to reference that piece of data in subsequent transactions. ObjectHandles are guaranteed to be persistent over at least a session.

out-of-band events: Transport provides a separate logical connection for transmitting events, and events are therefore asynchronous from operation transactions.

PC: Personal Computer. Personal computing device, which may employ various hardware architectures and operating systems.

PhysicalStorageID: Most significant sixteen bits of a StorageID. This value uniquely identifies one physical storage area on a device, although there may be more than one logical store per physical store.

PIMA: Photographic & Imaging Manufacturer's Association. This organisation serves to represent the common interests among manufacturers of imaging technology products. See http://www.pima.net.

PNG: Portable Network Graphics. Extensible file format for lossless, portable, compressed storage of raster images. PNG supports indexed-color, grayscale, truecolor, and an optional alpha channel.

protocol: Defined mechanisms for exchanging data between devices.

pull model: Use paradigm for DSPDs where the object receiver initiates the operation requests to transfer data objects from the sender.

push model: Use paradigm for DSPDs where the object sender initiates the operation requests to transfer data objects to the receiver.

QuickDraw picture: File format consisting of sequences of saved drawing commands, commonly referred to as PICT.

Responder: Device that responds to operations from the Initiator. The Responder is analogous to the server in the client/server paradigm.

session: Logical connection between two devices defining a period of time during which obtained state information, such as handle persistence, may be relied upon.

square pixel sampling: Image having equal sample spacing in the two orthogonal sampling directions.

StorageID: Device-specific four byte unsigned integer (UINT32) that represents a unique storage area that may contain data objects. The most significant sixteen bits of a StorageID represents the PhysicalStorageID, while least significant sixteen bits of a StorageID represents the LogicalStorageID.

transport aggregation: Act of taking one or more transport-specific lists of conforming devices that are logically or physically accessible in a system and grouping them into one set that spans all transports across the particular system.

transport: Means of attaching the digital capture device to some other digital device. It may include a physical wire or a wireless connection.

USB: Universal serial bus, a digital interface for connecting up to 127 devices in a tiered-star topology. See http://www.usb.org.

4 Digital Still Photography Device Model

4.1 Overview

Digital still photography devices (DSPDs) are used to acquire digitally encoded still images. These devices include persistent storage capability so that any digital images and other data acquired by the device are preserved across power cycle operations unless they are specifically deleted.

A DSPD might support many different features. This standard supports devices with a wide range of potential features. However, a small number of features are required for conformance with this standard, while many others are optional. Clause 4.2, **Baseline Requirements,** describes the required features and functionality. Annex C, **Optional Device Features**, describes features that are not required for conformance, but should be implementable using this standard and its extension mechanisms.

Standard data formats for datatypes and datasets are described in Clause 5, **Data Format Specification**.

Clause 6, **Image and Data Object Formats**, describes required and optional support for particular image and non-image formats and metadata. This clause also describes methods for associating data objects.

A particular feature set places requirements on the transports used to connect the DSPD to other devices. Clause 7, **Transports,** describes these requirements.

All DSPDs must store images in some form of storage area. Clause 8, **Persistent Storage**, describes the usage of these stores, as well as the methods for referencing them.

Clause 9, **Communication Protocol**, describes the roles of devices, sessions, and transactions that transports are required to use in order to communicate with and/or between DSPDs. Clause 10, **Operations**, lists the standard operations, their corresponding optional operation codes, and their usages. Standard responses to operations are defined in Clause 11, **Responses**. The use of events is mandatory in order to ensure synchronization between devices. Clause 12, **Events**, describes events and their usages.

In order to expose device controls and manipulate properties in a common way, a standard set of device properties and their usages have been defined in Clause 13, **Properties**.

Clause 14, **Conformance**, serves as a summary of the individual operations and events that are required to be supported by particular devices, as well as a checklist that can be used by implementers.

4.2 Baseline Requirements

This section lists the requirements that must be met in order for a DSPD to conform to this standard.

4.2.1 Implementation of a suitable transport

The DSPD shall provide appropriate hardware and software support for at least one transport that meets the requirements specified in Clause 6, **Transports**.

4.2.2 Thumbnail Support

The DSPS shall provide support for thumbnails as described in Clause 6.1, **Thumbnail Image File Formats**.

4.2.3 Standard Image and Data Reference Behavior

In order to ensure interoperability, it is necessary to define a standard mechanism for describing image and data objects present on a device. The DSPD shall meet the requirements described in Clause 6, **Image and Data Object Formats**.

4.2.4 Asynchronous Event Support

The DSPD shall be capable of generating and reacting to asynchronous events. Clause 12, **Events**, describes events and their usages.

5 Data Format Specification

In order to ensure interoperability, this standard provides conventions for encoding relevant datatypes and datasets.

5.1 General Format

5.1.1 Multibyte Data

For the purposes of interpretability, all data fields showing internal content representations shall be read from left to right, in order of decreasing byte significance, commonly referred to as big-endian notation. Therefore, the left-most byte shall represent the Most Significant Byte (MSB), and the right-most byte shall represent the Least Significant Byte (LSB). The most significant four bits of the MSB are referred to as the Most Significant Nibble (MSN), while the least significant four bits of the LSB are referred to as the Least Significant Nibble (LSN). The actual multibyte format used on the wire is transport-specific, while the actual multibyte format used at the application interface is platform-specific.

5.1.2 Bit Format

Bit fields presented in this standard are numbered so that the least significant bit is at the zero position, holding the right-most position in the field. For example, the most significant bit of a UINT32 would be referred to as bit 31, while the least significant bit would be referred to as bit 0.

5.1.3 Hexadecimal Notation

This standard uses hexadecimal notation as a means to concisely describe multibyte fields. All hexadecimal bytefields are represented with the prefix '`0x`'. Following this prefix are pairs of characters, where each pair represents one byte, with the most significant byte appearing first, and the least significant byte appearing last.

5.2 Datatype Summary

The following types of data are defined in this standard as having specific interpretations of their data content:

Table 1: Datatype Summary

Name	Size (Bytes)	Format
OperationCode	2	Datacode (UINT16)
ResponseCode	2	Datacode (UINT16)
EventCode	2	Datacode (UINT16)
DevicePropCode	2	Datacode (UINT16)
ObjectFormatCode	2	Datacode (UINT16)
StorageID	4	Special (UINT32)
ObjectHandle	4	Handle (UINT32)
DateTime	Variable	String
DeviceInfo	Variable	Dataset
StorageInfo	Variable	Dataset
ObjectInfo	Variable	Dataset
DevicePropDesc	Variable	Dataset
DevicePropDescEnum	Variable	Enumerated form of DevicePropDesc
DevicePropDescRange	Variable	Range form of DevicePropDesc
Object	Variable	Variable

5.2.1 Datacodes

Datacodes are 16-bit unsigned integers (UINT16) with specified interpretations, used for the purposes of enumeration. In order to aid in visual interpretation, potential transport debugging, and to simplify some transport implementations, the primary and vendor-defined datacodes for operations, responses, data formats, events, and properties in this standard have mutually exclusive values. The most significant four bits of a datacode (most significant nibble) shall have a particular bit pattern that identifies its code type. Therefore the allocation of these four bits to type specification infers that the minimum value of any enumerated datacode is 0 (`xxxx0000-00000000`) and the maximum value is 4,095 (`xxxx1111-11111111`).

It is strongly recommended that transport implementations use these codes directly in their binary representations, but this is not mandatory. Particular transport implementations may be unable to use the specified code systems for one or more code types, due to pre-existing structure formats for data-wrapping, or other constraints. Where it is possible to use the codes, they should be used. If one or more particular datacode types cannot be used, the transport implementation specification should still attempt to accommodate those datacode types that can be used. If the binary form suggested in this standard is not used for a

particular datacode type, an appropriate corresponding enumerated identifier in an alternate form should be made available where possible for each datatype enumeration specified here, each having the same usage and definition as those specified in this standard. This allows for transport-aggregating abstractions in host software to use the codes defined in this standard, even though a particular code might not be transmitted across the wire for a particular transport in the binary form specified. Transports may also need to perform multiple transactions over the wire in order to fulfill one operation defined in this standard, and therefore one operation code may not be sufficient.

For example, if a transport does not use the sixteen-bit OperationCodes, it should still provide an equivalent mechanism for the GetObject operation that supports the same usage defined in this standard. Another example would be a transport that uses OperationCodes for some operations but not others, because the transport in question possesses a built-in mechanism for performing the equivalent operation, and provides its own operation identification scheme for that operation.

Table 2: Datacode Formats

Bit 15	Bit 14	Bit 13	Bit 12	Bits 11-0	Code Type
0	0	0	0	Any	Undefined (not a conforming code)
0	0	0	1	Any	Standard OperationCode
0	0	1	0	Any	Standard ResponseCode
0	0	1	1	Any	Standard ObjectFormatCode
0	1	0	0	Any	Standard EventCode
0	1	0	1	Any	Standard DevicePropCode
0	1	1	0	Any	Reserved
0	1	1	1	Any	Reserved
1	0	0	0	Any	Undefined
1	0	0	1	Any	Vendor-Defined OperationCode
1	0	1	0	Any	Vendor-Defined ResponseCode
1	0	1	1	Any	Vendor-Defined ObjectFormatCode
1	1	0	0	Any	Vendor-Defined EventCode
1	1	0	1	Any	Vendor-Defined DevicePropCode
1	1	1	0	Any	Reserved
1	1	1	1	Any	Reserved

It is a convention of this standard that all datacodes shall set bit 15 to 1 in order to indicate that the code value is vendor-specific, and therefore undefined in this standard. Codes indicating that they are vendor-defined should be interpreted according to the VendorExtensionID and VendorExtensionVersion fields of the DeviceInfo dataset as described in Clause 5.5.1.

Individual datacode interpretations and usage are described in the appropriate section of this standard for each type of datacode.

5.3 Simple Types

The following table describes the generic datatypes that may be used in this standard:

Table 3: Datatype Codes

Datatype Code	Type	Description
0x0000	UNDEF	Undefined
0x0001	INT8	Signed 8 bit integer
0x0002	UINT8	Unsigned 8 bit integer
0x0003	INT16	Signed 16 bit integer
0x0004	UINT16	Unsigned 16 bit integer
0x0005	INT32	Signed 32 bit integer
0x0006	UINT32	Unsigned 32 bit integer
0x0007	INT64	Signed 64 bit integer
0x0008	UINT64	Unsigned 64 bit integer
0x0009	INT128	Signed 128 bit integer
0x000A	UINT128	Unsigned 128 bit integer
0x4001	AINT8	Array of Signed 8 bit integers
0x4002	AUINT8	Array of Unsigned 8 bit integers
0x4003	AINT16	Array of Signed 16 bit integers
0x4004	AUINT16	Array of Unsigned 16 bit integers
0x4005	AINT32	Array of Signed 32 bit integers
0x4006	AUINT32	Array of Unsigned 32 bit integers
0x4007	AINT64	Array of Signed 64 bit integers
0x4008	AUINT64	Array of Unsigned 64 bit integers
0x4009	AINT128	Array of Signed 128 bit integers
0x400A	AUINT128	Array of Unsigned 128 bit integers
0xFFFF	STR	Variable-length Unicode String
All other values	Undefined	Reserved

All datatypes having bit 14 set to 1 are uniform arrays of individual fixed-length types.

5.3.1 Integers

The most common data type that is required in this standard is integer. Integers may be signed or unsigned, and may be placed into arrays.

5.3.2 Handles

Handles are 32-bit device-unique unsigned integers (UINT32) that are exposed externally in order to allow consistent referencing to its logical and/or physical elements by other conforming devices. All handles act as UINT32 datatypes with the added constraint that they

have individually unique values relative to the currently open session. Handles shall be persistent over at least a particular session, and may or may not be persistent over an entire power cycle, or even across power cycles. There is no significance to the value of a handle other than it is unique, and therefore they do not need to be consecutively assigned. The values 0x00000000 and 0xFFFFFFFF may not refer to any valid objects, and are reserved for context-specific meanings, such as "no handle," "default handle," or "all handles." Any use of context-specific meanings shall be described in the appropriate clauses.

Handles are used to refer to image and non-image objects that are present on a device as well as objects that are expected to immediately become present, and in this context are referred to as ObjectHandles. ObjectHandles may refer to image or non-image objects that are capable of being retrieved as the response to an operation. The existence of an ObjectHandle infers the ability to produce an ObjectInfo dataset for the object that is referred to, as well as production of the object itself for object types that are not fully qualified by the ObjectInfo dataset. An association (e.g. a generic folder) is an example of an object type that is fully qualified by an ObjectInfo dataset, and therefore possesses no actual object to send. The type of data object that the ObjectHandle refers to may be determined by examining the ObjectFormatCode described in the data object's ObjectInfo dataset described in Clause 5.5.2.

5.3.3 Decimal Types

Fixed-point datatypes may be represented using integers by contextually specifying a scalar factor. This protocol does not currently require conventions for handling floating-point data. Transmitted data of this type is typically contained inside of a data object that is opaque to this protocol.

5.3.4 Strings

All strings shall consist of standard 2-byte Unicode characters as described in ISO10646, with the added constraint of having a maximum number of characters of 255. Strings shall be represented by a combination of two fields as described in the following table. An empty string shall consist of one byte with value of 0x00.

Table 4: String Format

Dataset Field	Size (bytes)	Data Type
NumChars	1	UINT8
StringChars	Variable	Unicode null-terminated string

NumChars: Represents the number of characters in the string, including the terminating null character for non-empty strings. Allowed values are 0 to 255.

StringChars: This field holds the actual Unicode null-terminated string with 2 byte characters. This field shall not contain more than 255 characters, including the terminating null characters. This field is not present for empty strings.

5.3.4.1 DateTime String

When needed, the date and time shall be expressed using a compatible subset of ISO 8601 so that it can be easily parseable. This shall take the form of a Unicode string in the format "YYYYMMDDThhmmss.s" where YYYY is the year, MM is the month 01-12, DD is the day of the month 01-31, T is a constant character, hh is the hours since midnight 00-23, mm is the minutes 00-59 past the hour, and ss.s is the seconds past the minute, with the ".s" being optional tenths of a second past the second. This string can optionally be appended with Z to indicate UTC, or +/-hhmm to indicate the time is relative to a time zone. Appending neither indicates the time zone is unspecified.

5.4 Arrays

An array is a concatenation of one or more elements of the same fixed-length type. All array datasets contain a field representing the total number of elements contained in the array. The datatype, element size, and interpretation of the individual elements shall be context-specific, and therefore not explicit within the array structure. An empty array would consist of only a single UINT32 value of 0x00000000. For internal array referencing purposes, arrays shall be treated as zero-based.

Table 5: Array Format

Field	Size (bytes)	Format
NumElements	4	UINT32
ArrayEntry[0]	*Element Size*	Special
ArrayEntry[1]	*Element Size*	Special
…	*Element Size*	…
ArrayEntry[*NumElements-1*]	*Element Size*	Special

NumElements: Represents the total number of elements contained in the array, and therefore is equal to the number of ArrayEntry fields. This value may be 0x00000000, which indicates an empty array that would only be 4 total bytes in size.

ArrayEntry[*n*]: The total number of these fields shall equal the value held by the NumElements field. The size and interpretation (i.e. datatype) of each ArrayEntry is

context-dependent, but should be the same for all elements in an array. These fields are not present for empty arrays.

5.5 Datasets

For the purposes of this standard, the term dataset is defined as a transport-independent collection of one or more individual data items with defined interpretations. A dataset serves as a set of requirements for the data that must accounted for by an opposing device as the result of a particular operation or event. Individual data items may hold other embedded data items. A transport implementation may or may not need to place its own wrappers around the dataset, split datasets in multiple pieces, pack the fields a particular way, or pass different fields using different mechanisms.

In order to avoid inconsistencies, transport implementations should reference the field ordering in this specification when defining their implementations of the datasets. If the transport is wrapping the dataset completely, it should retain the field ordering defined in this specification unless a distinct benefit can be obtained by not doing so. The particular byte packing mechanism for individual fields must be clearly defined, such as little-endian or big-endian.

All unused fields in all datasets shall initially be set to their default value. If a field does not have an explicit default value, the implicit default value is zero for integer datatypes, and the empty string for strings. Any fields in a dataset that are unused are considered to have no default value, and should be set to zero.

If datasets are changed in future versions of this standard, fields will only be added, never deleted. No existing fields will be re-defined in future versions of this standard, and the specified field ordering shall remain constant. Unless otherwise indicated, reserved fields in datasets are required to be transmitted by the transport implementation, but should be set to 0x00000000 for integers and to an empty string for strings.

5.5.1 DeviceInfo Dataset

This dataset is used to hold the description information for a device. The Initiator can obtain this dataset from the Responder without opening a session with the device. This dataset holds data that describes the device and its capabilities. This information is only static if the device capabilities cannot change during a session, which would be indicated by a change in the FunctionalMode value in the dataset. For example, if the device goes into a sleep mode in which it can still respond to GetDeviceInfo requests, the data in this dataset should reflect the capabilities of the device while it is in that mode only (including any operations and properties needed to change the FunctionalMode, if this is allowed remotely). If the power state or the capabilities of the device changes (due to a FunctionalMode change), a DeviceInfoChanged event shall be issued to all sessions in order to indicate how its capabilities have changed.

Table 6: DeviceInfo Dataset

Dataset Field	Field Order	Size (bytes)	Data Type
StandardVersion	1	2	UINT16
VendorExtensionID	2	4	UINT32
VendorExtensionVersion	3	2	UINT16
VendorExtensionDesc	4	Variable	String
FunctionalMode	5	2	UINT16
OperationsSupported	6	Variable	OperationCode Array
EventsSupported	7	Variable	EventCode Array
DevicePropertiesSupported	8	Variable	DevicePropCode Array
CaptureFormats	9	Variable	ObjectFormatCode Array
ImageFormats	10	Variable	ObjectFormatCode Array
Manufacturer	11	Variable	String
Model	12	Variable	String
DeviceVersion	13	Variable	String
SerialNumber	14	Variable	String

All optional strings that are not provided should consist of an empty string. Clause 5.3.3 describes the use of strings.

Standard Version: Highest version of the standard that the device can support. This represents the standard version expressed in hundredths (e.g. 1.32 would be stored as 132).

VendorExtensionID: Provides the context for interpretation of any vendor extensions used by this device. If no extensions are supported, this field shall be set to 0x00000000. If vendor-specific codes of any type are used, this field is mandatory, and should not be set to 0x00000000. These IDs are assigned by PIMA, as described in Clause 9.5.

VendorExtensionVersion: The vendor-specific version number of extensions that are supported. This shall be expressed in hundredths (e.g. 1.32 would be stored as 132).

VendorExtensionDesc: An optional string used to hold a human-readable description of the VendorExtensionID. This field should only be used for informational purposes, and not as the context for the interpretation of vendor-extensions.

FunctionalMode: An optional field used to hold the functional mode. This field controls whether the device is in an alternate mode that provides a different set of capabilities (i.e. supported operations, events, etc.) If the device only supports one mode, this value should always be zero. The following table describes the standard functional modes:

Table 7: FunctionalMode Values

Value	Description
0x0000	Standard Mode
0x0001	Sleep State
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

The functional mode information is held by the device as a device property. This property is described in Clause 13.4.2. In order to change the functional mode of the device remotely, a session needs to be opened with the device, and the SetDeviceProp operation needs to be used.

OperationsSupported: This field is an array of OperationCodes representing operations that the device is currently supporting, given the FunctionalMode indicated. Clause 10.2 describes these codes.

EventsSupported: This field is an array of EventCodes representing the events that are currently generated by the device in appropriate situations, given the FunctionalMode indicated. Clause 12.5 describes these codes.

DevicePropertiesSupported: This field is an array of DevicePropCodes representing DeviceProperties that are currently exposed for reading and/or modification, given the FunctionalMode indicated. Clause 13.3.5 describes these codes.

CaptureFormats: The list of data formats in ObjectFormatCode form that the device can create using an InitiateCapture operation and/or an InitiateOpenCapture operation, given the FunctionalMode indicated. These are typically image object formats, but can include any object format that can be fully captured using a single trigger mechanism, or an initiate/terminate mechanism. All image object formats that a device can capture data in shall be listed prior to any non-image object formats, and shall be in preferential order such that the default capture format is first. ObjectFormats are described in Clause 6, **Image and Data Object Formats**.

ImageFormats: The list of image formats in ObjectFormatCode form that the device supports in order of highest preference to lowest preference. Support for an image format refers to the ability to interpret image file contents according to that format's specifications, for display and/or manipulation purposes. For image output devices, this field represents the image formats that the output device is capable of outputting. This field does not describe any device format-translation capabilities. Refer to Clause 6 for more information on image format support.

Manufacturer: An optional human-readable string used to hold the Responder's manufacturer.

Model: An optional human-readable string used to communicate the Responder's model name.

DeviceVersion: An optional string used to communicate the Responder's firmware or software version in a vendor-specific way

SerialNumber: An optional string used to communicate the Responder's serial number, which is defined as a unique value among all devices sharing identical Model and Device Version fields. If unique serial numbers are not supported, this field shall be set to the empty string. The presence of a non-null string in the SerialNumber field for one device infers that this field is non-zero and unique among all devices of that model and version.

5.5.2 ObjectInfo Dataset

This dataset is used to define the information about data objects in persistent store, as well as optional information if the data is known to be an image or an association object. It is required that these data items be accounted for in response to a GetObjectInfo operation. If the data is not known to be an image, or the image information is unavailable, the image-specific fields shall be set to zero. Objects of type Association are fully qualified by the ObjectInfo dataset.

Table 8: ObjectInfo Dataset

Dataset Field	Field Order	Size (bytes)	Data Type	Image Specific	Association Specific
StorageID	1	4	StorageID	No	No
ObjectFormat	2	2	ObjectFormatCode	No	No
ProtectionStatus	3	2	UINT16	No	No
ObjectCompressedSize	4	4	UINT32	No	No
ThumbFormat	5	2	ObjectFormatCode	Yes	No
ThumbCompressedSize	6	4	UINT32	Yes	No
ThumbPixWidth	7	4	UINT32	Yes	No
ThumbPixHeight	8	4	UINT32	Yes	No
ImagePixWidth	9	4	UINT32	Yes	No
ImagePixHeight	10	4	UINT32	Yes	No
ImageBitDepth	11	4	UINT32	Yes	No
ParentObject	12	4	ObjectHandle	No	No
AssociationType	13	2	AssociationCode	No	Yes
AssociationDesc	14	4	AssociationDesc	No	Yes
SequenceNumber	15	4	UINT32	No	No
Filename	16	Variable	String	No	No
CaptureDate	17	Variable	DateTime String	No	No
ModificationDate	18	Variable	DateTime String	No	No
Keywords	19	Variable	String	No	No

StorageID: The StorageID of the device's store in which the image resides. See Clause 8.1 for a description of StorageIDs.

ObjectFormat: Indicates ObjectFormatCode of the object. See Clause 6.2 for a list of these codes.

ObjectCompressedSize: The size of the buffer needed to hold the entire binary object in bytes. This field may be used for memory allocation purposes in object receivers by transport implementations.

ProtectionStatus: An optional field representing the write-protection status of the data object. Objects that are protected may not be deleted as the result of any operations specified in this standard without first separately removing their protection status in a separate transaction. The values are enumerated according to the following table:

Table 9: ObjectInfo ProtectionStatus Values

Value	Description
0x0000	No Protection
0x0001	Read-Only
All other values	Reserved

All values not explicitly defined are reserved for future use. This protection field is distinctly different in scope than the AccessCapability field present in the StorageInfo dataset, described in Clause 5.5.3. If an attempt to delete an object is made, success will only occur if the ProtectionStatus of the object is 0x0000 and the AccessCapability of the store allows deletion. If a device does not support object protection, this field should always be set to 0x0000, and the SetProtection operation should not be supported. Refer to Clause 5.5.3 for a description of the StorageInfo dataset.

ThumbFormat: Indicates ObjectFormat of the thumbnail. In order for an object to be referred to as an image, it must be able to produce a thumbnail as the response to a request. Therefore, this value should only be 0x00000000 for the case of non-image objects. Refer to Clause 6.2 for a list of ObjectFormatCodes.

ThumbCompressedSize: The size of the buffer needed to hold the thumbnail. This field may be used for memory allocation purposes. In order for an object to be referred to as an image, it must be able to produce a thumbnail as the response to a request. Therefore, this value should only be 0x00000000 for the case of non-image objects.

ThumbPixWidth: An optional field representing the width of the thumbnail in pixels. If this field is not supported or the object is not an image, the value 0x00000000 shall be used.

ThumbPixHeight: An optional field representing the height of the thumbnail in pixels. If this field is not supported or the object is not an image, the value 0x00000000 shall be used.

ImgPixWidth: An optional field representing the width of the image in pixels. If the data is not known to be an image, this field should be set to 0x00000000. The purpose of this field is to enable an application to provide the width information to a user prior to transferring the image. If this field is not supported, the value 0x00000000 shall be used.

ImgPixHeight: An optional field representing the height of the image in pixels. If the data is not known to be an image, this field should be set to 0x00000000. The purpose of this field is to enable an application to provide the height information to a user prior to transferring the image. If this field is not supported, the value 0x00000000 shall be used.

ImgBitDepth: An optional field representing the total number of bits per pixel of the uncompressed image. If the data is not known to be an image, this field should be set to 0x00000000. The purpose of this field is to enable an application to provide the bit depth information to a user prior to transferring the image. This field does not attempt to specify the number of bits assigned to particular color channels, but instead represents the total number of bits used to describe one pixel. If this field is not supported, the value 0x00000000 shall be used. This field should not be used for memory allocation purposes, but is strictly information that is typically inside of an image object, that may affect whether or not a user wishes to transfer the image, and therefore is exposed prior to object transfer in the ObjectInfo dataset.

ParentObject: Indicates the handle of the object that is the parent of this object. The ParentObject must be of object type Association. If the device does not support associations, or the object is in the "root" of the hierarchical store, then this value should be set to 0x00000000.

AssociationType: A field that is only used for objects of type Association. This code indicates the type of association. Refer to Clause 6.4 for a description of associations and a list of defined types. If the object is not an association, this field should be set to 0x0000.

AssociationDesc: This field is used to hold a descriptor parameter for the association, and may therefore only be non-zero if the AssociationType is non-zero. The interpretation of this field is dependent upon the particular AssociationType, and is only used for certain types of associations. If unused, this field should be set to 0x00000000. Refer to Clause 6.4 for information on this descriptor.

SequenceNumber: This field is optional, and is only used if the object is a member of an association, and only if the association is ordered. If the object is not a member of an ordered association, this value should be set to 0x00000000. These numbers should be created consecutively. However, to be a valid sequence, they do not need to be consecutive, but only monotonically increasing. Therefore, if a data object in the sequence is deleted, the SequenceNumbers of the other objects in the ordered association do not need to be renumbered, and examination of the sequential numbers will indicate a possibly deleted object by the missing sequence number.

Filename: An optional string representing filename information. This field should not include any filesystem path information, but only the name of the file or directory itself. The interpretation of this string is dependent upon the FilenameFormat field in the StorageInfo dataset that describes the logical storage area in which this object is stored. See Clause 5.5.3 for information on this field.

CaptureDate: A static optional field representing the time that the data object was initially captured. This is not necessarily the same as any date held in the ModificationDate field. This dataset uses the DateTime string described in Clause 5.3.4.1.

ModificationDate: An optional field representing the time of last modification of the data object. This is not necessarily the same as the CaptureDate field. This dataset uses the DateTime string described in Clause 5.3.4.1.

Keywords: An optional string representing keywords associated with the image. Each keyword shall be separated by a space. A keyword that consists of more than one word shall use underscore (_) characters to separate individual words within one keyword.

5.5.3 StorageInfo Dataset

This dataset is used to hold the state information for a storage device.

Table 10: StorageInfo Dataset

Dataset Field	Field Order	Length (bytes)	Data Type
StorageType	1	2	UINT16
FilesystemType	2	2	UINT16
AccessCapability	3	2	UINT16
MaxCapacity	4	8	UINT64
FreeSpaceInBytes	5	8	UINT64
FreeSpaceInImages	6	4	UINT32
StorageDescription	7	Variable	String
VolumeLabel	8	Variable	String

StorageType: The code that identifies the type of storage, particularly whether the store is inherently random-access or read-only memory, and whether it is fixed or removable media.

Table 11: Storage Types

Code Value	Storage Type
0x0000	Undefined
0x0001	Fixed ROM
0x0002	Removable ROM
0x0003	Fixed RAM
0x0004	Removable RAM
All other values	Reserved

All undefined values are reserved for future use.

FilesystemType: This optional code indicates the type of filesystem present on the device. This field may be used to determine the filenaming convention used by the storage device, as well as to determine whether support for a hierarchical system is present. If the storage device is DCF-conformant, it shall indicate so here.

Table 12: FilesystemType Values

Value	Description
0x0000	Undefined
0x0001	Generic Flat
0x0002	Generic Hierarchical
0x0003	DCF
All other values with Bit 15 set to 0	Reserved
All values with Bit 15 set to 1	Vendor-Defined

All values having bit 31 set to zero are reserved for future use. If a proprietary implementation wishes to extend the interpretation of this field, bit 31 should be set to 1.

AccessCapability: This field indicates whether the store is read-write or read-only. If the store is read-only, deletion may or may not be allowed. The allowed values are described in the following table. Read-Write is only valid if the StorageType is non-ROM, as described in the StorageType field above.

Table 13: StorageInfo AccessCapability Values

Value	Description
0x0000	Read-Write
0x0001	Read-Only without Object Deletion
0x0002	Read-Only with Object Deletion
All other values	Reserved

All values having bit 15 set to zero are reserved for future use. If a proprietary implementation wishes to extend the interpretation of this field, bit 15 should be set to 1.

MaxCapacity: This is an optional field that indicates the total storage capacity of the store in bytes. If this field is unused, it should report 0xFFFFFFFF.

FreeSpaceInBytes: The amount of free space that is available in the store in bytes. If this value is not useful for the device, it may set this field to 0xFFFFFFFF and rely upon the FreeSpaceInImages field instead.

FreeSpaceInImages: The number of images that may still be captured into this store according to the current image capture settings of the device. If the device does not implement this capability, this field should be set to 0xFFFFFFFF. This field may be

used for devices that do not report FreeSpaceInBytes, or the two fields may be used in combination.

StorageDescription: An optional field that may be used for a human-readable text description of the storage device. This should be used for storage-type specific information as opposed to volume-specific information. Examples would be "Type I Compact Flash," or "3.5-inch 1.44 MB Floppy". If unused, this field should be set to the empty string.

VolumeLabel: An optional field that may be used to hold the volume label of the storage device, if such a label exists and is known. If unused, this field should be set to the empty string.

6 Image and Data Object Formats

A data object is defined to be an image or other type of data that exists in persistent storage of a DSPD or other device. This standard specifies how image and data objects are transferred between a digital still photography device (DSPD) and other devices, but it does not specify how they are stored within such devices. A DSPD conforming to this standard shall transfer images to other devices using a particular data or datafile format for storage. The DSPD shall identify the data format used for the main (e.g. full size) image transferred, and shall provide a thumbnail (e.g. reduced size) image using one of the two thumbnail image file formats defined in this clause. The image file formats used to transfer the main image shall be either:

- EXIF/JPEG version 2.1, the preferred format
- One of the allowed image file formats listed in table 21
- A proprietary image file format

All devices conforming to this standard that receive images shall be capable of receiving (but not necessarily displaying) image files using any of these file formats. In addition, if the receiving device displays images, it shall be capable of decoding and displaying both of the thumbnail image file formats defined in this clause.

For all image and data objects, an ObjectFormatCode is provided in the ObjectInfo dataset to specify the format, as described in Clause 5.5.2. Setting bit 31 of this code can be used in conjunction with the VendorExtensionID in the DeviceInfo dataset to create a vendor-extended ObjectFormatCode to handle transfer of these non-standard file formats.

6.1 Thumbnail Formats

- The thumbnail images provided by a DSPD shall be either compressed JPEG images or uncompressed TIFF images. Compressed JPEG images are preferred.
- The device may optionally be capable of producing a thumbnail for non-still-image object formats, such as video clips, audio formats, etc. The method that a device should use to generate a thumbnail from these object types is not specified. This capability would be apparent if the device correctly responded to a GetThumb operation request for a data object with a DataFormat that is a non-still-image type.

6.1.1 Compressed JPEG thumbnail image files

The format of compressed thumbnail files shall be a JPEG interchange file using the baseline process specified in table 1 of ISO/IEC 10918-1. This requires that the image file use:

- DCT-based image compression process
- 8-bit samples within each component

- Baseline sequential (non-progressive) process
- Huffman coding with 2 AC and 2 DC tables

The baseline image file format is further restricted in that it shall have:

- Square pixel sampling
- A single interleaved scan (e.g. Y, Cr, Cb interleaved blocks)
- sRGB color space as defined in IEC 61966-2-1
- 3 color components: Y, Cr, Cb derived from the sRGB signals as follows, as specified in ITU-R BT.601:

  Y = 0.299 R + 0.587 G + 0.114 B

  Cb = (-0.299R - 0.587 G + 0.886B) * 0.564 + offset

  Cr = (-0.701R - 0.587 G + 0.114B) * 0.713 + offset

  or a single (luminance) component
- Either 4:2:2 or 4:2:0 Y:Cb:Cr spatially centered color subsampling or Y:Cb:Cr co-sited subsampling. Writers may use either option. Readers shall support both options.

Note that these features are all requirements of the EXIF version 2.1 specification.

In addition, it is preferred that the thumbnails meet the thumbnail image data format requirements specified in section 3.3.6 of "Design rule for Camera File system" (DCF), version 1.0. These requirements include:

- JPEG compression using 4:2:2 chrominance sampling and the "typical" Huffman table provided in ISO/IEC 10918-1
- 160 x 120 pixel image record with a 4:3 aspect ratio

6.1.2 Uncompressed TIFF thumbnail image files

The format of the uncompressed thumbnail images shall meet the requirements for the TIFF/EP format described in ISO 12234-2 *Photography - Electronic still picture cameras - Removable memory, Part 2: Image data format - TIFF/EP*. The thumbnail image shall have a Compression tag value = 1 (no compression) with the thumbnail image contained within IFD0.

6.2 ObjectFormatCodes

A list of object formats is given in the following table. An ObjectFormatCode is a 16-bit unsigned integer (UNIT16) that represents the format of an image or data object that resides on a DSPD. Setting bit 15 to 1 in the ObjectFormatCode indicates a vendor-defined format. All ObjectFormatCodes that represent image formats shall also have bit 11 set to 1, including those that are vendor-defined.

Table 14: ObjectFormatCodes

Object FormatCode	Type	Format	Description
0x3000	A	Undefined	Undefined non-image object
0x3001	A	Association	Association (e.g. folder)
0x3002	A	Script	Device-model-specific script
0x3003	A	Executable	Device-model-specific binary executable
0x3004	A	Text	Text file
0x3005	A	HTML	HyperText Markup Language file (text)
0x3006	A	DPOF	Digital Print Order Format file (text)
0x3007	A	AIFF	Audio clip
0x3008	A	WAV	Audio clip
0x3009	A	MP3	Audio clip
0x300A	A	AVI	Video clip
0x300B	A	MPEG	Video clip
0x300C	A	ASF	Microsoft Advanced Streaming Format (video)
0x3800	I	Undefined	Unknown image object
0x3801	I	EXIF/JPEG	Exchangeable File Format, JEIDA standard
0x3802	I	TIFF/EP	Tag Image File Format for Electronic Photography
0x3803	I	FlashPix	Structured Storage Image Format
0x3804	I	BMP	Microsoft Windows Bitmap file
0x3805	I	CIFF	Canon Camera Image File Format
0x3806	I	Undefined	Reserved
0x3807	I	GIF	Graphics Interchange Format
0x3808	I	JFIF	JPEG File Interchange Format
0x3809	I	PCD	PhotoCD Image Pac
0x380A	I	PICT	Quickdraw Image Format
0x380B	I	PNG	Portable Network Graphics
0x380C	I	Undefined	Reserved
0x380D	I	TIFF	Tag Image File Format
0x380E	I	TIFF/IT	Tag Image File Format for Information Technology (graphic arts)
0x380F	I	JP2	JPEG2000 Baseline File Format
0x3810	I	JPX	JPEG2000 Extended File Format
All other codes with MSN of 0011	Any	Undefined	Reserved for future use
All other codes with MSN of 1011	Any	Vendor-Defined	Vendor-Defined

Type: Image File Format (I) Ancillary Data File Format (A)

6.3 Object Format Version Identification

The version of the object format should be contained in the data object in a context-specific manner. Support for a particular ObjectFormatCode implies that the device possesses the capability to read the internal structure of the data object in a way that is particular to its format, and this includes the discovery and interpretation of any version information that may exist. The format for specifying the version information is dependent upon the versioning methodology used by the particular format. Formats that change over time should remain backward compatible with older formats. However, if backward compatibility is not retained for a certain format, the specific format for that version and above should be considered as a separate format, and therefore a new ObjectFormatCode should be assigned to the new versions of that format.

6.4 Data Object Association

This standard provides an optional method for associating related image and data objects, and this mechanism is what is used for representing folders and filesystems. Associations are represented using objects that are of type Association. All of the objects that are part of an association must be in the branch of the object tree underneath the corresponding association, in a folder-like manner. The association to which an object belongs may be determined by examining the ParentObject fields of each object's ObjectInfo dataset. There are different types of associations. The type is specified in the AssociationType field of the ObjectInfo dataset of the Association object.

6.4.1 Association Types

The following table describes the various associations that may be used:

Table 15: Association Types

AssociationCode	AssociationType	AssociationDesc Interpretation
0x0000	Undefined	Undefined
0x0001	GenericFolder	Unused
0x0002	Album	Reserved
0x0003	TimeSequence	DefaultPlaybackDelta
0x0004	HorizontalPanoramic	Unused
0x0005	VerticalPanoramic	Unused
0x0006	2DPanoramic	ImagesPerRow
0x0007	AncillaryData	Undefined
All other values with bit 15 set to 0	Reserved	Undefined
All values with bit 15 set to 1	Vendor-Defined	Vendor-Defined

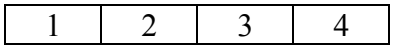
GenericFolder: This association type is used to represent a folder that may hold any type of object, and is analogous to the standard folder present in most filesystems. This association is typically used to represent a local grouping of objects, with no other relationship implied.

Album: This association type is the same as a folder but is used to hold image and data objects that have logical groupings according to content, capture sessions, or any other unspecified user-determined grouping. These are typically created by a user or automation technique. Some devices may wish to expose albums to the user but not all generic folders. Devices that do not distinguish between albums and folders should only use the AssociationType of GenericFolder. The AssociationDesc field is reserved for future definition by this standard, and for devices that support this version of the specification, shall be set to 0x00000000.

TimeSequence: Indicates that the data objects are part of a sequence of data captures that make up a set of time-ordered data of the same subject. This association is used to represent time-lapse or burst sequences. The order is interpreted to be sequential by capture time from first captured to last, and is indicated by the increasing values of the SequenceNumber fields in the ObjectInfo dataset for each object. If known, the AssociationDesc acts as a DefaultPlaybackDelta, and should be set to the desired time in milliseconds to delay between each object if exposing them sequentially in real-time. If unknown, this value should be set to 0x00000000.

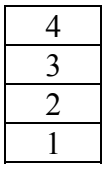
HorizontalPanoramic: Indicates that the associated data objects make up a panoramic series of images that are arranged side-by-side, in a horizontal fashion. The order of the sequence, from left to right when facing the subject, is indicated by the increasing values of the SequenceNumber fields in each object. The AssociationDesc is unused, and should be set to 0x00000000. For example, four images would have SequenceNumbers assigned as follows:

Figure 1: HorizontalPanoramic SequenceNumber Example



VerticalPanoramic: Indicates that the associated data objects make up a panoramic series of images that are arranged bottom-to-top, in a vertical fashion. The order of the sequence, from bottom to top when facing the subject, is indicated by the increasing values of the SequenceNumber fields in each object. The AssociationDesc is unused, and should be set to 0x00000000. For example, four images would have SequenceNumbers assigned as follows:

Figure 2: VerticalPanoramic SequenceNumber Example



2DPanoramic: Indicates that the associated data objects make up a two-dimensional panoramic series of images that are arranged left-to-right and bottom-to-top in adjacent or overlapping horizontal strips. The order of the sequence, from bottom-left to top-right when facing the subject, is indicated by the increasing values of the SequenceNumber fields in each object. The AssociationDesc is used to indicate the number of images in each row. For example, sixteen images arranged in a 4x4 2Dpanoramic would have SequenceNumbers assigned as follows:

Figure 3: 2DPanoramic SequenceNumber Example

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

AncillaryData: Indicates that the association represents one or more non-image objects being associated with an image object. For example, an image capture that also stores independent audio text files that are temporally related to the image capture may use this type of association to indicate the relationship. Optionally, if the individual objects are ordered (e.g. multiple temporally-related sound files), the SequenceNumber field of each object's ObjectInfo dataset should contain increasing integers. If the individual objects are unordered, the AssociationDesc should be set to 0x00000000. The AssociationDesc field is unused, and should be set to 0x00000000.

6.4.1.1 Unordered Associations

Unordered associations are the standard type of association used to relate data objects non-sequentially, such as filesystem folders, albums, etc. Objects that belong to unordered associations always possess ObjectInfo datasets with SequenceNumber fields equal to zero.

6.4.1.2 Ordered Associations

Ordered associations are an optional mechanism that may be used to represent associated objects, some or all of which are sequential in some context-specific fashion. Objects that are part of the ordered association may or may not have a non-zero SequenceNumber.

A. Objects of any type may be children of ordered associations, including other ordered or unordered associations.

B. Sequence numbers are used to order a proper or improper subset of child objects of a particular ordered association.

C. The value of a sequence number is only relevant between objects at the same level of the same hierarchical tree branch (i.e. siblings with the common ordered association parent.)

D. The value of a sequence number is only persistent while the child/parent relationship is intact. If an object that is part of an ordered association is assigned a new parent, the object's SequenceNumber should be invalidated (i.e. set to zero or to a new appropriate SequenceNumber if the new parent is an ordered association.)

E. Not all child objects of an ordered association have to have a sequence number. Objects without sequence numbers whose parent is an ordered association are considered related to the entire set of sequential objects, but are not a proper part of the sequence.

F. Non-zero sequence numbers among objects in an ordered association must be unique.

6.4.2 Associations as Filesystem Folders

Unordered associations may be used to represent file system layouts of data objects within a store. This allows representation of filesystems that are not dependent upon particular pathname conventions. Each object contains a ParentObject field in its ObjectInfo dataset. This ParentObject is the ObjectHandle of the Association object that "contains" this data object. This mechanism serves to expose a bottom-up singly-linked list that may be used to reconstruct the filesystem hierarchy, which may be stored on the opposing device optionally using a different data structure (e.g. double-linked list, top-down enumerator, etc.) Only objects of type Association may serve as ParentObjects. The bottom-up nature of the child/parent link information results in ParentObjects that are stateless with regard to which objects are their children, and infers that if a child is deleted, the parent and the ObjectInfo dataset describing it do not need to be updated. If the object exists in the "root" of the store, the value 0x00000000 shall be used for its ParentObject. Pathnames may be recreated using the Filename and ParentObject fields of each ObjectInfo dataset, inserting the separators of choice, in a platform-specific manner. Some receiving devices may not support exposing a hierarchical structure, in which case objects of type Association would be ignored. In other cases, devices may only wish to show associations of type Album as hierarchical. For an example of representing a filesystem using associations, refer to Appendix B.2.

7 Transport Requirements

This standard requires that underlying transports support certain features to enable the functionality described in this standard. This section describes these required features.

7.1 Disconnection Events

The transport shall be capable of notifying the overlying agents or system software that device has been disconnected from its previous environment. This environment may be either a wire interconnect or a wireless medium.

7.2 Reliable, Error Free Channel

The transport shall provide a reliable connection between devices comprising a session, assuming that the connection is not physically broken or terminated. This may be provided in a transport specific manner using error correction codes, retry scenarios, and fault recovery.

7.3 Asynchronous Event Support

Since a conforming device is required to notify other devices about any changes in its operational status, configuration, available objects or stores, the transport shall provide a mechanism that supports generation of outgoing events and servicing of incoming events asynchronously from operations, responses, or data transfers. In order to generate events, some transports may require initiation of an operation while others may require a notification mechanism.

7.4 Device Discovery and Enumeration

Transports shall possess the capability to discover and enumerate connected devices.

7.5 Specific Transports

The following transports are particularly relevant to digital still photography devices, and have independent organizations and specifications related to device implementations.

7.5.1 USB

Devices using this transport shall conform to the following specifications:

- *USB Specification*, Version 1.1, Sept. 23rd, 1998.

7.5.2 IrDA

Devices using this transport shall conform to the following specifications:

- *Infrared Data Association Serial Infrared Physical Layer Specification*, Version 1.1;
- *Infrared Data Association Serial Infrared Link Access Protocol Specification*, Version 1.1
- *Infrared Data Association Link Management Protocol Specification*, Version 1.1
- *Infrared Data Association Tiny Transport Protocol*, Version 1.1

7.5.3 IEEE1394

Devices using this transport IEEE1394 shall conform to the following specifications:

- IEEE Std 1394-1995, *Standard for High Performance Serial Bus*.

7.5.4 RS232C (Serial)

Typical serial transport layers will likely not meet the transport requirements for this standard without the addition of a support layer.

8 Persistent Storage

Digital still photography devices all share the capability for persistent local storage. It is the purpose of this section to describe how these storage devices are described and referenced.

The StorageInfo dataset is used to contain information on a store and is described in Clause 5.5.3.

8.1 StorageID

Every storage area on a device is represented by a unique, 4-byte, unsigned integer (UINT32) referred to as a StorageID. Each StorageID has two parts. The sixteen most significant bits represent a physical storage device, while the lower sixteen least significant bits represent a logical storage area within a physical store.

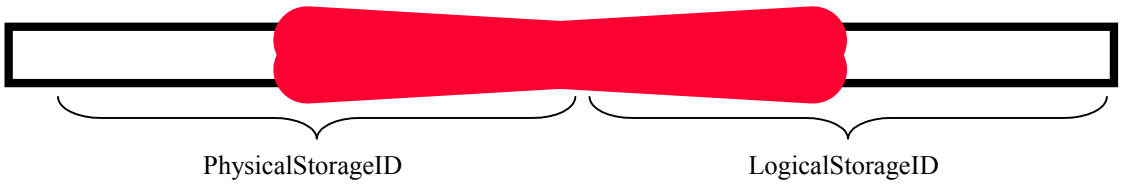


Figure 4: StorageID Layout

A physical store may possess zero or more logical stores. The StorageIDs for all logical stores within a physical store shall have the same PhysicalStorageID (i.e. the upper sixteen bits shall be identical). PhysicalStorageIDs are unique for a particular device at all times. In cases where there is more than one logical store within a physical store, all LogicalStorageIDs (i.e. the lower sixteen bits) within that physical store shall be unique. LogicalStorageIDs need only be unique within a physical store, and not globally across all stores. If a physical store does not contain any logical stores, the LogicalStorageID should be set to 0x0000, and it shall be assumed that the store does not contain any data, nor can it receive any data. Neither PhysicalStorageIDs nor LogicalStorageIDs need be consecutive, nor are they interpreted in any other way than as unique identifiers, in the same manner as handles.

StorageIDs should remain unique and persistent over a session. StorageIDs of 0x00000000 and 0xFFFFFFFF are not used to refer to real stores, but have context-specific meanings, such as “all-stores,” or “the default store.” These context-specific extended meanings are indicated in the sections where they are appropriate.

8.2 Data Object Referencing

8.2.1 Referencing via ObjectHandles

In order to ensure interoperability, images and data files shall be referenced using a 4-byte unsigned integer (heretofore referred to as an *ObjectHandle*). For information on this datatype, refer to Clause 5.3.2.

8.2.1.1 ObjectHandle Assignment

All conforming DSPDs shall allow data object referencing via the following behavior:

A. ObjectHandles shall be globally unique across all physical and logical storage areas on the device.

B. The values 0x00000000 and 0xFFFFFFFF shall be considered undefined or have context-specific meanings, and shall not be assigned to any valid object.

C. ObjectHandles shall be persistent for each data object present on the device for at least the particular session within which they are exposed, unless the object that the handle refers to becomes inaccessible (e.g. the data object is deleted or the store in which it resides is removed, in which case the device would be required to have issued the appropriate event indicating this change).

D. A particular ObjectHandle is only meaningful when used in the context of the device that assigned that handle. Moving the data object to another device does not imply that it will have the same handle on the new device that is possessed on the originating device, nor the originally assigned handle if the object is placed back into the originating device.

E. For each new data object that is added to the device while a session is open, the device shall:

1) Immediately assign a unique handle to the newly acquired data object that does not conflict with any other currently assigned handles.

2) Broadcast an appropriate ObjectAdded event to all open sessions according to Clause 12 of this standard. The Initiator is responsible for re-obtaining the StorageInfo dataset if necessary in order to get an updated version of the FreeSpaceInBytes or FreeSpaceInImages field.

F. If a data object is removed from the device during a session, the device shall:

1) Broadcast an appropriate ObjectRemoved event to all open sessions according to Clause 12 of this standard. If the data object is deleted as the result of an operation, the session within which the operation was issued should not receive the event. The Initiator is responsible for re-obtaining the StorageInfo dataset if necessary in order to get an updated version of the FreeSpaceInBytes or FreeSpaceInImages field.

8.2.2 AccessCapability

All stores have an AccessCapability field in their StorageInfo dataset that indicates whether they are read-only or not. If a store is read-only, objects cannot be sent to it, regardless of the individual protection status of individual data objects, as described in each object's ObjectInfo dataset. Optionally, a device may still allow objects to be deleted from the store even if the store is read-only with regard to new objects being placed there. For a description of allowed values, refer to Clause 5.5.3. For a description of individual data object protection, refer to the description of the ObjectInfo dataset in Clause 5.5.2.

8.3 Receiver Object Placement

The object receiver must possess the ability to determine where to place an incoming object that it has requested. In a GetObject scenario, the object receiver is the Initiator. In a SendObject scenario, the object receiver is the Responder, and therefore the Responder shall possess the ability to determine where to put an object that it is receiving if the destination is unspecified by the sender. The destination location chosen by the receiver may or may not be a function of the information that describes the object in its ObjectInfo dataset, such as ObjectFormat, size parameters, etc. In a SendObject scenario, the receiver informs the sender of the actual location it will be placed into upon operation completion using the response parameters from the SendObjectInfo operation. This is to allow for cases where the sender may have to deal with different kinds of receivers, some more capable than others. This allows receiver intervention from a GUI or receiver-side script to allow all incoming objects to be handled in some prescribed or functionally dynamic way, to queue images for printing by placing them in a spool folder, etc. The sender may attempt to request that the object be stored in a specific place on the receiver using the parameters in the SendObjectInfo operation, but the receiver may or may not allow this behavior. A particular receiver might not provide a structured filesystem, and may store all images in a "flat" area of memory. This type of device would either ignore all Association objects or store the associations using some technique other than a filenaming/foldering convention, such as a database.

In SendObject scenarios, a sending device may attempt one of two techniques for specifying object receiver location:

1. The Initiator does not specify receiver location for object being sent. The Responder chooses the location, and informs the Initiator where the object will be placed in the response to the SendObjectInfo operation.

2. The Initiator attempts to specify where the object should be placed on the Responder by using the operation parameters of SendObjectInfo. If the Responder cannot comply with placing the object there, the SendObjectInfo operation should fail with one of the following responses:

   A. The appropriate access response such as Invalid_StorageID, Invalid_ParentObject, Store_Full, Access_Denied, etc. In these cases, the Initiator may wish to try the SendObjectInfo operation again with an alternate destination.

   B. Specification_of_Destination_Unsupported. This is used to indicate that the Responder does not support Initiator-specified destination locations. The Initiator should not attempt to specify the destination again.

9 Communication Protocol

This section describes the protocol that shall be used by devices and transports in order to conform to this standard.

9.1 Device Roles

Rather than assume a host to device connection or a peer-to-peer connection, this standard will only refer to two devices, an *Initiator* and a *Responder*. The Initiator is the device that opens the session and initiates operation requests over a suitable transport according to that transport's implementation specification. A Responder is a device that responds to operation requests by sending requested data, responses, and events. Devices may have the capability to be only an Initiator, only a Responder, or both. A personal computer that detects and configures a USB camera is likely to only be capable of being an Initiator, while a USB-only camera may only be capable of being a Responder. An IR camera that opens an IrDA connection with a printer and pushes images to it might only be capable of being an Initiator, while that corresponding printer might only be capable of being a Responder. A digital camera that can send and receive images to and from other digital cameras must be capable of both roles. By assuming the Initiator role, a device is assuming added responsibility for device enumeration, transport aggregation if the device supports multiple PTP conforming transports, controlling the flow of the conversation, and negotiating the transport-specific attributes of the session, all in transport-specific ways. Initiators typically will have some form of graphical user interface that allows thumbnails to be displayed and selected by a user, as where devices that are only Responders will typically not.

9.2 Sessions

Sessions are defined as logical connections between two devices, over which ObjectHandles and StorageIDs are persistent.

Sessions are required when requesting handles from a device, as sessions define the minimum persistence period for handle assignments. Sessions do not need to be opened to obtain device capabilities via the GetDeviceInfo operation, but do need to be opened to transfer image and data objects, as well as their descriptors, such as the StorageInfo and ObjectInfo datasets. Any such datasets communicated during a session are considered valid for the duration of that session unless an explicit known event occurs indicating otherwise. All ObjectHandles and datasets other than the DeviceInfo dataset shall be considered invalid outside of the session in which they were provided.

A session is considered open as soon as the OpenSession operation sent by the Initiator completes with a valid response from the Responder. A session is closed when the CloseSession operation request is sent or the transport closes the communications channel, whichever occurs first.

9.2.1 SessionID

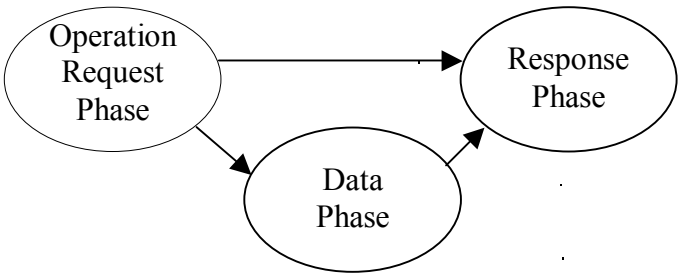
Each session shall have a SessionID that consists of one device-unique 32-bit unsigned integer (UINT32). SessionIDs are assigned by the Initiator as a parameter to the OpenSession operation, and must be non-zero.

9.3 Transactions

All transactions are considered atomic invocations whose origins can be traced to a single operation request being issued by an Initiator to a Responder. All transactions are considered synchronous and blocking within a session unless otherwise indicated. Devices that support multiple sessions must be able to keep each session asynchronous and opaque to each other. Asynchronous types of transactions, such as InitiateCapture, are handled by making the operation initiation synchronous. The response to these types of operations indicates only the success or failure of the operation initiation, and asynchronous events are used to handle the communication of new objects becoming available on the device at a later point in time. If an operation request is received which cannot be accommodated due to another previously invoked asynchronous operation still being executed, the device shall indicate that it is busy, using a Device_Busy response, and the Initiator is required to re-issue the operation at a later time.

Transactions in this protocol consist of three phases. Depending on the operation, the data phase may not be present. If the data phase is present, data may either be sent from the Initiator to the Responder (I -> R), or from the Responder to the Initiator (R -> I), but never in both directions for the same operation. The operation and response phases are always present. Only one transaction at a time can take place within a session. A transaction may be cancelled by an event. The following table describes the sequence of a transaction, with time increasing from left to right:

Figure 5: Transaction Sequence



9.3.1 TransactionID

Each transaction within a session shall have a unique transaction identifier called TransactionID that is a session-unique 32-bit unsigned integer (UINT32). TransactionIDs are continuous sequences in numerical order starting from 0x00000001. The TransactionID used for the OpenSession operation shall be 0x00000000. The first operation issued by an Initiator after an OpenSession operation shall possess a TransactionID of 0x00000001, the

second operation shall possess a TransactionID of 0x00000002, etc. The TransactionID of 0xFFFFFFFF shall not be considered valid, and is reserved for context-specific meanings. The presence of TransactionID allows asynchronous events to refer to specific previously initiated operations. If this field reaches its maximum value (0xFFFFFFFE), the device should "rollover" to 0x00000001. TransactionIDs allow events to refer to particular operation requests, allow correspondence between data objects and their describing datasets, and aid in debugging.

9.3.2 Operation Request Phase

The operation request phase consists of the transport-specific transmission of a 30-byte operation dataset from the Initiator to the Responder. The following table describes the fields that must be accounted for by the Responder in order for the OperationRequest phase to be considered complete:

Table 16: OperationRequest Dataset

Field	Size (Bytes)	Data Type
OperationCode	2	UINT16
SessionID	4	UINT32
TransactionID	4	UINT32
Parameter1	4	Any
Parameter2	4	Any
Parameter3	4	Any
Parameter4	4	Any
Parameter5	4	Any

OperationCode: The code indicating which operation is being initiated. For a list of these codes and their usages, refer to Clause 10.2.

SessionID: The identifier for the session within which this operation is being initiated. This value is assigned by the Initiator using the OpenSession operation. This field should be set to 0x00000000 for operations that do not occur within a session, and for the OpenSession OperationRequest dataset. Refer to Clause 9.2.1 for a description of SessionIDs.

TransactionID: The identifier of this particular transaction. This value shall be a value that is unique within a particular session, and shall increment by one for each subsequent transaction. Refer to Clause 9.3.1 for a description of transaction identifiers. This field should be set to 0x00000000 for the OpenSession operation.

Parameter *n*: This field holds the operation-specific n^{th} parameter. Operations may have at most five parameters. The interpretation of any parameter is dependent upon the OperationCode. Any unused parameter fields should be set to 0x00000000. If a

parameter holds a value that is less than 32 bits, the lowest significant bits shall be used to store the value, with the most significant bits being set to zeros.

9.3.3 Data Phase

The data phase is an optional phase that is used to transmit data that is larger than what can fit in the operation or response datasets. Typically, this is either a data object such as an image, an ancillary data file, or other datasets that are defined in this standard or are vendor extensions. All data that is not composed of small, fixed-length types must be sent via a data phase. For any particular operation, during the data phase data may flow from the Responder to the Initiator, from the Initiator to the Responder, or not at all. However, no operation shall send data in both directions during the data phase.

The format for data transferred during a data phase is context-specific by operation and ObjectFormat. If a particular transport wishes to wrap the data with some sort of header, or allow for division into multiple packets with wrappers for re-assembly and possibly event insertion, it is the responsibility of the transport-implementation to specify how the data should be handled, wrapped, and re-assembled. If the data phase is being used to transport a data object, it shall be considered opaque to the transport. If a dataset is being transmitted, it would likely be sent in some sort of transport-specific structure(s) or re-assembled from various transport-specific descriptors.

9.3.4 Response Phase

The response phase consists of the transport-specific transmission of a 30-byte response dataset from the Responder to the Initiator. The following table describes the fields that are required to be accounted for as part of the response phase:

Table 17: Response Dataset

Field	Size (Bytes)	Format
ResponseCode	2	UINT16
SessionID	4	UINT32
TransactionID	4	UINT32
Parameter1	4	Special
Parameter2	4	Special
Parameter3	4	Special
Parameter4	4	Special
Parameter5	4	Special

ResponseCode: Indicates the interpretation of the response as defined in the ResponseCode section in Clause 11.2.

SessionID: The identifier for the session within which this operation is being responded to. This value is assigned by the Initiator using the OpenSession operation, and should be

copied from the OperationRequest dataset that is received by the Responder prior to responding.

TransactionID: The identifier of the particular transaction. This field should be copied from the OperationRequest dataset that is received by the Responder prior to responding.

Parameter *n*: This field holds the operation-specific n^{th} response parameter. Response datasets may have at most five parameters. The interpretation of any parameter is dependent upon the OperationCode for which the response has been generated, and secondarily may be a function of the particular ResponseCode itself. Any unused parameter fields should be set to 0x00000000. If a parameter holds a value that is less than 32 bits, the lowest significant bits shall be used to store the value, with the most significant bits being set to zeros.

9.4 Operation Flow

The purpose of this section is to describe the flow of a conversation between two devices. The conversation varies depending on the scenario. The Initiator determines the flow of operations, while the Responder issues responses to the operations and may also issue events. Push mode consists of an Initiator sending one or more objects to the Responder. In Pull mode, the Initiator retrieves objects from the Responder. Pull mode usually first involves retrieving thumbnails so that a user can choose which images to retrieve. Ancillary data can also be transferred through either of these modes.

An Initiator should always be prepared to receive an event asynchronously relative to response or data phases, as event reception is not limited to transaction phase boundaries. This is of particular concern during the data phase, which may be lengthy if large image or data objects are being transferred. Similarly, the Responder should always be prepared to receive an event in place of an operation request, data, or a response for transports that use in-band events. Refer to Clause 12 for a description of **Events**.

9.4.1 Pull Scenarios

This clause describes some typical scenarios that are likely to occur between an Initiator and a Responder when the Initiator wants to get objects from the Responder.

9.4.1.1 Scenario 1

Initiator requests all image objects from the Responder, ignoring thumbnails, non-image objects and associations.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	GetObjectHandles	0xFFFFFFFF	0xFFFFFFFF	0x00000000	Send ObjectHandle Array
4	GetObjectInfo	ObjHandle 1	0x00000000	0x00000000	Send ObjectInfo 1 Dataset
5	Repeat Step 4 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send ObjectInfo *n* Dataset
6	GetObject	ObjHandle 1	0x00000000	0x00000000	Send Object 1 Data
7	Repeat Step 6 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send Object *n* Data
8	Close Session	0x00000000	0x00000000	0x00000000	None

9.4.1.2 Scenario 2

Initiator requests all objects from Responder, including non-image objects and associations.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	GetObjectHandles	0xFFFFFFFF	0x00000000	0x00000000	Send ObjectHandle Array
4	GetObjectInfo	ObjHandle 1	0x00000000	0x00000000	Send ObjectInfo 1 Dataset
5	Repeat Step 4 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send ObjectInfo *n* Dataset
6	GetObject	ObjHandle 1	0x00000000	0x00000000	Send Object 1 Data
7	Repeat Step 6 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send Object *n* Data
8	Close Session	0x00000000	0x00000000	0x00000000	None

9.4.1.3 Scenario 3

Initiator requests all thumbnails from the Responder, ignoring non-images and associations. Initiator then requests a subset of the image objects.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	GetObjectHandles	0xFFFFFFFF	0xFFFFFFFF	0x00000000	Send ObjectHandle Array
4	GetObjectInfo	ObjHandle 1	0x00000000	0x00000000	Send ObjectInfo 1 Dataset
5	Repeat Step 4 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send ObjectInfo *n* Dataset
6	GetThumb	ObjHandle 1	0x00000000	0x00000000	Send Thumb 1 Data
7	Repeat Step 6 for each ObjectHandle	ObjHandle *n*	0x00000000	0x00000000	Send Thumb *n* Data
8	GetObject	ObjHandle *a*	0x00000000	0x00000000	Send Object *a* Data
9	Repeat Step 8 for each ObjectHandle Selected	ObjHandle *m*	0x00000000	0x00000000	Send Object *m* Data
10	Close Session	0x00000000	0x00000000	0x00000000	None

9.4.1.4 Scenario 4

The Initiator examines the list of stores on the Responder. The Initiator chooses one store and discovers all objects in the root of the hierarchical file system of that store. The Initiator then retrieves all thumbnails for any image objects discovered there.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	GetStorageIDs	0x00000000	0x00000000	0x00000000	Send StorageID Array
4	GetStorageInfo	StorageID 1	0x00000000	0x00000000	Send StorageInfo 1
5	Repeat Step 4 for each ObjectHandle in array returned in step 4	StorageID *n*	0x00000000	0x00000000	Send StorageInfo *n*
6	GetObjectHandles	StorageID of selected store	0x00000000	0xFFFFFFFF	Send ObjectHandle Array for objects in root of selected store
7	GetObjectInfo	ObjHandle 1	0x00000000	0x00000000	Send ObjectInfo 1 Dataset
8	Repeat Step 7 for each ObjectHandle in array returned in step 6	ObjHandle *n*	0x00000000	0x00000000	Send ObjectInfo *n* Dataset
9	GetThumb	ObjHandle of first image object	0x00000000	0x00000000	Send Thumb 1 Data
10	Repeat Step 9 for each ObjectHandle in array returned in step 6 that represents an image	ObjHandle *n*	0x00000000	0x00000000	Send Thumb *n* Data
11	Close Session	0x00000000	0x00000000	0x00000000	None

9.4.2 Push Scenarios

This clause describes some typical scenarios that are likely to occur between an Initiator and a Responder, when the Initiator wants to send objects to the Responder.

9.4.2.1 Scenario 1

The Initiator pushes all objects to the Responder, allowing the Responder to determine where to place the objects it is receiving.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	SendObjectInfo	0x00000000	0x00000000	0x00000000	Allocate memory, assign new ObjHandle, return StorageID, Parent ObjHandle, ObjHandle
4	SendObject	0x00000000	0x00000000	0x00000000	Write data to store, invalidate SendObjInfo
5	Repeat Steps 3 and 4 for each Object to send in a top-down fashion				
6	Close Session	0x00000000	0x00000000	0x00000000	None

9.4.2.2 Scenario 2

The Initiator pushes all objects to the Responder, specifying the intended location on the Responder for each.

Step	Initiator Action	Parameter1	Parameter2	Parameter3	Responder Action
1	GetDeviceInfo	0x00000000	0x00000000	0x00000000	Send DeviceInfo dataset
2	OpenSession	SessionID	0x00000000	0x00000000	Create ObjectHandles, Storage IDs if necessary
3	SendObjectInfo	Responder Target StorageID	Responder Target Parent's ObjectHandle	0x00000000	Allocate memory, assign new ObjHandle, return StorageID, Parent ObjHandle, ObjHandle
4	SendObject	0x00000000	0x00000000	0x00000000	Write data to store, invalidate ObjInfo
5	Repeat Steps 3and 4 for each Object to send in a breadth-wise fashion				
6	Close Session	0x00000000	0x00000000	0x00000000	None

9.5 Vendor Extensions

Device manufacturers can extend this standard through several mechanisms outlined in this specification. The VendorExtensionID and VendorExtensionVersion fields of the DeviceInfo dataset can be used to uniquely identify the vendor extensions. For example, two vendors may happen to use the same vendor extended OperationCode 0x8001 for different uses, but the VendorExtensionID and/or VendorExtensionVersion fields for the two devices will be different, and therefore the intended interpretation and usage can be determined. All vendor-extended values shall be qualified with the VendorExtensionID for unique identification. For example, a device cannot use a vendor extended operation until it checks the VendorExtensionID and VendorExtensionVersion fields of sent by the other device in the DeviceInfo dataset. VendorExtensionIDs shall be assigned by PIMA (Photographic Imaging Manufacturer's Association). A list of assigned IDs may be found at http://www.pima.net. VendorExtensionVersion numbers shall be maintained internally by each vendor, and should be publicly advertised if vendors wish others to be able to access their extensions.

10 Operations

This section describes the various operations that are defined in these standard, along with their parameters and intended usages. The operations have been defined in order to ensure the following:

- The basic core functionality is accessible easily and uniformly for all devices.
- Advanced functionality is exposed in an optional, common way for more capable devices.
- Vendor extensibility is accessible via a well-defined mechanism.
- The transport layer and the controlling device are completely abstracted.

10.1 Operation Parameters

Many operations have parameters that must be accounted for by all implementations, even if particular values of particular parameters are not supported. All parameters are 32 bits in length. If a particular parameter for a particular operation needs less than 32 bits, the least significant bits shall be used, with the non-used most significant bits being set to zero.

For some operations, parameters act like device properties, as both can modify the way an operation is carried out. In the case of an inconsistency between operation parameters and device properties, the parameter shall take precedence over any device property, but will not cause the device property or the device behavior to change for any subsequent similar operation requests that do not specify the parameter similarly.

10.2 OperationCode Format

OperationCodes are transferred as part of the operation dataset described in Clause 9.3.3. All OperationCodes shall take the form of a 16-bit integer, are referred to using hexadecimal notation, and have bit 12 set to 1, and bits 13 and 14 set to 0. All non-defined ResponseCodes having bit 15 set to zero are reserved for future use. If a proprietary implementation wishes to define a proprietary OperationCode, bit 15 should be set to 1.

10.3 OperationCode Summary

The following table summarizes the operations defined by this standard and their corresponding OperationCodes. Refer to Clause 14, **Conformance Section**, for information on which operations are required and which are optional. Parameters that are in brackets are optional. For details on extending operations, refer to Clause 9.5, **Vendor Extensions**. The following table gives a summary of the operations and parameters defined in this standard:

Table 18: Operation Summary

Operation Code	Operation Name
0x1000	Undefined
0x1001	GetDeviceInfo
0x1002	OpenSession
0x1003	CloseSession
0x1004	GetStorageIDs
0x1005	GetStorageInfo
0x1006	GetNumObjects
0x1007	GetObjectHandles
0x1008	GetObjectInfo
0x1009	GetObject
0x100A	GetThumb
0x100B	DeleteObject
0x100C	SendObjectInfo
0x100D	SendObject
0x100E	InitiateCapture
0x100F	FormatStore
0x1010	ResetDevice
0x1011	SelfTest
0x1012	SetObjectProtection
0x1013	PowerDown
0x1014	GetDevicePropDesc
0x1015	GetDevicePropValue
0x1016	SetDevicePropValue
0x1017	ResetDevicePropValue
0x1018	TerminateOpenCapture
0x1019	MoveObject
0x101A	CopyObject
0x101B	GetPartialObject
0x101C	InitiateOpenCapture
All other codes with MSN of 0001	Reserved
All codes with MSN of 1001	Vendor-Extended Operation Code

10.4 Operation Descriptions

This Clause describes each individual operation, as well as the usage of the operation.

10.4.1 GetDeviceInfo

OperationCode: `0x1001`

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: DeviceInfo dataset

Data Direction: R -> I

ResponseCode Options: OK, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: This operation returns information and capabilities about the Responder device by returning a DeviceInfo dataset. This dataset is described in Clause 5.5.1. This operation is the only operation that may be issued inside or outside of a session. When used outside a session, both the SessionID and the TransactionID in the OperationRequest dataset shall be set to 0x00000000.

10.4.2 OpenSession

OperationCode: `0x1002`

Operation Parameter1: SessionID

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Parameter_Not_Supported, Invalid_Parameter, Session_Already_Open, Device_Busy

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Causes device to allocate resources, assigns handles to data objects if necessary, and performs any connection-specific initialization. The SessionID will then be used by all other operations during the session. Unless otherwise specified, an open session is required to invoke an operation. If the first parameter is 0x00000000, the operation should fail with a response of Invalid_Parameter. If a session is already open, and the device does not support multiple sessions, the response Session_Already_Open should be returned, with the SessionID of the already open session as the first response parameter. The response Session_Already_Open should also be used if the device supports multiple sessions, but a session with that ID is already open. If the device supports multiple sessions, and the maximum number of sessions are open, the device should respond with Device_Busy.

The SessionID and TransactionID fields of the operation dataset should both be set to 0x00000000 for this operation.

10.4.3 CloseSession

OperationCode: `0x1003`

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Session_Not_Open, Invalid_TransactionID, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Closes the session. Causes device to perform any session-specific cleanup.

10.4.4 GetStorageIDs

OperationCode: `0x1004`

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: StorageIDArray

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: This operation returns a list of the currently valid StorageIDs. This array shall contain one StorageID for each valid logical store. One StorageID should also be present for each removable media that is not inserted, which would contain a non-zero PhysicalStorageID and a LogicalStorageID with the value 0x0000.

10.4.5 GetStorageInfo

OperationCode: `0x1005`

Operation Parameter1: StorageID

Operation Parameter2: None

Operation Parameter3: None

Data: StorageInfo

Data Direction: R -> I

ResponseCode Options: OK, Session_Not_Open, Invalid_TransactionID, Access_Denied, Invalid_StorageID, Store_Not_Available, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Returns a StorageInfo dataset for the particular storage area indicated in the first parameter. This dataset is defined in Clause 5.5.3.

10.4.6 GetNumObjects

OperationCode: `0x1006`

Operation Parameter1: StorageID

Operation Parameter2: [ObjectFormatCode]

Operation Parameter3: [ObjectHandle of Association for which number of children is desired]

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_StorageID, Store_Not_Available, Specification_By_Format_Unsupported, Invalid_Code_Format, Parameter_Not_Supported, Invalid_ParentObject, Invalid_ObjectHandle, Invalid_Parameter

Response Parameter1: NumObjects

Response Parameter2: None

Response Parameter3: None

Description: Returns the total number of objects present in the store indicated by the first parameter. If the number of objects aggregated across all stores is desired, a StorageID of 0xFFFFFFFF may be used. If a single store is specified, and the store is unavailable because of media removal, this operation should return Store_Not_Available.

By default, this operation returns the total number of objects, which includes both image and non-image objects of all types.

The second parameter, ObjectFormatCode, is optional, and may not be supported. This parameter is used to identify a particular ObjectFormatCode, so that only objects of the particular type will be counted towards NumObjects. If the number of objects of all formats that are images is desired, the value 0xFFFFFFFF may be used. If this parameter is not used, it shall be set to 0x00000000. If the value is non-zero, and the Responder does not support specification by ObjectFormatCode, it should fail the operation by returning a ResponseCode with the value of Specification_By_Format_Unsupported.

The third parameter is optional, and may be used to request only the number of objects that belong directly to a particular association. If the third parameter is a valid ObjectHandle for an Association, this operation should only return the number of ObjectHandles that exist for objects that are direct children of the Association, and therefore only the number of ObjectHandles which refer to objects that possess an ObjectInfo dataset with the ParentObject field set to the value indicated in the third parameter. If the number of only those ObjectHandles corresponding to objects in the "root" of a store is desired, this parameter may be set to 0xFFFFFFFF. If the ObjectHandle referred to is not a valid ObjectHandle, the appropriate response is Invalid_ObjectHandle. If this parameter is specified, is a valid ObjectHandle, but the object referred to is not an association, the response Invalid_ParentObject should be returned. If unused, this operation returns the number of ObjectHandles aggregated across the entire device (modified by the second parameter), and the third parameter should be set to 0x00000000.

10.4.7 GetObjectHandles

OperationCode: 0x1007

Operation Parameter1: StorageID

Operation Parameter2: [ObjectFormatCode]

Operation Parameter3: [ObjectHandle of Association for which a list of children is desired]]

Data: ObjectHandleArray

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_StorageID, Store_Not_Available, Invalid_ObjectFormatCode, Specification_By_Format_Unsupported, Invalid_Code_Format, Invalid_ObjectHandle, Invalid_Parameter, Parameter_Not_Supported, Invalid_ParentObject, Invalid_ObjectHandle

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Returns an array of ObjectHandles present in the store indicated by the StorageID in the first parameter. If an aggregated list across all stores is desired, this value shall be set to 0xFFFFFFFF. Arrays are described in Clause 5.4.

The second parameter is optional, and may or may not be supported. This parameter allows the Initiator to ask for only the handles that represent data objects that possess a format specified by the ObjectFormatCode. If a list of handles that represent only image objects is desired, this second parameter may be set to 0xFFFFFFFF. If it is not used, it shall be set to 0x00000000. If the value is non-zero, and the Responder does not support specification by ObjectFormatCode, it should fail the operation by returning a ResponseCode with the value of Specification_By_Format_Unsupported. If a single store is specified, and the store is unavailable because of media removal, this operation should return Store_Not_Available.

The third parameter is optional, and may be used to request only a list of the handles of objects that belong to a particular association. If the third parameter is a valid ObjectHandle for an Association, this operation should return only a list of ObjectHandles of objects that are direct children of the Association, and therefore only ObjectHandles who refer to objects that possess an ObjectInfo dataset with the ParentObject field set to the value indicated in the third parameter. If a list of only those ObjectHandles corresponding to objects in the "root" of a store is desired, this parameter may be set to 0xFFFFFFFF. If the ObjectHandle referred to is not a valid ObjectHandle, the appropriate response is Invalid_ObjectHandle. If this parameter is specified, is a valid ObjectHandle, but the object referred to is not an association, the response Invalid_ParentObject should be returned. If the third parameter is unused, this operation

returns ObjectHandles aggregated across the entire device (modified by the second parameter), and the third parameter should be set to 0x00000000.

10.4.8 GetObjectInfo

OperationCode: `0x1008`

Operation Parameter1: ObjectHandle

Operation Parameter2: None

Operation Parameter3: None

Data: ObjectInfo

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_ObjectHandle, Store_Not_Available, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Returns the ObjectInfo dataset, as described in Clause 5.5.2. The primary purpose of this operation is to obtain information about a data object present on the device before deciding whether to retrieve that object or its thumbnail with a succeeding GetThumb or GetObject operation. This information may also be used by the caller to allocate memory before receiving the object. Objects that possess an ObjectFormat of type Association do not require a GetObject operation, as these objects are fully qualified by their ObjectInfo dataset.

10.4.9 GetObject

OperationCode: `0x1009`

Operation Parameter1: ObjectHandle

Operation Parameter2: None

Operation Parameter3: None

Data: DataObject

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_ObjectHandle, Invalid_Parameter, Store_Not_Available, Parameter_Not_Supported, Incomplete_Transfer

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Retrieves one object from the device. This operation is used for all types of data objects present on the device, including both images and non-image data objects, and should be preceded (although not necessarily immediately) by a GetObjectInfo operation that uses the same ObjectHandle. This operation is not necessary for objects of type Association, as these objects are fully qualified by their ObjectInfo dataset. If the store that contains the object being sent is removed during the object transfer, the Incomplete_Transfer response should be used, along with the Store_Removed event.

10.4.10 GetThumb

OperationCode: `0x100A`

Operation Parameter1: ObjectHandle

Operation Parameter2: None

Operation Parameter3: None

Data: ThumbnailObject

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_ObjectHandle, Thumbnail_Not_Present, Invalid_ObjectFormatCode, Store_Not_Available, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Retrieves the thumbnail from the device that is associated with the ObjectHandle that is indicated in the first parameter.

10.4.11 DeleteObject

OperationCode: `0x100B`

Operation Parameter1: ObjectHandle

Operation Parameter2: [ObjectFormatCode]

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_ObjectHandle, Object_WriteProtected, Store_Read_Only, Partial_Deletion, Store_Not_Available, Specification_By_Format_Unsupported, Invalid_Code_Format, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Deletes the data object specified by the ObjectHandle from the device if it is not protected. If the ObjectHandle passed has the value of 0xFFFFFFFF, then all objects on the device shall be deleted. Any write-protected objects shall also not be deleted by this operation. If one object is indicated for deletion and it is write-protected, the response code Object_WriteProtected shall be returned. If all objects are indicated for deletion and a subset of the objects are write-protected, only the objects that are not protected shall be deleted, and the response code of Partial_Deletion shall be returned. If the store is read-only without object deletion, the response Store_Read_Only should be returned. If the store is read-only with object deletion, this operation should succeed unless other factors prevent it from succeeding.

The second parameter is optional, and may not be supported. This parameter may only be used if the first parameter is set to 0xFFFFFFFF. This parameter is used to indicate that objects only of the type specified are to be deleted. If this second parameter is also set to 0xFFFFFFFF, then only objects that are images shall be deleted. If it is not used, it shall be set to 0x00000000. If the value is non-zero, and the Responder does not support specification by ObjectFormatCode, it should fail the operation by returning a ResponseCode with the value of Specification_By_Format_Unsupported.

If the ObjectHandle indicated in the first parameter is an Association, then all objects that are a part of that association (and all descendants of descendants) shall be deleted as well. If only individual items within an association are to be deleted, then individual DeleteObject operations should be issued on each object or sub-association individually.

10.4.12 SendObjectInfo

OperationCode: `0x100C`

Operation Parameter1: [Destination StorageID on Responder]

Operation Parameter2: [Parent ObjectHandle on Responder where object should be placed]

Operation Parameter3: None

Data: ObjectInfo

Data Direction: I -> R

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Access_Denied, Invalid_StorageID, Store_Read_Only, Store_Full, Invalid_ObjectFormatCode, Store_Not_Available, Parameter_Not_Supported, Invalid_ParentObject, Invalid_Dataset

Response Parameter1: Responder StorageID in which the object will be stored

Response Parameter2: Responder Parent ObjectHandle in which the object will be stored

Response Parameter3: Responder's reserved ObjectHandle for the incoming object

Description: This operation is used as the first operation when the Initiator wishes to send an object to the Responder. This operation sends an ObjectInfo dataset from the Initiator to the Responder. All the fields in this ObjectInfo dataset are from the perspective of the Initiator, meaning that the StorageID, for example, would be interpreted as the StorageID of the store in which the object resides on the Initiator before being sent to the Responder. This operation is sent prior to the SendObject operation, described in Clause 10.4.13, in order to inform the Responder about the properties of the object that it intends to send later, and to effectively ask permission whether the object can be sent to the Responder. A response of OK infers that the receiver can accept the object, and serves to inform the sender that it may now issue a SendObject operation for the object.

The first parameter is optionally used to indicate the store on the Responder into which the object should be stored. If this parameter is specified, and the Responder will not be able to store the object in the indicated store, the operation should fail, and the appropriate response, such as Specification_Of_Destination_Unsupported, Store_Not_Available, Store_Read_Only, or Store_Full should be used. If this parameter is unused, it should be set to 0x00000000, and the Responder shall decide in which store to place the object, be that a Responder-determined default location, or the location with the most room (or possibly the only location with enough room).

The second parameter is optionally used to indicate where on the indicated store the object should be placed (i.e. the association/folder that the object should become a child of.) If this parameter is used, the first parameter must also be used. If the receiver is unable to place the object as a child of the indicated second parameter, the operation should fail. If the problem with the attempted specification is the general inability of the receiving device to allow the specification of destination, the response Specification_of_Destination_Unsupported should be sent. This response infers that the Initiator should not try to specify a destination location in future invocations of SendObjectInfo, as all attempts at such specification will fail. If the problem is only with the particular destination specified, the Invalid_ObjectHandle or Invalid_ParentObject response should be used, depending on whether the ObjectHandle did not refer to a valid object, or whether the indicated object is a valid object but is not an association. If the root directory of the indicated store is desired, the second parameter should be set to 0xFFFFFFFF. If this parameter is unused, it should be set to 0x00000000, and the Responder shall decide where in the indicated store the object is to be placed. If neither the first nor the second parameter is used, the Responder shall

decide both which store to place the object in as well as where to place it within that store.

If the Responder agrees that the object may be sent, it is required to retain this ObjectInfo dataset until the next SendObject or SendObjectInfo operation is performed subsequently within the session. If the SendObjectInfo operation succeeds, and the next occurring SendObject operation does not return a successful response, the SendObjectInfo held by the Responder shall be retained in case the Initiator wishes to re-attempt the SendObject operation for that previously successful SendObjectInfo operation. If the Initiator wishes to resend the ObjectInfo dataset before attempting to resend the object it may do so. Successful completion of the SendObjectInfo operation conveys that the Responder possesses a copy of the ObjectInfo and that the Responder has allocated space for the incoming data object. Any other response code other than OK indicates that the Responder has not retained the ObjectInfo dataset, and that the object should not attempt to be sent.

For a particular session, the receiving device shall only retain one ObjectInfo that is the result of a SendObjectInfo operation in memory at a time. If another SendObjectInfo operation occurs before a SendObject operation, the new ObjectInfo shall replace the previously held one. If this occurs, any storage space or memory space reserved for the object described in the overwritten ObjectInfo dataset should be freed before overwriting and allocation of the resources for the new ObjectInfo dataset.

The first response parameter of this operation should be set to the StorageID that the Responder will store the object into if it sent. The second response parameter of this operation should be set to the Parent ObjectHandle of the association that the object becomes a child of. If the object is stored in the root of the store, this parameter should be set to 0xFFFFFFFF.

If the Initiator wishes to retain associations and/or hierarchies on the Responder for the objects it is sending, then the objects should be sent top down, starting with the highest level of the hierarchy, proceeding in either a depth-first or breadth-first fashion down the hierarchy tree. The Initiator shall use the Responder's newly assigned ObjectHandle in the third response parameter for the ParentObject that is returned in the SendObjectInfo response as the second operation parameter for a child's SendObjectInfo operation.

10.4.13 SendObject

OperationCode: `0x100D`

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: DataObject

Data Direction: I -> R

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Store_Full, Store_Not_Available, No_Valid_ObjectInfo, Device_Busy, Parameter_Not_Supported, Incomplete_Transfer

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: This operation is used as the second operation when the Initiator wishes to send an object to the Responder, following the SendObjectInfo operation, described in Clause 10.4.12. This operation sends a data object to the device to be written to the Responder's store, according to the information in the ObjectInfo dataset as transmitted during the most recent SendObjectInfo operation in the same session, and the information indicated by the responder in the response parameters of the SendObjectInfo.

Upon successful completion of this operation, the Responder should discard and/or invalidate the Initiator's ObjectInfo that the Responder held while waiting for that object. If there is no valid ObjectInfo held by the Responder, the response No_Valid_ObjectInfo should be returned. Any response other than OK indicates that the SendObject failed, for the reason indicated by the response code. In this case, the unassigned ObjectInfo should be retained by the Responder in case the Initiator wishes to attempt to resend the object, for at most the duration of the session. If the destination store is removed during object transmission, the Incomplete_Transfer response should be issued along with the StoreRemoved event.

10.4.14 InitiateCapture

OperationCode: `0x100E`

Operation Parameter1: [StorageID]

Operation Parameter2: [ObjectFormatCode]

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_StorageID, Store_Full, Invalid_ObjectFormatCode, Invalid_Parameter, Store_Not_Available, Invalid_Code_Format, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Causes the device to initiate the capture of one or more new data objects according to its current device properties, storing the data into the store indicated by the first parameter. If the StorageID is 0x00000000, the object(s) will be stored in a store that is determined by the capturing device. If the particular store specified is not available, or no store is specified and there are no stores available, this operation should return Store_Not_Available.

The capturing of new data objects is an asynchronous operation. This operation may be used to capture images or any type of data that can be fully captured using a single operation trigger. For these types of captures, the length of the capture and the number of objects to capture is known apriori by the Responder, as opposed to being dynamically terminable after capture initiation by the Initiator. A separate operation, InitiateOpenCapture, described in Clause 10.4.28, can be used to support dynamically controlled captures that are terminable by the Initiator.

If the ObjectFormatCode in the second operation parameter is 0x00000000, the device shall capture an image in the format that is the default for the device. A successful response to an InitiateCapture operation indicates the Responder's acceptance of the InitiateCapture operation, and not the completion status of the actual object capture, which is indicated using the CaptureComplete event.

As the capture is executed, one or more new data objects should be created on the device. The number of objects to be captured is not specified as part of the InitiateCapture operation, but is determined by the state of the capturing device, and may optionally be set by the Initiator using an appropriate DeviceProperty. As each of the newly captured objects becomes available, the Responder is required to send an ObjectAdded event to the Initiator, indicating the ObjectHandle that is assigned to each as described in Clause 12.5.2. This ObjectAdded event shall contain the TransactionID of the InitiateCapture operation with which it is associated. If, at any time, the store becomes full, the device shall invoke a Store_Full event, which shall contain the TransactionID of the InitiateCapture operation that failed to cause a new object to be stored. In the case of multiple objects being captured, each object shall be handled separately, so any object captured before the store becomes full should be retained. When all objects have been captured, the Responder shall send a CaptureComplete event to the Initiator. If the Store_Full event was issued, the CaptureComplete event should not be issued. If another capture is occurring when this operation is invoked, the Device_Busy response should be used.

Figure 6: Single Object InitiateCapture Sequence

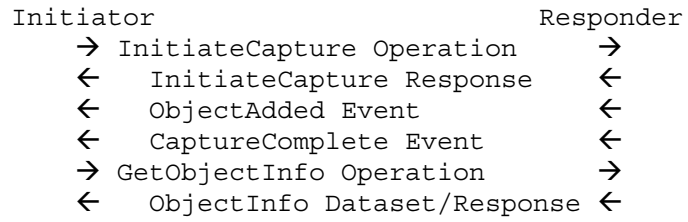
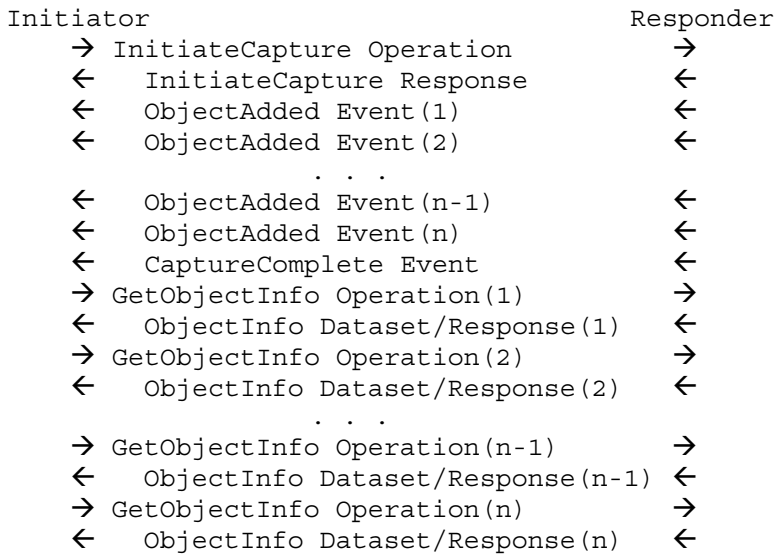


Figure 7: Multiple Object InitiateCapture Sequence



10.4.15 FormatStore

OperationCode: `0x100F`

Operation Parameter1: StorageID

Operation Parameter2: [FilesystemFormat]

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_StorageID, Store_Not_Available, Device_Busy, Parameter_Not_Supported, Invalid_Parameter, Store_Read_Only

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Formats the media specified by the StorageID. The second parameter is optional and may be used to indicate the format that the store should be formatted in, according to the FilesystemFormat codes described in Clause 5.5.3. If a given format is not supported, the response Invalid_Parameter should be returned. If the device is currently capturing objects to the store, or is otherwise unable to format due to concurrent access, the Device_Busy operation should be returned.

10.4.16 ResetDevice

OperationCode: `0x1010`

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Device_Busy

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Resets the device to its device-dependent default state. This does not include resetting any device properties, which is performed using ResetDeviceProp. This does include closing the current session, and any other open sessions. If this operation is supported and the device supports multiple concurrent sessions, the device is responsible for supporting the DeviceReset event, which should be sent to all open sessions excluding the one within which the ResetDevice operation was initiated prior to closing the sessions.

10.4.17 SelfTest

OperationCode: `0x1011`

Operation Parameter1: [SelfTestType]

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, SelfTest_Failed, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Causes the device to initiate a device-dependent self-test. The first parameter is used to indicate the type of self-test that should be performed, according to the following table:

Table 19: SelfTestType Values

Value	Description
0x0000	Default device-specific self-test
All other values with Bit 15 set to 0	Reserved
All values with Bit 15 set to 1	Vendor-Defined

10.4.18 SetObjectProtection

OperationCode: 0x1012

Operation Parameter1: ObjectHandle

Operation Parameter2: ProtectionStatus

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Access_Denied, Invalid_ObjectHandle, Invalid_Parameter, Store_Not_Available, Parameter_Not_Supported, Store_Read_Only

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Sets the write-protection status for the data object referred to in the first parameter to the value indicated in the second parameter. For a description of the ProtectionStatus field, refer to the ObjectInfo dataset described in Clause 5.5.2. If the ProtectionStatus field does not hold a legal value, the ResponseCode should be Invalid_Parameter.

10.4.19 PowerDown

OperationCode: 0x1013

Operation Parameter1: None

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Causes the device to power down. This will cause all currently open sessions to close.

10.4.20 GetDevicePropDesc

OperationCode: `0x1014`

Operation Parameter1: DevicePropCode

Operation Parameter2: None

Operation Parameter3: None

Data: DevicePropDesc Dataset

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Access_Denied, DeviceProp_Not_Supported, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Returns the appropriate Property Describing Dataset as indicated by the first parameter.

10.4.21 GetDevicePropValue

OperationCode: `0x1015`

Operation Parameter1: DevicePropCode

Operation Parameter2: None

Operation Parameter3: None

Data: DeviceProperty Value

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, DeviceProp_Not_Supported, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Returns the current value of a property. The size and format of the data returned from this operation should be determined from the corresponding DevicePropDesc dataset returned from the GetDevicePropDesc operation. The current value of a property can also be retrieved directly from the DevicePropDesc, so this operation is not typically required unless a DevicePropChanged event occurs.

10.4.22 SetDevicePropValue

OperationCode: `0x1016`

Operation Parameter1: DevicePropCode

Operation Parameter2: None

Operation Parameter3: None

Data: Device Property Value

Data Direction: I -> R

ResponseCode Options: OK, Session_Not_Open, Invalid_TransactionID, Access_Denied, DeviceProp_Not_Supported, Property_Not_Supported, Invalid_DeviceProp_Format, Invalid_DeviceProp_Value, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Sets the current value of the device property indicated by parameter 1 to the value indicated in the data phase of this operation. The format of the property value object sent in the data phase can be determined from the DatatypeCode field of the property's DevicePropDesc dataset. If the property is not settable, the response Access_Denied should be returned. If the value is not allowed by the device, Invalid_DeviceProp_Value should be returned. If the format or size of the property value is incorrect, Invalid_DeviceProp_Format should be returned.

10.4.23 ResetDevicePropValue

OperationCode: `0x1017`

Operation Parameter1: DevicePropCode

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: None

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, DeviceProp_Not_Supported, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Sets the value of the indicated device property to the factory default setting. The first parameter may be set to 0xFFFFFFFF to indicate that all properties should be reset to their factory default settings.

10.4.24 TerminateOpenCapture

OperationCode: `0x1018`

Operation Parameter1: TransactionID

Operation Parameter2: None

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Parameter_Not_Supported, Invalid_Parameter, Capture_Already_Terminated

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: This operation is used after an InitiateOpenCapture operation for situations where the capture operation length is open-ended, and determined by the Initiator. This operation is not used for trigger captures, which are invoked using a separate operation, InitiateCapture, described in Clause 10.4.14. This operation allows the termination of one capture operation that is being used to capture many objects over some period of time, such as a burst, or for long single objects such as manually-controlled image exposures, audio captures, or video clips. The first parameter of this operation indicates the TransactionID of the InitiateOpenCapture operation that is being terminated. If the capture has already terminated for some other reason, this operation should return Capture_Already_Terminated. If the TransactionID parameter does not refer to transaction that was an InitiateOpenCapture, this operation should return Invalid_TransactionID.

10.4.25 MoveObject

OperationCode: 0x1019

Operation Parameter1: ObjectHandle

Operation Parameter2: StorageID of store to move object to

Operation Parameter3: ObjectHandle of new ParentObject

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Store_Read_Only, Store_Not_Available, Invalid_ObjectHandle, Invalid_ParentObject, Device_Busy, Parameter_Not_Supported, Invalid_StorageHandle

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: This operation causes the object to be moved from its location within the hierarchy to a new location indicated by the second and third parameters. If the root of the store is desired, the third parameter may be set to 0x00000000. If the third parameter does not refer to a valid object of type Association, the response Invalid_ParentObject should be returned. If a store is read-only (with or without deletion) the response Store_Read_Only should be returned. This operation does not cause the ObjectHandle of the object that is being moved to change. If the object is to be moved

10.4.26 CopyObject

OperationCode: 0x101A

Operation Parameter1: ObjectHandle

Operation Parameter2: StorageID that the newly copied object should be placed into

Operation Parameter3: ObjectHandle of newly copied object's parent

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Store_Read_Only, Invalid_ObjectHandle, Invalid_ParentObject, Device_Busy, Store_Full, Parameter_Not_Supported, Invalid_StorageID

Response Parameter1: ObjectHandle of new copy of object

Response Parameter2: None

Response Parameter3: None

Description: This operation causes the object to be replicated within the Responder. The first parameter refers to the ObjectHandle of the object that is to be copied. The second parameter refers to the StorageID into which the newly copied object should be placed. The third parameter refers to the ParentObject of where the newly replicated copy should be placed. If the object is to be copied into the root of the store, this value should be set to 0x00000000.

10.4.27 GetPartialObject

OperationCode: `0x101B`

Operation Parameter1: ObjectHandle

Operation Parameter2: Offset in bytes

Operation Parameter3: Maximum number of bytes to obtain

Data: DataObject

Data Direction: R -> I

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_ObjectHandle, Invalid_ObjectFormatCode, Invalid_Parameter, Store_Not_Available, Device_Busy, Parameter_Not_Supported

Response Parameter1: Actual number of bytes sent

Response Parameter2: None

Response Parameter3: None

Description: Retrieves a partial object from the device. This operation is optional, and may be used in place of the GetObject operation for devices that support this alternative. If supported, this operation should be generic, and therefore useable with all types of data objects present on the device, including both images and non-image data objects, and should be preceded (although not necessarily immediately) by a GetObjectInfo operation that uses the same ObjectHandle. For this operation, the size fields in the ObjectInfo represent maximum size as opposed to actual size. This operation is not necessary for objects of type Association, as objects of this type are fully qualified by their ObjectInfo dataset.

The operation behaves exactly like GetObject, except that the second and third parameters hold the offset in bytes and the number of bytes to obtain starting from the offset, respectively. If the portion of the object that is desired is from the offset to the end, the third parameter may be set to 0xFFFFFFFF. The first response parameter should contain the actual number of bytes of the object sent, not including any wrappers or overhead structures.

10.4.28 InitiateOpenCapture

OperationCode: 0x101C

Operation Parameter1: [StorageID]

Operation Parameter2: [ObjectFormatCode]

Operation Parameter3: None

Data: None

Data Direction: N/A

ResponseCode Options: OK, Operation_Not_Supported, Session_Not_Open, Invalid_TransactionID, Invalid_StorageID, Store_Full, Invalid_ObjectFormatCode, Invalid_Parameter, Store_Not_Available, Invalid_Code_Format, Device_Busy, Parameter_Not_Supported

Response Parameter1: None

Response Parameter2: None

Response Parameter3: None

Description: Causes the device to initiate the capture of one or more new data objects according to its current device properties, storing the data into the store indicated by the StorageID. If the StorageID is 0x00000000, the object(s) will be stored in a store that is determined by the capturing device. If the particular store specified is not available, or no store is specified and there are no stores available, this operation should return Store_Not_Available.

The capturing of new data objects is an asynchronous operation. This operation may be used to implement an Initiate/Terminate mechanism to capture one or more objects over an Initiator-controlled time period, such as a single long still exposure, a series of stills, audio capture, etc. Whether the time period controls the time of capture for a single object or the number of fixed-time objects that are captured is determined by the Responder, and may be a function of the ObjectFormat as well as any appropriate DeviceProperties.

A separate operation, InitiateCapture, described in Clause 10.4.14, can be used to support captures that do not require the Initiator to indicate when the capture should terminate.

If the ObjectFormatCode in the second operation parameter is 0x00000000, the device shall capture an image in the format that is the default for the device. A successful response to an InitiateOpenCapture operation indicates the Responder's acceptance of the InitiateOpenCapture operation, and not the completion status of the capture operation.

A successful response to the InitiateOpenCapture operation implies that the Responder has started to capture one or more objects. When the Initiator wishes to terminate the capture, it is required to send a TerminateOpenCapture operation. The CaptureComplete

event is not used for this operation, as the end of the capture period is determined by the Initiator. As each of the newly captured objects becomes available, the Responder is required to send an ObjectAdded event to the Initiator, indicating the ObjectHandle that is assigned to each as described in Clause 12.5.2. The ObjectAdded event shall contain the TransactionID of the InitiateOpenCapture operation with which it is associated. If, at any time, the store becomes full, the device shall issue a Store_Full event, which shall contain the TransactionID of the InitiateOpenCapture operation that failed to cause a new object to be stored. In the case of multiple objects being captured, each object shall be treated separately, so any object captured before the store becomes full should be retained. Whether or not an object that was partially captured can be retained and used is a function of the device's behavior and object format. For example, if the device runs out of room while capturing a video clip, it may be able to save the portion that it had room to store. Any object that is retained in these situations should cause an ObjectAdded event to be issued, while any object that is not retained should cause no event to be issued. A Store_Full event effectively terminates the capture, and in these cases, issuing the TerminateOpenCapture operation is not used. If another object capture is occurring when this operation is invoked, the Device_Busy response should be used.

Figure 8: Single Object InitiateOpenCapture Sequence

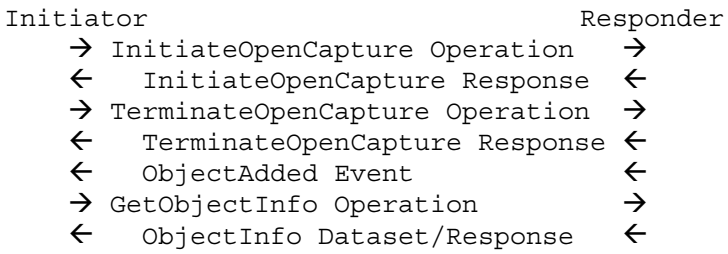
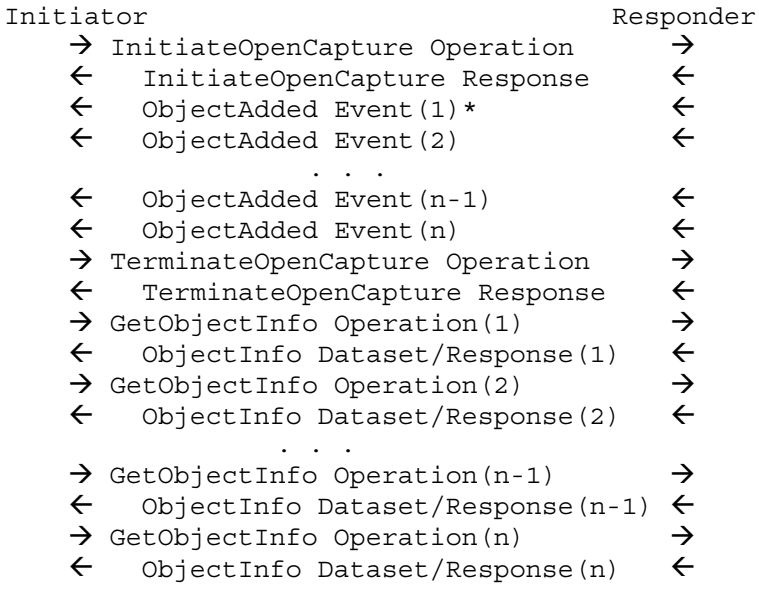


Figure 9: Multiple Object InitiateOpenCapture Sequence



11 Responses

11.1 ResponseCode Format

ResponseCodes are part of the response dataset described in Clause 9.3.3. All ResponseCodes shall take the form of a 16-bit integer, are referred to using hexadecimal notation, have bits 12 and 14 set to 0, and bit 13 set to 1. All non-defined ResponseCodes having bit 15 set to zero are reserved for future use. If a proprietary implementation wishes to define a proprietary ResponseCode, bit 15 should be set to 1 as well.

ResponseCodes other than “OK” indicate failure of an operation to complete.

11.2 ResponseCode Summary

The following table summarizes the ResponseCodes defined by this standard.

Table 20: ResponseCode Summary

ResponseCode	Description
0x2000	Undefined
0x2001	OK
0x2002	General Error
0x2003	Session Not Open
0x2004	Invalid TransactionID
0x2005	Operation Not Supported
0x2006	Parameter Not Supported
0x2007	Incomplete Transfer
0x2008	Invalid StorageID
0x2009	Invalid ObjectHandle
0x200A	DeviceProp Not Supported
0x200B	Invalid ObjectFormatCode
0x200C	Store Full
0x200D	Object WriteProtected
0x200E	Store Read-Only
0x200F	Access Denied
0x2010	No Thumbnail Present
0x2011	SelfTest Failed
0x2012	Partial Deletion
0x2013	Store Not Available
0x2014	Specification By Format Unsupported
0x2015	No Valid ObjectInfo
0x2016	Invalid Code Format
0x2017	Unknown Vendor Code
0x2018	Capture Already Terminated
0x2019	Device Busy
0x201A	Invalid ParentObject
0x201B	Invalid DeviceProp Format
0x201C	Invalid DeviceProp Value
0x201D	Invalid Parameter
0x201E	Session Already Open
0x201F	Transaction Cancelled
0x2020	Specification of Destination Unsupported
All other codes with MSN of 0010	Reserved
All codes with MSN of 1010	Vendor-Extended Response Code

11.3 Response Descriptions

11.3.1 OK

ResponseCode: 0x2001

Desc: Operation completed successfully.

11.3.2 General Error

ResponseCode: 0x2002

Desc: Operation did not complete. This response is used when the cause of the error is unknown or there is no better failure ResponseCode to use.

11.3.3 Session Not Open

ResponseCode: 0x2003

Desc: Indicates that the session handle of the operation is not a currently open session.

11.3.4 Invalid TransactionID

ResponseCode: 0x2004

Desc: Indicates that the TransactionID is zero or does not refer to a valid transaction.

11.3.5 Operation Not Supported

ResponseCode: 0x2005

Desc: Indicates that the indicated OperationCode appears to be a valid code, but the Responder does not support the operation. This error should not normally occur, as the Initiator should only invoke operations that the Responder indicated were supported in its DeviceInfo dataset.

11.3.6 Parameter Not Supported

ResponseCode: 0x2006

Desc: Indicates that a non-zero parameter was specified in conjunction with the operation, and that parameter is not used for that operation. This response is distinctly different from Invalid_Parameter, as described in Clause 11.3.29.

11.3.7 Incomplete Transfer

ResponseCode: 0x2007

Desc: Indicates that the transfer did not complete. Any data transferred should be discarded. This response should not be used if the transaction was manually cancelled. See Response Transaction_Cancelled, described in Clause 11.3.31

11.3.8 Invalid StorageID

ResponseCode: 0x2008

Desc: Indicates that a StorageID sent with an operation does not refer to an actual valid store that is present on the device. The list of valid StorageIDs should be re-requested, along with any appropriate StorageInfo datasets.

11.3.9 Invalid ObjectHandle

ResponseCode: 0x2009

Desc: Indicates that an ObjectHandle does not refer to an actual object that is present on the device. The list of valid ObjectHandles should be re-requested, along with any appropriate ObjectInfo datasets.

11.3.10 DeviceProp Not Supported

ResponseCode: 0x200A

Desc: The indicated DevicePropCode appears to be a valid code, but that property is not supported by the device. This response should not normally occur, as the Initiator should only attempt to manipulate properties that the Responder indicated were supported in the DevicePropertiesSupported array in the DeviceInfo dataset.

11.3.11 Invalid ObjectFormatCode

ResponseCode: 0x200B

Desc: Indicates that the device does not support the particular ObjectFormatCode supplied in the given context.

11.3.12 Store Full

ResponseCode: 0x200C

Desc: Indicates that the store that the operation referred to is full.

11.3.13 Object WriteProtected

ResponseCode: `0x200D`

Desc: Indicates that the object that the operation referred to is write-protected.

11.3.14 Store Read-Only

ResponseCode: `0x200E`

Desc: Indicates that the store that the operation referred to is read-only.

11.3.15 Access Denied

ResponseCode: `0x200F`

Desc: Indicates the access to the data referred to by the operation was denied. The intent of this response is not to indicate that the device is busy, but that given the current state of the device does not change, access will be continued to be denied.

11.3.16 No Thumbnail Present

ResponseCode: `0x2010`

Desc: Indicates that a data object exists with the specified ObjectHandle, but the data object does not contain a producible thumbnail.

11.3.17 Self Test Failed

ResponseCode: `0x2011`

Desc: Indicates that the device failed an internal device-specific self-test.

11.3.18 Partial Deletion

ResponseCode: `0x2012`

Desc: Indicates that only a subset of the objects indicated for deletion were actually deleted, due to the fact that some were write-protected, or that some objects were on stores that are read-only.

11.3.19 Store Not Available

ResponseCode: 0x2013

Desc: Indicates that the store indicated (or the store that contains the indicated object) is not physically available. This can be caused by media ejection. This response shall not be used to indicate that the store is busy, as described in Clause 11.3.25.

11.3.20 Specification By Format Unsupported

ResponseCode: 0x2014

Desc: Indicates that the operation attempted to specify action only on objects of a particular format, and that capability is unsupported. The operation should be re-attempted without specifying by format. Any response of this nature infers that any future attempt to specify by format with the indicated operation will result in the same response.

11.3.21 No Valid ObjectInfo

ResponseCode: 0x2015

Desc: Indicates that the Initiator attempted to issue a SendObject operation without having previously sent a corresponding SendObjectInfo successfully. The Initiator should successfully complete a SendObjectInfo operation before attempting another SendObject operation.

11.3.22 Invalid Code Format

ResponseCode: 0x2016

Desc: Indicates that the indicated data code does not have the correct format, and is therefore invalid. This response is used when the most-significant nibble of a datacode does not have the format required for that type of code.

11.3.23 Unknown Vendor Code

ResponseCode: 0x2017

Desc: Indicates that the indicated data code has the correct format, but has bit 15 set to 1. Therefore, the code is a vendor-extended code, and this device does not know how to handle the indicated code. This response should typically not occur, as the supported vendor extensions should be identifiable by examination of the VendorExtensionID and VendorExtensionVersion fields in the DeviceInfo dataset.

11.3.24 Capture Already Terminated

ResponseCode: 0x2018

Desc: Indicates that an operation is attempting to terminate a capture session initiated by a preceding InitiateOpenCapture operation, and that preceding operation has already terminated. This response is only used for the TerminateOpenCapture operation, which is only used for open-ended captures. Clause 10.4.24 provides a description of the TerminateOpenCapture operation.

11.3.25 Device Busy

ResponseCode: 0x2019

Desc: Indicates that the device is not currently able to process a request because it, or the specified store, is busy. The intent of this response is to imply that perhaps at a future time, the operation should be re-requested. This response shall not be used to indicate that a store is physically unavailable, as described in Clause 11.3.19.

11.3.26 Invalid ParentObject

ResponseCode: 0x201A

Desc: Indicates that the indicated object is not of type Association, and therefore is not a valid ParentObject. This response is not intended to be used for specified ObjectHandles that do not refer to valid objects, which is handled by the Invalid_ObjectHandle response described in Clause 11.3.9.

11.3.27 Invalid DeviceProp Format

ResponseCode: 0x201B

Desc: Indicates that an attempt was made to set a DeviceProperty, but the DevicePropDesc dataset is not the correct size or format.

11.3.28 Invalid DeviceProp Value

ResponseCode: 0x201C

Desc: Indicates that an attempt was made to set a DeviceProperty to a value that is not allowed by the device.

11.3.29 Invalid Parameter

ResponseCode: 0x201D

Desc: Indicates that a parameter was specified in conjunction with the operation, and that although a parameter was expected, the value of the parameter is not a legal value. This response is distinctly different from Parameter_Not_Supported, as described in Clause 11.3.6.

11.3.30 Session Already Open

ResponseCode: 0x201E

Desc: This response code may be used as the response to an OpenSession operation. For multisession devices/transports, this response indicates that a session with the specified Session ID is already open. For single-session devices/transports, this response indicates that a session is open, and must be closed before another session can be opened.

11.3.31 Transaction Cancelled

ResponseCode: 0x201F

Desc: This response code may be used to indicate that the operation was interrupted due to manual cancellation by the opposing device.

11.3.32 Specification of Destination Unsupported

ResponseCode: 0x2020

Desc: This response code may be used as the response to a SendObjectInfo operation to indicate that the Responder does not support the specification of destination by the Initiator. This response infers that the Initiator should not attempt to specify the object destination in any future SendObjectInfo operations, as they will also fail with the same response.

12 Events

This section describes events, their datasets, and their usages. Event support shall be mandatory.

12.1 Event Types

Although either the Initiator or the Responder can send an event, most events are typically sent by the Responder. The Responder also uses events to communicate a state change (e.g. arrival of a new object or store) or optionally to ask the Initiator to start a transaction. CancelTransaction is used to cancel a transaction, and may be sent by either an Initiator or a Responder.

12.1.1 Transports with In-Band Events

For transports that use in-band events due to the lack of existence of a separate logical connection for events, a method of interleaving events into a data stream that meets the device's responsiveness requirements will need to be implemented in a transport-specific fashion.

12.1.2 Transports with Out-of-Band Events

Transports that use a separate logical connection for interrupts effectively have support for out-of-band events. This support means that such transport implementations will not need to break up long image transfers into smaller data blocks in order to accommodate the potential need to interleave events that may occur during the transfer.

12.2 Event Dataset

Events are described using the Event Dataset, which consists of the minimal information that is required for qualified notification. Fully qualified events will need to send only this dataset in order to fully describe the event and obtain post-event synchronization. In different transport implementations, the fulfillment of all the fields of this dataset may or may not happen atomically, and therefore may require a lightweight event notification mechanism separate from the operation of actually requesting the dataset fields. Some events, by the very nature of the information that they convey, will infer the need for the event-receiving device to perform an operation to re-synchronize inter-device state.

Table 21: Event Dataset

Field	Size	Format
EventCode	2 bytes	UINT16
SessionID	4 bytes	UINT32
TransactionID	4 bytes	UINT32
Parameter1	4 bytes	Any
Parameter2	4 bytes	Any
Parameter3	4 bytes	Any

SessionID: Indicates the SessionID of the session for which the event is relevant. If the event is relevant to all open sessions, this field should be set to 0xFFFFFFFF. Refer to Clause 9.2.1 for a description of session handles.

EventCode: Indicates the event as defined in the EventCode section.

TransactionID: If the event corresponds to a previously initiated transaction, this field shall hold the TransactionID of that operation. If the event is not specific to a particular transaction, this field shall be set to 0xFFFFFFFF. Refer to Clause 9.3.1 for a description of TransactionID.

Parameter *n*: This field holds the event-specific n^{th} parameter. Events may have at most three parameters. The interpretation of any parameter is dependent upon the EventCode. Any unused parameter fields should be set to 0x00000000. If a parameter holds a value that is less than 32 bits, the lowest significant bits shall be used to store the value, with the most significant bits being set to zeros.

12.3 EventCode Format

EventCodes are part of the event dataset described in Clause 12.2. All EventCodes shall take the form of a 16-bit integer, are referred to using hexadecimal notation, have bits 12 and 13 set to zero, and bit 14 set to 1. All non-defined ResponseCodes having bit 15 set to zero are reserved for future use. If a proprietary implementation wishes to define a proprietary EventCode, bit 15 should be set to 1 as well.

12.4 EventCode Summary

The following events are defined in this standard:

Table 22: EventCode Summary

EventCode	Name
0x4000	Undefined
0x4001	CancelTransaction
0x4002	ObjectAdded
0x4003	ObjectRemoved
0x4004	StoreAdded
0x4005	StoreRemoved
0x4006	DevicePropChanged
0x4007	ObjectInfoChanged
0x4008	DeviceInfoChanged
0x4009	RequestObjectTransfer
0x400A	StoreFull
0x400B	DeviceReset
0x400C	StorageInfoChanged
0x400D	CaptureComplete
0x400E	UnreportedStatus
All other codes with MSN of 0100	Reserved
All codes with MSN of 1100	Vendor-Extended Response Code

12.5 Event Descriptions

12.5.1 CancelTransaction

EventCode: 0x4001

Parameter1: None

Parameter2: None

Parameter3: None

Desc: This formal event is used to cancel a transaction for transports that do not have a specified or standard way of canceling transactions. The particular method used to cancel transactions may be transport-specific. When an Initiator or Responder receives a CancelTransaction event, it should abort the transaction referred to by the TransactionID in the event dataset. If that transaction is already complete, the event should be ignored.

After receiving a CancelTransfer event from the Initiator, the Responder shall send an IncompleteTransfer response for the operation that was cancelled. Both devices will then be ready for the next transaction.

12.5.2 ObjectAdded

EventCode: 0x4002

Parameter1: ObjectHandle

Parameter2: None

Parameter3: None

Desc: A new data object was added to the device. The new handle assigned by the device to the new object should be passed in the Parameter1 field of the event. If more than one object was added, each new object should generate a separate ObjectAdded event. The appearance of a new store on the device should not cause the creation of new ObjectAdded events for the new objects present on the new store, but should instead cause the generation of a StoreAdded event, as described in Clause 12.5.4.

12.5.3 ObjectRemoved

EventCode: 0x4003

Parameter1: ObjectHandle

Parameter2: None

Parameter3: None

Desc: A data object was removed from the device unexpectedly due to something external to the current session. The handle of the object that was removed should be passed in the Parameter1 field of the event. If more than one image was removed, the separate ObjectRemoved events should be generated for each. If the data object that was removed was removed because of a previous operation that is a part of this session, no event needs to be sent to the opposing device. The removal of a store on the device should not cause the creation of ObjectRemoved events for the objects present on the removed store, but should instead cause the generation of one StoreRemoved event with the appropriate PhysicalStorageID, as described in Clause 12.5.5.

12.5.4 StoreAdded

EventCode: 0x4004

Parameter1: StorageID

Parameter2: None

Parameter3: None

Desc: A new store was added to the device. If this is a new physical store that contains only one logical store, then the complete StorageID of the new store should be indicated in the first parameter. If the new store contains more than one logical store, then the first parameter should be set to 0x00000000. This indicates that the list of StorageIDs should be re-obtained using the GetStorageIDs operation, and examined appropriately. Any new StorageIDs discovered should result in the appropriate invocations of GetStorageInfo operations, as described in Clause 10.4.5.

12.5.5 StoreRemoved

EventCode: `0x4005`

Parameter1: StorageID

Parameter2: None

Parameter3: None

Desc: The indicated stores are no longer available. The opposing device may assume that the StorageInfo datasets and ObjectHandles associated with those stores are no longer valid. The first parameter is used to indicate the StorageID of the store that is no longer available. If the store removed is only a single logical store within a physical store, the entire StorageID should be sent, which indicates that any other logical stores on that physical store are still available. If the physical store and all logical stores upon it are removed (e.g. removal of an ejectable media with multiple partitions), the first parameter should contain the PhysicalStorageID in the most significant sixteen bits, with the least significant sixteen bits set to 0xFFFF.

12.5.6 DevicePropChanged

EventCode: `0x4006`

Parameter1: DevicePropCode

Parameter2: None

Parameter3: None

Desc: A property changed on the device due to something external to this session. The appropriate property dataset should be requested from the opposing device.

12.5.7 ObjectInfoChanged

EventCode: `0x4007`

Parameter1: ObjectHandle

Parameter2: None

Parameter3: None

Desc: Indicates that the ObjectInfo dataset for a particular object has changed, and that it should be re-requested.

12.5.8 DeviceInfoChanged

EventCode: 0x4008

Parameter1: None

Parameter2: None

Parameter3: None

Desc: Indicates that the capabilities of the device have changed, and that the DeviceInfo should be re-requested. This may be caused by the device going into or out of a sleep state, or by the device losing or gaining some functionality in some way.

12.5.9 RequestObjectTransfer

EventCode: 0x4009

Parameter1: ObjectHandle

Parameter2: None

Parameter3: None

Desc: This event can be used by a Responder to ask the Initiator to initiate a GetObject operation on the handle specified in the first parameter. This allows for push-mode to be enabled on devices/transports that are intrinsically pull mode.

12.5.10 Store Full

EventCode: 0x400A

Parameter1: StorageID

Parameter2: None

Parameter3: None

Desc: This event shall be sent when a store becomes full. Any multi-object capture that may be occurring should retain the objects that were written to a store before the store became full.

12.5.11 Device Reset

EventCode: 0x400B

Parameter1: None

Parameter2: None

Parameter3: None

Desc: This event needs only to be supported for devices that support multiple sessions or in the case if the device is capable of resetting itself automatically or manually through user intervention while connected. This event shall be sent to all open sessions other than the session that initiated the operation. This event shall be interpreted as indicating that the sessions are about to be closed.

12.5.12 StorageInfoChanged

EventCode: `0x400C`

Parameter1: StorageID

Parameter2: None

Parameter3: None

Desc: This event is used when information in the StorageInfo dataset for a store changes. This can occur due to device properties changing, such as ImageSize, which can cause changes in fields such as FreeSpaceInImages. This event is typically not needed if the change is caused by an in-session operation that affects whole objects in a deterministic manner. This includes changes in FreeSpaceInImages or FreeSpaceInBytes caused by operations such as InitiateCapture or CopyObject, where the Initiator can recognize the changes due to the successful response code of the operation, and/or related required events.

12.5.13 CaptureComplete

EventCode: `0x400D`

Parameter1: TransactionID

Parameter2: None

Parameter3: None

Desc: This event is used to indicate that a capture session, previously initiated by the InitiateCapture operation, is complete, and that no more ObjectAdded events will occur as the result of this asynchronous operation. This operation is not used for InitiateOpenCapture operations.

12.5.14 UnreportedStatus

EventCode: `0x400E`

Parameter1: None

Parameter2: None

Parameter3: None

Desc: This event may be implemented for certain transports where situations can arise where the Responder was unable to report events to the Initiator regarding changes in its internal status. When an Initiator receives this event, it is responsible for doing whatever is necessary to ensure that its knowledge of the Responder is up to date. This may include re-obtaining individual datasets, ObjectHandle lists, etc., or may even result in the session being closed and re-opened. This event is typically only needed in situations where the transport used by the device supports a suspend/resume/remote-wakeup feature and the Responder has gone into a suspend state and has been unable to report state changes during that time period. This prevents the need for queuing of these unreportable events. The details of the use of this event are transport-specific and should be fully specified in the specific transport implementation specification.

13 Device Properties

A device that is conformant to this standard may optionally provide different modes of operations or different attributes that can be modified. Collectively these items comprise the device properties. Properties are attributes of the device, and not any particular data object contained in the device. Each property has an associated DevicePropCode. Attributes of individual data objects may be determined by examining their corresponding ObjectInfo datasets as described in Clause 5.5.2, or are contained inside the data objects themselves in a manner specified by its data format, as described by the ObjectFormat field.

13.1 Values of a Device Property

This standard defines how the device is to report the values that it supports for a given property and a means for controlling the setting of the property value. This standard also defines how the physical units and the data type that the values of a particular property are cast. The device vendor chooses the set of properties to implement.

This standard identifies the methods used to:

- Determine the value of the factory default setting of a particular device property.
- Determine the value of the current setting of a particular device property.
- Change the current value of a particular device property.
- Describe an enumerated list of the properties that the device supports.
- Describe the values supported for a given property for a given device.

In a conforming PIMA15740 device:

- Device properties may be read-only or read-write.
- Multi-session devices shall have one global set of device properties that apply to all sessions. A change in a property caused by one session shall cause a DevicePropChanged event to be issued to all other sessions for each property that changes as the result of the initial change.
- Single standard operations are able to be set or retrieved only one device property at a time. Vendor extended properties may be provided that handle multiple properties for one operation.
- Property functionality should be able to take advantage of any event mechanism supported by the transport. (e.g. a change in a device property value should be able to transparently signal an event to notify any connected devices of a change in state).

13.2 Device Property Management Requirements

This section describes requirements around the management of properties.

A device that is conformant with this standard may be requested to perform the following operations.

- The device may be queried (by an initiating device) to determine the properties that it supports.
- The device may be queried (by an initiating device) to determine the possible values for a particular property that it supports.
- The device may be queried (by an initiating device) to determine the current value of a particular property.
- The device may be requested (by an initiating device) to change a current property value to a new value, for properties that are identified as being settable.

13.2.1 Device Property Interdependencies

Upon the completion of an operation that requests a change in a device property, other properties on the device may change due to property interdependency. This standard does not attempt to provide a mechanism for a-priori discovery of these interdependencies. Instead, each operation is limited to directly changing only single properties at a time. If multiple properties are tightly integrated into one multi-dimensional property (e.g. ImageSize, RGBgain), one string property may be used with multiple fields separated by specified delimiters. Each time a property is changed, the device is responsible for changing any other properties in a device-dependent way that are affected by the new property setting. This typically includes defaulting to a value that is compatible with the new value of the property for which the change was requested. In these cases, the device is required to notify all open sessions with a DevicePropChanged event for each property that changed as a side effect of the requested change.

13.3 Device Property Identification

This standard defines a set of DevicePropCodes that identify common properties recognized by all conforming devices capable of functioning as an Initiator. A conforming device shall provide information describing what properties it supports. The conforming device shall also provide information that describes the particular values of a property supported by the device. This information is returned to the initiating device when the responding device is queried for the supported values of a particular property.

13.3.1 Device Property Describing Requirements

The describing information shall contain the DevicePropCode, the data type of the property, the values of the property supported by the device, and an identification of which of the supported values is the current value as well as the factory default value.

13.3.2 Device Property Describing Methods

A device's supported values of a particular property may be described using one of two methods. The device may return a list that enumerates all supported values of a particular property. In situations where the set of supported values comprise a linear range a triad of parameters; minimum, maximum, and step size may be used to describe the range of supported property values.

13.3.3 Device Property Describing Dataset

A PIMA15740 conforming device shall be capable of returning a Device Property Describing Dataset (DevicePropDesc) for each property supported. This dataset is used to hold the information detailing the datatype, allowed values, whether the property is settable, etc. This dataset also holds the current value of the property. This dataset is returned as the response to the GetDevicePropDesc operation. The GetDevicePropValue operation may also be used if only the current value of the device property is desired.

The supported values of a particular device property may be described by an enumeration of the values a device supports or by describing the minimum and maximum supported range of values with a corresponding inter-value step size. Two forms of the Property Describing Dataset are defined in this standard to enable the two describing methods. The following tables describe the content of the Device Property Describing Dataset.

Table 23: Device Property Describing Dataset (DevicePropDesc)

Field	Field Order	Size (Bytes)	Description
Device Property Code	1	2	A specific DevicePropCode
DataType	2	2	This field identifies the Datatype Code of the property, as indicated in Section 5.3
GetSet	3	1	This field indicates whether the property is read-only (Get) or read-write (Get/Set). 0x00 Get 0x01 Get/Set
Factory Default Value	4	DTS	This field identifies the value of the factory default setting for the property
Current Value	5	DTS	This field identifies the current value of the property
Form Flag	6	1	This field indicates the format of the next field. 0x00 None. This is for properties like DateTime. In this case the FORM field is not present. 0x01 Range-Form 0x02 Enumeration-Form
FORM	N/A	<variable>	This dataset is the Enumeration-Form or the Range-Form, or is absent if Form Flag = 0

Table 24: Property Describing Dataset, Range Form

Field	Field Order	Size (Bytes)	Description
MinimumValue	7	DTS	Minimum value of property supported by the device.
MaximumValue	8	DTS	Maximum value of property supported by the device.
StepSize	9	DTS	A particular vendor's device shall support all values of a property defined by MinimumValue + N x StepSize which is less than or equal to MaximumValue where N= 0 to a vendor defined maximum

Table 25: Property Describing Dataset, Enumeration Form

Field	Field Order	Size (Bytes)	Description
Number of Values	7	2	This field indicates the number of values of size DTS of the particular property supported by the device.
SupportedValue1	8	DTS	A particular vendor's device shall support this value of the property.
SupportedValue2	9	DTS	A particular vendor's device shall support this value of the property.
SupportedValue3	10	DTS	A particular vendor's device shall support this value of the property.
-	-	-	-
SupportedValueM	Special	DTS	A particular vendor's device shall support this value of the property.

13.3.4 DevicePropCode Format

All DevicePropCodes shall take the form of a 16-bit integer, are referred to using hexadecimal notation, have bits 12 and 14 set to 1, and bit 13 set to zero. All non-defined DevicePropCodes having bit 15 set to zero are reserved for future use. If a proprietary implementation wishes to define a proprietary DevicePropCode, bit 15 should be set to 1 as well.

13.3.5 DevicePropCode Summary

The following properties are defined in this Standard:

Table 26: DevicePropCode Summary

DevicePropCode	Name
0x5000	Undefined
0x5001	BatteryLevel
0x5002	FunctionalMode
0x5003	ImageSize
0x5004	CompressionSetting
0x5005	WhiteBalance
0x5006	RGB Gain
0x5007	F-Number
0x5008	FocalLength
0x5009	FocusDistance
0x500A	FocusMode
0x500B	ExposureMeteringMode
0x500C	FlashMode
0x500D	ExposureTime
0x500E	ExposureProgramMode
0x500F	ExposureIndex
0x5010	ExposureBiasCompensation
0x5011	DateTime
0x5012	CaptureDelay
0x5013	StillCaptureMode
0x5014	Contrast
0x5015	Sharpness
0x5016	DigitalZoom
0x5017	EffectMode
0x5018	BurstNumber
0x5019	BurstInterval
0x501A	TimelapseNumber
0x501B	TimelapseInterval
0x501C	FocusMeteringMode
0x501D	UploadURL
0x501E	Artist
0x501F	CopyrightInfo
All other codes with MSN of 0101	Reserved
All codes with MSN of 1101	Vendor-Extended Property Code

13.4 Device Property Descriptions

13.4.1 BatteryLevel

DevicePropCode = 0x5001

DataType: UINT8

DescForms: Enum, Range

Get/Set: Get

Description: Battery level is a read-only property typically represented by a range of integers. The minimum field should be set to the integer used for no power (example 0), and the maximum should be set to the integer used for full power (example 100). The step field, or the individual thresholds in an enumerated list, are used to indicate when the device intends to generate a DevicePropChanged event to let the opposing device know a threshold has been reached, and therefore should be conservative (example 10).

The value 0 may be realized in situations where the device has alternate power provided by the transport or some other means.

13.4.2 FunctionalMode

DevicePropCode = 0x5002

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: Allows the functional mode of the device to be controlled. All devices are assumed to default to a "standard mode." Alternate modes are typically used to indicate support for a reduced mode of operation (e.g. sleep state) or an advanced mode or add-on that offers extended capabilities. The definition of non-standard modes is device-dependent. Any change in capability caused by a change in FunctionalMode shall be evident by the DeviceInfoChanged event that is required to be sent by a device if its capabilities can change. This property is described using the Enumeration form of the DevicePropDesc dataset. This property is also exposed outside of sessions in the corresponding field in the DeviceInfo dataset. The allowed values are described in Clause 5.5.1.

13.4.3 ImageSize

DevicePropCode = 0x5003

Data Type: String

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the height and width of the image that will be captured in pixels supported by the device. This property takes the form of a Unicode, null-terminated string that is parsed as follows: "WxH" where the W represents the width and the H represents the height interpreted as unsigned integers. Example: width = 800, height = 600, ImageSize string = "800x600" with a null-terminator on the end. This property may be expressed as an enumerated list of allowed combinations, or if the individual width and height are linearly settable and orthogonal to each other, they may be expressed as a range. For example, for a device that could set width from 1 to 640 and height from 1 to 480, the minimum in the range field would be "1x1" (null-terminated), for a one-pixel image, and the maximum would be "640x480" (null-terminated), for the largest possible image. In this example, the step would be "1x1" (null-terminated), indicating that the width and height are each incrementable to the integer.

Changing this device property often causes fields in StorageInfo datasets to change, such as FreeSpaceInImages. If this occurs, the device is required to issue a StorageInfoChanged event immediately after this property is changed.

13.4.4 CompressionSetting

DevicePropCode = `0x5004`

Data Type: UINT8

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: Compression setting is a property intended to be as close as is possible to being linear with respect to perceived image quality over a broad range of scene content, and is represented by either a range or an enumeration of integers. Low integers are used to represent low quality (i.e. maximum compression) while high integers are used to represent high quality (i.e. minimum compression). No attempt is made in this standard to assign specific values of this property with any absolute benchmark, so any available settings on a device are relative to that device only and are therefore device-specific.

13.4.5 WhiteBalance

DevicePropCode = `0x5005`

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property is used to set how the device weights color channels. The device enumerates its supported values for this property.

Table 27: White Balance Settings

Value	Setting
0x0000	Undefined
0x0001	Manual
0x0002	Automatic
0x0003	One-push Automatic
0x0004	Daylight
0x0005	Florescent
0x0006	Tungsten
0x0007	Flash
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

Manual: The white balance is set directly using the RGB Gain property, described in Clause 13.4.6, and is static until changed.

Automatic: The device attempts to set the white balance using some kind of automatic mechanism.

One-push Automatic: The user must press the capture button while pointing the device at a white field, at which time the device determines the white balance setting.

Daylight: The device attempts to set the white balance to a value that is appropriate for use in daylight conditions.

Tungsten: The device attempts to set the white balance to a value that is appropriate for use in conditions with a tungsten light source.

Flash: The device attempts to set the white balance to a value that is appropriate for flash conditions.

13.4.6 RGB Gain

DevicePropCode = 0x5006

Data Type: String

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property takes the form of a Unicode, null-terminated string that is parsed as follows: “R:G:B” where the R represents the red gain, the G represents the green gain, and the B represents the blue gain. For example, for an RGB ratio of (red=4, green=2, blue=3), RGB string could be “4:2:3” (null-terminated) or “2000:1000:1500” (null-terminated). The string parser for this property value should be able to support up to UINT16 integers for R,

G, and B. These values are relative to each other, and therefore may take on any integer value. This property may be supported as an enumerated list of settings, or using a range. The minimum value would represent the smallest numerical value (typically "1:1:1" null terminated). Using values of zero for a particular color channel would mean that color channel would be dropped, so a value of "0:0:0" would result in images with all pixel values being equal to zero. The maximum value would represent the largest value each field may be set to (up to "65535:65535:65535" null-terminated), effectively determining the setting's granularity by an order of magnitude per significant digit. The step value is typically "1:1:1". If a particular implementation desires the capability to enforce minimum and/or maximum ratios, the green channel may be forced to a fixed value. An example of this would be a minimum field of "1:1000:1", a maximum field of "20000:1000:20000" and a step field of "1:0:1".

13.4.7 FNumber

DevicePropCode = `0x5007`

DataType: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property corresponds to the aperture of the lens. The units are equal to the F-number scaled by 100. When the device is in an automatic Exposure Program Mode, the setting of this property via the SetDeviceProp operation may cause other properties such as Exposure Time and Exposure Index to change. Like all device properties that cause other device properties to change, the device is required to issue DevicePropChanged events for the other device properties that changed as a side effect of the invoked change. The setting of this property is typically only valid when the device has an ExposureProgramMode setting of Manual or Aperture Priority.

13.4.8 FocalLength

DevicePropCode = `0x5008`

Data Type: UINT32

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property represents the 35mm equivalent focal length. The values of this property correspond to the focal length in millimeters multiplied by 100.

13.4.9 FocusDistance

DevicePropCode = 0x5009

Data Type: UINT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: The values of this property are unsigned integers with the values corresponding to millimeters. A value of 0xFFFF corresponds to a setting greater than 655 meters.

13.4.10 FocusMode

DevicePropCode = 0x500A

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: The device enumerates the supported values of this property. The following values are defined:

Table 28: FocusMode Settings

Value	Description
0x0000	Undefined
0x0001	Manual
0x0002	Automatic
0x0003	Automatic Macro (close-up)
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.11 ExposureMeteringMode

DevicePropCode = 0x500B

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: The device enumerates the supported values of this property. The following values are defined:

Table 29: ExposureMeteringMode Settings

Value	Description
0x0000	Undefined
0x0001	Average
0x0002	Center-weighted-average
0x0003	Multi-spot
0x0004	Center-spot
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.12 FlashMode

DevicePropCode = 0x500C

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: The device enumerates the supported values of this property. The following values are defined:

Table 30: FlashMode Settings

Value	Description
0x0000	Undefined
0x0001	auto flash
0x0002	Flash off
0x0003	Fill flash
0x0004	Red eye auto
0x0005	Red eye fill
0x0006	External Sync
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.13 ExposureTime

DevicePropCode = 0x500D

Data Type: UINT32

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property corresponds to the shutter speed. It has units of seconds scaled by 10,000. When the device is in an automatic Exposure Program Mode, the setting of this property via SetDeviceProp may cause other properties to change. Like all properties that cause other properties to change, the device is required to issue DevicePropChanged events for the other properties that changed as the result of the initial change. This property is typically only used by the device when the ProgramExposureMode is set to Manual or Shutter Priority.

13.4.14 ExposureProgramMode

DevicePropCode = `0x500E`

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property allows the exposure program mode settings of the device, corresponding to the "Exposure Program" tag within an EXIF or a TIFF/EP image file, to be constrained by a list of allowed exposure program mode settings supported by the device. The following values are defined:

Table 31: ExposureProgramMode Settings

Value	Description
0x0000	Undefined
0x0001	Manual
0x0002	Automatic
0x0003	Aperture Priority
0x0004	Shutter Priority
0x0005	Program Creative (greater depth of field)
0x0006	Program Action (faster shutter speed)
0x0007	Portrait
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.15 ExposureIndex

DevicePropCode = 0x500F

Data Type: UINT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property allows for the emulation of film speed settings on a Digital Camera. The settings correspond to the ISO designations (ASA/DIN). Typically, a device supports discrete enumerated values but continuous control over a range is possible. A value of 0xFFFF corresponds to Automatic ISO setting.

13.4.16 ExposureBiasCompensation

DevicePropCode = 0x5010

Data Type: INT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property allows for the adjustment of the set point of the digital camera's auto exposure control. For example, a setting of 0 will not change the factory set auto exposure level. The units are in "stops" scaled by a factor of 1000, in order to allow for fractional stop values. A setting of 2000 corresponds to 2 stops more exposure (4X more energy on the sensor) yielding brighter images. A setting of -1000 corresponds to one stop less exposure ($^1/_2$x the energy on the sensor) yielding darker images. The setting values are in APEX units (Additive system of Photographic Exposure). This property may be expressed as an enumerated list or as a range. This property is typically only used when the device has an ExposureProgramMode of Manual.

13.4.17 DateTime

DevicePropCode = 0x5011

Data Type: String

DescForms: None

Get/Set: Get, Get/Set

Description: This property allows the current device date/time to be read and set. Date and time are represented in ISO standard format as described in ISO 8601 from the most significant number to the least significant number. This shall take the form of a Unicode string in the format "YYYYMMDDThhmmss.s" where YYYY is the year, MM is the month 01-12, DD is the day of the month 01-31, T is a constant character, hh is the hours since midnight 00-23, mm is the minutes 00-59 past the hour, and ss.s is the seconds past the minute, with the ".s" being optional tenths of a second past the second. This string

can optionally be appended with Z to indicate UTC, or +/-hhmm to indicate the time is relative to a time zone. Appending neither indicates the time zone is unknown.

This property does not need to use a range or an enumeration, as the possible allowed time values are implicitly specified by the definition of standard time and the format given in this and the ISO 8601 specifications.

13.4.18 CaptureDelay

DevicePropCode = 0x5012

Data Type: UINT32

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This value describes the amount of time delay that should be inserted between the capture trigger and the actual initiation of the data capture. This value shall be interpreted as milliseconds. This property is not intended to be used to describe the time between frames for single-initiation multiple captures such as burst or time-lapse, which have separate interval properties outlined in Clauses 13.4.25 and 13.4.27. In those cases it would still serve as an initial delay before the first image in the series was captured, independent of the time between frames. For no pre-capture delay, this property should be set to zero.

13.4.19 StillCaptureMode

DevicePropCode = 0x5013

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property allows for the specification of the type of still capture that is performed upon a still capture initiation, according to the following table:

Table 32: StillCaptureMode Settings

Value	Description
0x0000	Undefined
0x0001	Normal
0x0002	Burst
0x0003	Timelapse
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.20 Contrast

DevicePropCode = 0x5014

Data Type: UINT8

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the perceived contrast of captured images. This property may use an enumeration or range. The minimum supported value is used to represent the least contrast, while the maximum value represents the most contrast. Typically a value in the middle of the range would represent normal (default) contrast.

13.4.21 Sharpness

DevicePropCode = 0x5015

Data Type: UINT8

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the perceived sharpness of captured images. This property may use an enumeration or range. The minimum value is used to represent the least amount of sharpness, while the maximum value represents maximum sharpness. Typically a value in the middle of the range would represent normal (default) sharpness.

13.4.22 DigitalZoom

DevicePropCode = 0x5016

Data Type: UINT8

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the effective zoom ratio of digital camera's acquired image scaled by a factor of 10. No digital zoom (1X) corresponds to a value of 10, which is the standard scene size captured by the camera. A value of 20 corresponds to a 2X zoom where $^{1}/_{4}$ of the standard scene size is captured by the camera. This property may be represented by an enumeration or a range. The minimum value should represent the minimum digital zoom (typically 10), while the maximum value should represent the maximum digital zoom that the device allows.

13.4.23 EffectMode

DevicePropCode = 0x5017

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property addresses special image acquisition modes of the camera. The following values are defined:

Table 33: EffectMode Setting

Value	Description
0x0000	Undefined
0x0001	Standard (color)
0x0002	Black & White
0x0003	Sepia
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.24 BurstNumber

DevicePropCode = 0x5018

Data Type: UINT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the number of images that the device will attempt to capture upon initiation of a burst operation.

13.4.25 BurstInterval

DevicePropCode = 0x5019

Data Type: UINT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the time delay between captures upon initiation of a burst operation. This value is expressed in whole milliseconds.

13.4.26 TimelapseNumber

DevicePropCode = 0x501A

Data Type: UINT16

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the number of images that the device will attempt to capture upon initiation of a time-lapse capture.

13.4.27 TimelapseInterval

DevicePropCode = 0x501B

Data Type: UINT32

DescForms: Enum, Range

Get/Set: Get, Get/Set

Description: This property controls the time delay between captures upon initiation of a time-lapse capture operation. This value is expressed in milliseconds.

13.4.28 FocusMeteringMode

DevicePropCode = 0x501C

Data Type: UINT16

DescForms: Enum

Get/Set: Get, Get/Set

Description: This property controls which automatic focus mechanism is used by the device. The device enumerates the supported values of this property. The following values are defined:

Table 34: FocusMeteringMode Settings

Value	Description
0x0000	Undefined
0x0001	Center-spot
0x0002	Multi-spot
All other values with Bit 15 set to zero	Reserved
All values with Bit 15 set to 1	Vendor-Defined

13.4.29 UploadURL

DevicePropCode = 0x501D

Data Type: String

DescForms: None

Get/Set: Get, Get/Set

Description: This property is used to describe a standard Internet URL (Universal Resource Locator) that the receiving device may use to upload images or objects to once they are acquired from the device.

13.4.30 Artist

DevicePropCode = 0x501E

Data Type: String

DescForms: None

Get/Set: Get, Get/Set

Description: This property is used to contain the name of the owner/user of the device. This property is intended for use by the device to populate the Artist field in any EXIF images that are captured with the device.

13.4.31 Copyright

DevicePropCode = 0x501F

Data Type: String

DescForms: None

Get/Set: Get, Get/Set

Description: This property is used to contain the copyright notification. This property is intended for use by the device to populate the Copyright field in any EXIF images that are captured with the device.

14 Conformance Section

To determine if the Responder supports Push mode, the Initiator should look for the SendObject OperationCode in the device's DeviceInfo dataset. Presence of the GetObject operation indicates Pull mode is supported.

An Initiator shall be able to send operations, receive responses and events, and poll for in-band events if necessary for that transport. A Responder shall be able to respond to all operations that it reports in its DeviceInfo dataset.

All devices shall implement the following operations: GetDeviceInfo, OpenSession, and CloseSession.

A Responder shall support either GetObject and GetObjectInfo, or SendObject and SendObjectInfo, or all four operations.

If the Responder supports GetObject, it must also support GetNumImages as well as GetObjectHandles, GetObjectInfo, and GetThumb.

Table 35: Operation Implementation Conformance

OperationCode	Operation Name	Mandatory	Pull	Push
0x1001	GetDeviceInfo	X	X	X
0x1002	OpenSession	X	X	X
0x1003	CloseSession	X	X	X
0x1004	GetStorageIDs	X	X	X
0x1005	GetStorageInfo	X	X	X
0x1006	GetNumObjects		X	
0x1007	GetObjectHandles		X	
0x1008	GetObjectInfo		X	
0x1009	GetObject		X	
0x100A	GetThumb		X	
0x100B	DeleteObject			
0x100C	SendObjectInfo			X
0x100D	SendObject			X
0x100E	InitiateCapture			
0x100F	FormatStore			
0x1010	ResetDevice			
0x1011	SelfTest			
0x1012	SetObjectProtection			
0x1013	PowerDown			
0x1014	GetDevicePropDesc			
0x1015	GetDevicePropValue			
0x1016	SetDevicePropValue			
0x1017	ResetDevicePropValue			
0x1018	TerminateOpenCapture			
0x1019	MoveObject			
0x101A	CopyObject			
0x101B	GetPartialObject			
0x101C	InitiateOpenCapture			

A device shall be capable of generating and/or responding to all events as indicated in the following table. A Responder may not need to issue many events if it disables manual or external object creation and device property manipulation while a session is open. While a session is open, a Responder must send an event whenever an image, ancillary data, or store is added or removed and when any device property or the DeviceInfo capabilities are modified.

Table 36: Event Implementation Conformance

EventCode	Event Name	Required?
0x4001	CancelTransaction	Only if transport implementation specification specifies using this formal event for canceling transactions
0x4002	ObjectAdded	Only if objects can be added from an external source during session
0x4003	ObjectRemoved	Only if objects can be removed by external sources during session
0x4004	StoreAdded	Only if device has stores that may become available/unavailable during a session
0x4005	StoreRemoved	Only if device has stores that may become available/unavailable during a session
0x4006	DevicePropChanged	Only if device properties are supported and some properties are inter-dependent or can be changed due to an external or indirect source
0x4007	ObjectInfoChanged	Only if it possible for the objects' ObjectInfo to change during a session
0x4008	DeviceInfoChanged	Only if device supports changes in functionality during an open session such as sleep modes or functional modes
0x4009	RequestObjectTransfer	Only if this functionality is desired
0x400A	StoreFull	Only if objects can be sent to the device or it supports InitiateCapture or InitiateOpenCapture
0x400B	DeviceReset	Only if it is possible for the device to be reset by something other than the Initiator
0x400C	StorageInfoChanged	Only if it is possible for one of the fields in StorageInfo to change (e.g. FreeSpaceInImages due to an ImageSize DeviceProperty change)
0x400D	UnreportedStatus	Only if the transport implementation supports suspend without self-resume capability and other events may occur during a suspend period

Annex A: Goals of this Standard

A.1 Transport independence

The requirements for communicating with digital still photography devices do not change between transports. The needs to successfully fulfill certain user-scenarios can successfully be abstracted with operations like GetObject that may be implemented in transport-specific ways for maximum efficiency. This protocol is intended to work on a wide variety of bus protocols, including IR, USB, RF, and 1394. The transport protocol needs to provide some basic services such as device discovery, enumeration, and reliable data delivery. For buses that do not provide all of these services (such as RS-232C), a future annex to this standard may contain additional transport-specific extensions. The transport also defines the start and end of a "session" by identifying when a connection between two devices is established and when it is broken. Although the standard should not limit itself to the lowest common denominator transport, it should recognize limitations on the transports and try to work within them.

A.2 Extensibility

The standard must be extensible both by the standard committee as well as by an individual hardware manufacturer. The extensions must be able to pass seamlessly through the driver layer so that an application can deal with them directly. Extensions include new operations, new device properties, and new file types.

A.3 Simplicity

The protocol must be easy to understand and implement. It will also assume only a minimum of processing capability for some devices, while still being able to scale up to more powerful devices.

A.4 Robustness

Although the transport is providing a reliable connection, faulty software or hardware can still corrupt a connection. At any point during a transaction, the devices on each end need to be able to recover gracefully from errors.

Annex B: Filesystem Implementation Examples

(informative)

This clause shows examples of how to implement support for filesystems.

B.1 ObjectHandle Assignment

Regardless of the filesystem used for a particular storage device, files stored in the file system may be assigned any handle. The simplest method for ObjectHandle creation is to traverse the file system hierarchy in a breadth-first or depth-first fashion assigning ObjectHandles to each directory and file using consecutively numbered values starting with 0x00000001.

Objects may be created to represent associations that do not exist as folders in the file system, according to the convention used by the particular device. For example, this type of association might be determined from the naming convention used for the individual files that are part of the association. An example would be a store that contained only three files that were part of a burst association as indicated by the filenames burst001.jpg, burst002.jpg, and burst003.jpg. In this case, four ObjectHandles would be assigned; one for each of the images in the association (e.g. 0x00000001, 0x00000002, 0x00000003), and an extra handle to represent the virtual association that is the burst relationship (e.g. 0x00000004). The ObjectInfo datasets returned for each of the image objects should contain a ParentObject field with a value equal to the ObjectHandle of the virtual folder (i.e. 0x00000004). No GetObject would be necessary for the object with handle 0x00000004 because associations may be fully described and reproduced in the most appropriate form by examining the ObjectInfo dataset of the association.

B.2 DCF Filesystem Association Example

Associations may be used to represent filesystems. The following is an example showing a typical filsystem on a DSPD that conforms with DCF. This example shows examples of how a DPOF file would be handled, as well as generic folder associations, a burst association, and ancillary data associations, including both an audio file associated with a single EXIF, as well as a text file associated with an entire burst sequence.

DCF Filesystem Directory Example:

```
\MISC\AUTPRINT.MRK                  // DPOF text file
\DCIM\100MODEL\                     // DSPD-model-specific folder
\DCIM\100MODEL\DCP_0001.JPG         // EXIF Image #1
\DCIM\100MODEL\DCP_0002.JPG         // EXIF image #2
\DCIM\100MODEL\DCP_0002.WAV         // WAVE file assoc'ed with EXIF #2
\DCIM\100MODEL\B01_0003.JPG         // Burst Set #1 (1/3) EXIF #3
\DCIM\100MODEL\B01_0004.JPG         // Burst Set #1 (2/3) EXIF #4
\DCIM\100MODEL\B01_0005.JPG         // Burst Set #1 (3/3) EXIF #5
\DCIM\100MODEL\B01_0003.TXT         // Text file associated w/ burst seq
```

Figure B.1: DCF Filesystem Example

Object Handle	Object Description	ObjFormat Code	Parent Object	Seq Number
0x00000001	\MISC folder	0x3001	0x00000000	0x00000000
0x00000002	\MISC\AUTPRINT.MRK	0x3006	0x00000001	0x00000000
0x00000003	\DCIM folder	0x3001	0x00000000	0x00000000
0x00000004	\DCIM\100MODEL folder	0x3001	0x00000003	0x00000000
0x00000005	\DCIM\100MODEL\DCP_0001.JPG	0x3801	0x00000004	0x00000000
0x00000006	Ancillary Data Assoc for EXIF#2/WAV	0x3001	0x00000004	0x00000000
0x00000007	\DCIM\100MODEL\DCP_0002.JPG	0x3801	0x00000006	0x00000000
0x00000008	\DCIM\100MODEL\DCP_0002.WAV	0x3008	0x00000006	0x00000000
0x00000009	Burst Association for Burst Seq #1	0x3001	0x00000004	0x00000000
0x0000000A	\DCIM\100MODEL\B01_0003.JPG	0x3801	0x00000009	0x00000001
0x0000000B	\DCIM\100MODEL\B01_0004.JPG	0x3801	0x00000009	0x00000002
0x0000000C	\DCIM\100MODEL\B01_0005.JPG	0x3801	0x00000009	0x00000003
0x0000000D	\DCIM\100MODEL\B01_0003.TXT	0x3004	0x00000009	0x00000000

Annex C: Optional Device Features

(informative)

This clause describes the features that may be implemented at the vendor's discretion. These features are not required for a device to conform to this standard.

C.1 Open Write Store

Devices are not required to possess the ability to write images or other data directly to its internal storage areas via an external data transfer. Devices that are read-only may still fully conform to this standard. Devices might only be able to record images via a manual press of the shutter button or as a response to an InitiateCapture or InitiateOpenCapture operation.

C.2 Hierarchical Storage System

Devices are not required to possess anything other than a flat-file system for storing image data, but should be able to take advantage of a hierarchical filesystem if it is present.

C.3 Multisession Capability

Although devices are not required to support more than one simultaneous physical or logical session, they are not explicitly prohibited from doing so by this standard. A particular device may effectively deny concurrent service. The current version of this standard provides no method for avoiding the pitfalls of data corruption caused by multisession operation.

C.4 Out-of-Session Image Handle Persistence

According to Clause 8.2.1, devices must keep assigned image handles persistent within a particular session. However, devices may optionally wish to extend this persistence in the following ways:

A. Powered Handle Persistence

Devices that exhibit powered handle persistence retain the same ObjectHandles across all connection sessions while the device is actively powered, only re-enumerating image handles on power-up or when specifically requested to do so.

B. Permanent Handle Persistence

Devices that exhibit permanent handle persistence retain the same ObjectHandles permanently, across all connection sessions and power-on sessions, only re-enumerating image handles when specifically requested to do so.

C.5 Multiple Image Formats

Devices shall support image format as described in Clause 6. Devices may or may not support multiple image formats, bit depths, heights, widths, and aspect ratios.

C.6 Multiple Thumbnail Formats

Devices shall support thumbnail formats as described in Clause 6.1. Devices may or may not provide multiple thumbnail formats, bit depths, heights, widths, and aspect ratios.

C.7 Write-Protection Mechanism

Devices may optionally possess a write-protection mechanism for the images that they contain, but they are not required to do so.

C.8 Sub-Image Transfers

Devices may optionally transfer images subsampled on the fly to a lower resolution, or may allow specification by subsection. These features are not supported by the standard GetObject operation, but could be enabled by using a vendor extended operation.

C.9 Non-standard Functional Modes

Devices may optionally take on non-standard modes of operation such as sleep modes, or advanced functionality modes. A change in mode either requires all sessions to be closed, or requires support for the DeviceInfoChanged event, to indicate the changes in supported capabilities while in the new mode.

Annex D: USB Implementation of PIMA15740

D.1 Introduction

D.1.1 Purpose

This annex describes how devices supporting the Universal Serial Bus shall implement PIMA15740 in a platform-independent manner.

D.1.2 Terms and Abbreviations

Table D.1: USB Terms and Abbreviations

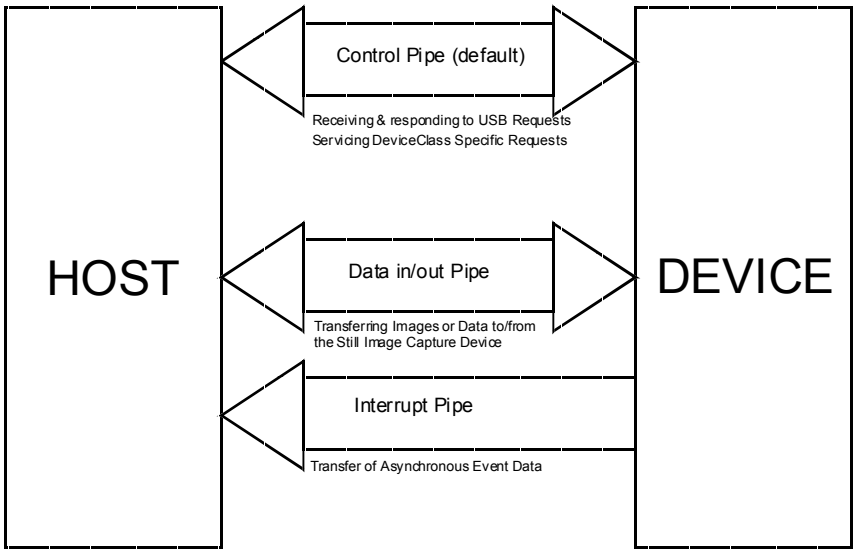
Term	Description
Configuration	A collection of one or more interfaces that may be selected on a USB device
Descriptor	Data structure used to describe a USB device capability or characteristic
Device	A USB peripheral
Driver	Host software that connects other drivers, DLLs or applications to USBDI.
Endpoint	Source or sink of data on a USB device
HCD	Acronym for Host Controller Driver, the Driver used to manage a host controller
HCDI	Acronym for HCD Interface, the programming interface used by USBD to interact with HCD
Host	A computer system where a Host Controller is installed
Host Controller	Hardware that connects a Host to USB
Host Software	Generic term for a collection of drivers, DLLs and/or applications that provide operating system support for a Device
IHV	Acronym for Independent Hardware Vendor
Interface	Collection of zero or more endpoints that present functionality to a host
OHCI	Acronym for Open Host Controller Interface, a hardware register specification defined by Compaq and Microsoft for a Host Controller
UHCI	Acronym for Universal Host Controller Interface, a hardware register specification defined by Intel for a Host Controller
USB	Acronym for Universal Serial Bus, a bus used to connect devices to a host
USBD	Acronym for Universal Serial Bus Driver, the Driver used to manage and use Devices among multiple Device Drivers
USBDI	Acronym for USBD Interface, the USBD programming interface

D.2 Overview

USB Still Image Capture Devices also use the bulk pipe to adjust device properties (i.e. controls). Specific PIMA15740 operations, GetDevicePropDesc, GetDevicePropValue, and SetDeviceProp, are used to manage the controls and mode settings of the device.

When an asynchronous event occurs in the device, such as a low battery indication or the removal of a memory card, the asynchronous event is reported over an interrupt pipe.

Figure D.1: Device Configuration



D.3 Assumptions and Constraints

D.3.1 Compliance

This section describes assumptions and constraints related to compliance.

D.3.1.1 USB Specification

The device shall be compliant with the USB Specification.

D.3.1.2 Device Framework

The device shall support standard and device-specific requests as described in the USB Specification.

D.3.2 Functional Overview

D.3.2.1 Image Data

Image data from the device is delivered to the host system through a bulk pipe. The format of the data is identified by an ObjectFormat data item in the ObjectInfo Dataset.

D.3.2.2 Device Controls and Status

Device controls and operational modes are managed by the GetDevicePropDesc, GetDevicePropValue, and SetDeviceProp operations defined in PIMA15740.

D.4 Device Characteristics

D.4.1 Configuration

The device configuration shall support at least one interface for image and data transfer. The device class code, device sub-class code, and the device protocol code in the Device Descriptor shall all be set to zero.

D.4.2 Interface

As noted above, the Still Image Capture Device shall support at least one interface for image and data transfer. As with all USB devices, the default endpoint shall be included in all interfaces by implication.

The Still Image Capture Device Interface Descriptor shall use a class code of Image D, a sub-class code of Still Image Capture Device D and a protocol code of D to identify that the device uses protocol defined in this class specification.

D.4.3 Endpoints

The device shall contain at least four endpoints: default, Data-In, Data-Out, and an Interrupt endpoint.

D.4.3.1 Default

The default endpoint shall use control transfers as defined in the USB Specification. The default endpoint shall be used to send standard, class and vendor-specific requests to the device, an interface or an endpoint. The endpoint number must be zero (0x00).

D.4.3.2 Data-In

The Data-In endpoint shall be used to receive image and non-image data (such as a script) from the device intended for delivery to an imaging application on the host. The Data-In endpoint shall use bulk transfers. The endpoint number may be any value between one (1) and fifteen (15) that is not used by another endpoint on the device. The direction shall be IN. The maximum packet size is implementation specific and may vary for different alternate settings.

D.4.3.3 Data-Out

The Data-Out endpoint shall be used to send image and non-image data from the host to the device. The Data-Out endpoint shall use bulk transfers. The endpoint number may be any value between (1) and fifteen (15) that is not used by another endpoint on the device. The direction shall be OUT. The maximum packet size is implementation specific and may vary for different alternate settings.

D.4.3.4 Interrupt

The Interrupt endpoint associated with the Still Image Interface shall be used to send event data to the host from the device. The Interrupt endpoint shall use interrupt transfers. The endpoint number may be any value between one (1) and fifteen (15) that is not used by another endpoint on the device. The direction shall be IN. The maximum packet size is implementation specific.

D.4.4 Data Characteristics

The USB still image capture device may support one or more image data formats. The data formats are identified in the PIMA 15740 DeviceInfo Dataset.

D.4.5 PIMA15740 Event Handling

USB Suspend and Resume signaling will affect the way a Still Image Capture Device that supports PIMA 15740 will handle PIMA15740 Events. Likewise the handling of events will be affected if the device supports the Remote Wakeup feature and whether or not the Remote Wakeup feature is disabled.

PIMA Events shall be handled as described by the cases that follow:

Case 1. USB not suspended, Normal PIMA 15740 Event handling

1. PIMA Events are reported to the host via an interrupt where the data format is described in Clause 7.3.1 of this document.

Case 2. USB Suspended, the Remote Wakeup feature is enabled

1. Device signals remote wakeup upon PIMA15740 Event detection
2. Device issues PIMA Events when bus signaling has resumed.

Case 3. USB Suspended, the Remote Wakeup feature is disabled, and no PIMA 15740 Events occur

1. When host or upstream hubs resume, the device continues, the PIMA15470 session is undisturbed.

Case 4. USB Suspended, the Remote Wakeup feature is disabled, and PIMA15740 events do occur

1. When host or upstream hubs resume, the device then posts the PIMA15740 "UnreportedStatus" Event. The PIMA15740 session remains open.
2. The host (Initiator) must examine the device. This includes checking the DeviceInfo dataset, checking object handles and ObjectInfo datasets, and checking the device status. The host (Initiator) may optionally close the session and restart.

The host may disable the remote wakeup feature at any time. However if a session is open, the awkward "UnreportedStatus" Event might occur.

This approach does not require the device to queue events, avoiding memory depth problems. After a suspension there will be at most one event.

D.5 Device Requests

D.5.1 Standard Requests

The Device shall support the standard USB device requests as described below. The device shall return STALL if any unrecognized or unsupported standard request is received.

D.5.1.1 Clear Feature

The Device shall return STALL for any unrecognized or unsupported Get Feature request.

D.5.1.2 Get Configuration

The Device shall support the Get Configuration request. The Device shall return zero if the device is unconfigured or the bConfiguration value is undefined in the Configuration Descriptor.

D.5.1.3 Get Descriptor

The Device shall support Get Descriptor requests for standard descriptors (Device, Configuration and String). The Device may support Get Descriptor requests for Class or Vendor-specific descriptors. The Device shall return STALL if a Get Descriptor request is made for an unrecognized or unsupported descriptor.

D.5.1.4 Get Interface

The Device shall support a Get Interface request for Interface 0 when configured by returning an Alternate Setting of zero. The Device shall return STALL for a Get Interface request for any other Interface or any Get Interface request before the Device is configured.

D.5.1.5 Get Status

The Device shall support a Get Status request directed at the device, Interface 0 or any defined endpoint (default, Data-In, or Data-Out). The Device shall return STALL if a Get Status request is received for Interface 0 or any defined Endpoint before the Device is configured. The Device shall return STALL if a Get Status request is received for any unrecognized or unsupported recipient.

D.5.1.6 Set Address

The Device shall support a Set Address request to change the Device Address from the default address (zero) to a unique address. The Device may return STALL if any subsequent Set Address request is received to change the Device Address from a non-zero value to any value (including zero).

D.5.1.7 Set Configuration

The Device shall support the Set Configuration request to set the Device Configuration to zero (unconfigured) or the bConfiguration value defined in the Configuration Descriptor. The Device shall return STALL if a Set Configuration request is received with any other value.

D.5.1.8 Set Descriptor

The Device may support Set Descriptor requests for any defined Descriptor (Device, Configuration, Interface, Endpoint, String, Class or Vendor-Specific). The Device shall return STALL if a Descriptor may not be updated, is unrecognized, or is unsupported.

D.5.1.9 Set Feature

The Device shall return STALL for any unrecognized or unsupported Set Feature request.

D.5.1.10 Set Interface

When configured, the Device shall support a Set Interface request to Interface 0 for defined Alternate Settings.

D.5.1.11 Synch Frame

The Device shall return STALL for any Synch Frame request.

D.5.2 Class-Specific Requests

The Device may support class-specific requests. The device shall return STALL if an unrecognized or unsupported device-specific request is received.

D.5.2.1 Cancel Request

The Still Image Capture device shall accept the Cancel Request from the host, which is a control write sequence to the device's control endpoint. The data stage transfers to the device information that identifies the transaction over the Bulk Pipe that was cancelled by the host.

D.5.2.1.1 Cancel Set Up

The host is responsible for establishing the values passed in the fields listed in Table D.2. The setup data packet has eight bytes.

Table D.2: Format of Setup Data for the Cancel Request

Offset	Field	Size	Value	Description
0	bmRequestType	1	bitmap	00100001 Host-to-Device, Class-Specific, Recipient-Interface
1	bRequest	1	code	Cancel_Request (0x64, for this request)
2	wValue	2	value	value equal to zero.
4	wIndex	2	value	value equal to zero.
6	wLength	2	count	Value = 0x0006

D.5.2.1.2 Format of Cancel Data

The data stage of the Cancel Request has the following format.

Table D.3: Format of Cancel Request Data

Offset	Field	Size	Value	Description
0	Cancellation Code	2	code	Value=0x4001, identifier for cancellation
2	TransactionID	4	number	An unsigned 32-bit field containing the PIMA15740 TransactionID

D.5.2.2 Get Extended Event Data

The Still Image Capture device shall accept the Get Extended Event Data Request from the host, which is a control, read sequence from the device's control endpoint. The data stage transfers to the host extended information regarding an asynchronous event or vendor condition.

D.5.2.2.1 Get Event Set Up

Table D.4 defines the 8-byte set up data for the Get_Extended_Event_Data request.

Table D.4: Format of Setup Data to retrieve the Extended Event Data

Offset	Field	Size	Value	Description
0	bmRequestType	1	bitmap	10100001 Device-to-Host, Class-Specific, Recipient-Interface
1	bRequest	1	code	Get_Extended_Event_Data (0x65, code for this request)
2	wValue	2	value	value equal to zero
4	wIndex	2	value	value equal to zero
6	wLength	2	count	Size of the host buffer allocated for the Extended Event Data

D.5.2.2.2 Format of Event Data

The data stage of the Get Event Data Request has the following format.

Table D.5: Data Format of Get Extended Event Data Request

Offset	Field	Size	Value	Description
0	Event Code	2	code	The PIMA15740 Event Code or Vendor Code
2	TransactionID	4	number	An unsigned 32-bit field containing the PIMA15740 TransactionID. This field is 0x00000000 if a TransactionID does not apply to this event.
6	Number of Parameters	2	number	This field contains a number that indicates the number of event parameters associated with the event code
8	Size of Parameter 1	2	number	This field contains a number that indicates the size in bytes of the corresponding event parameter.
10	Parameter 1	??	value	This field contains the actual event parameter. The format and meaning of the parameter is described in the description of the event.
??	Size of Parameter N	2	number	This field contains a number that indicates the size in bytes of the corresponding event parameter.
??	Parameter N	??	value	This field contains the actual event parameter. The format and meaning of the parameter is described in the description of the event.

D.5.2.3 Device Reset Request

The Still Image Capture device shall accept the Device Reset Request from the host, which is a no-data control sequence to the device's control endpoint. The Device Reset Request is used by the host to return the Still Image Capture Device to the Idle state after the Bulk-pipe has stalled. The request may also be used to clear any vendor specified suspend conditions.

D.5.2.3.1 Device Reset Set Up

The host is responsible for establishing the values passed in the fields listed in Table D.6. The setup data packet has eight bytes.

Table D.6: Format of Setup Data for the Device Reset Request

Offset	Field	Size	Value	Description
0	bmRequestType	1	bitmap	00100001 Host-to-Device, Class-Specific, Recipient-Interface
1	bRequest	1	code	Device_Reset_Request (0x66, for this request)
2	wValue	2	value	value equal to zero.
4	wIndex	2	value	value equal to zero.
6	wLength	2	count	Value of 0x0000, there is no data associated with this request

D.5.2.4 Get Device Status Request

The Still Image Capture device shall accept the Get Device Status Request from the host, which is a control, read sequence from the device's control endpoint. The data stage transfers to the host information regarding the status or protocol state of the device. This request is used by the host to retrieve information needed to clear halted endpoints that result from a device initiated data transfer cancellation.

D.5.2.4.1 Get Device Status Set Up

Table D.7 defines the 8-byte set up data for the Get Device Status Request.

Table D.7: Format of Setup Data to retrieve the Extended Event Data

Offset	Field	Size	Value	Description
0	bmRequestType	1	bitmap	10100001 Device-to-Host, Class-Specific, Recipient-Interface
1	bRequest	1	code	Get_Device_Status (0x67, code for this request)
2	wValue	2	value	value equal to zero
4	wIndex	2	value	value equal to zero
6	wLength	2	count	Size of the host buffer allocated for the Extended Event Data

D.5.2.4.2 Format of Device Status Data

The data stage of the Get Event Data Request has the following format.

Table D.8: Data Format of Get Device Status Request

Offset	Field	Size	Value	Description
0	wLength	2	number	This field specifies the total length of the status data
2	Code	2	code	The PIMA15740 Response Code or Vendor Code: 0x2001 Status OK 0x2019 Device Busy 0x201F Transaction Cancelled
4	Parameter 1	??	value	The format and meaning of the parameter depends on the Status Code. For device initiated cancels this parameter contains an endpoint number.
??	Parameter N	??	value	The format and meaning of the parameter depends on the Status Code. For device initiated cancels this parameter contains an endpoint number.

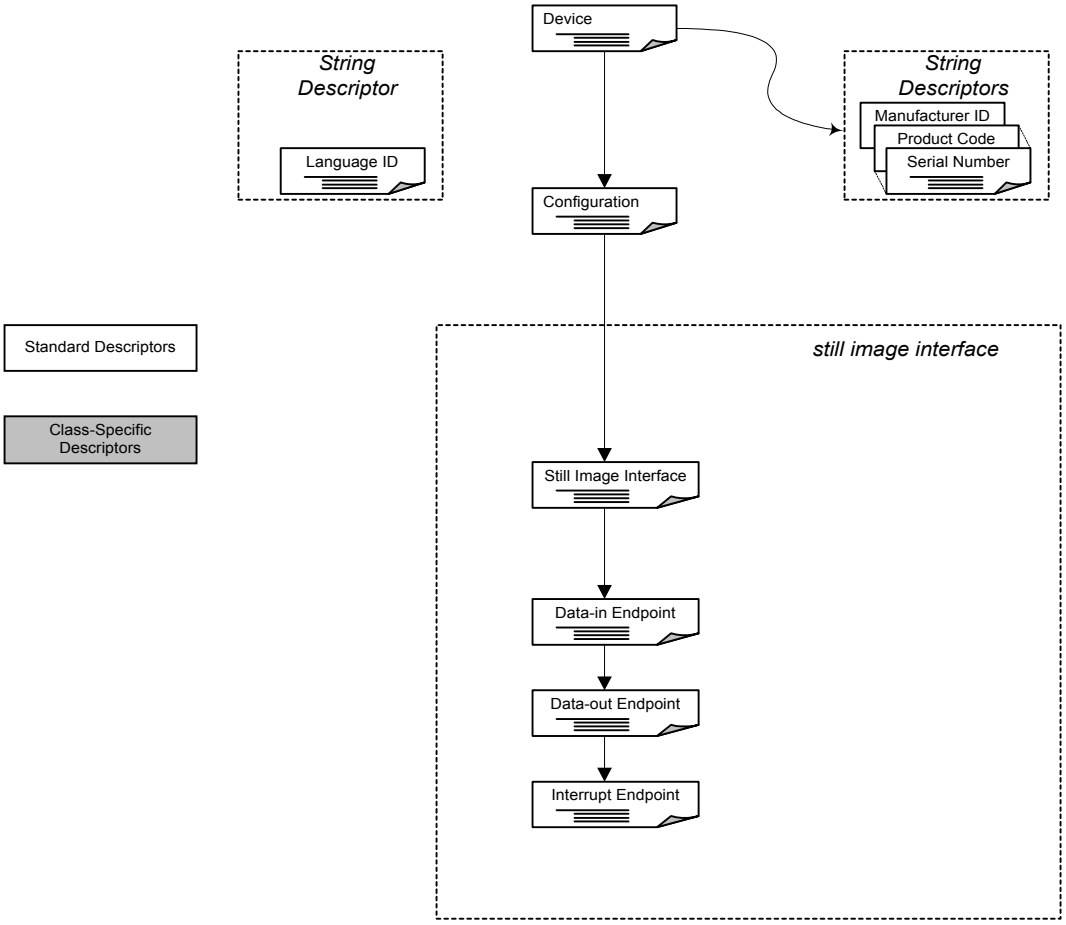
D.5.3 Vendor-Specific Requests

The Device may support vendor-specific requests. The device shall return STALL if an unrecognized or unsupported device-specific request is received.

D.6 Descriptors

The following figure shows the relationship among the descriptors in the Still Image Capture Device Class.

Figure D.2: Descriptor Tree



D.6.1 Standard Descriptors

The Device shall support the standard USB descriptors as described below. The device shall return STALL if a request is received for any unrecognized or unsupported standard descriptor.

D.6.1.1 Device

The device shall return a Device Descriptor with the following values:

Table D.9: Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x12	The length of this descriptor
1	bDescriptorType	1	0x01	Device Descriptor Type
2	bcdUSB	2	0x110	USB Specification Release Number
4	bDeviceClass	1	0x00	Class information may be found at the interface level
5	bDeviceSubClass	1	0x00	bDeviceSubClass is zero
6	bDeviceProtocol	1	0x00	bDeviceProtocol is zero
7	bMaxPacketSize0	1	number	Implementation specific, may be set to 8, 16, 32 or 64
8	idVendor	2	ID	Vendor ID assigned to IHV by USB-IF
10	idProduct	2	ID	Product ID assigned by IHV
12	bcdDevice	2	BCD	Device release number in BCD assigned by IHV to this release of model
14	iManufacturer	1	index	Index of string descriptor describing IHV. If set to zero (0), there is no man ufacturer string.
15	iProduct	1	index	Index of string describing this product. If set to zero (0), there is no product string.
16	iSerialNumber	1	index	Index of string descriptor with this spe cific device's serial number. If set to zero (0), this device does not have a serial number.
17	bNumConfigurations	1	number	Device has this number of configura tions.

D.6.1.2 Configuration Descriptor

The device shall return one or more Configuration Descriptors and other configuration related descriptors as described below:

Table D.10: Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	The length of this descriptor
1	bDescriptorType	1	0x02	Configuration Descriptor Type
2	wTotalLength	2	number	Total length of data returned for this configuration. Includes this descriptor and all of the descriptors that follow (interface, endpoint and vendor spe cific, if present).
4	bNumInterfaces	1	number	This configuration has <#> interfaces
5	bConfigurationVa lue	1	number	Implementation specific value used as an argument to Set Configuration to select this configuration
6	iConfiguration	1	index	Index of string describing this configu ration. If set to zero (0), there is no configuration string.
7	bmAttributes	1	bitmap	Configuration characteristics as defined by USB Specification
8	MaxPower	1	mA	Maximum power draw from the bus by this device when this configuration is selected.

D.6.1.3 Still Image Interface Descriptor

Table D.11: Still Image Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	The length of this descriptor
1	bDescriptorType	1	0x04	Interface Descriptor Type
2	bInterfaceNumber	1	number	Number of interface in configuration
3	bAlternateSetting	1	0x00	Default setting for this interface
4	bNumEndpoints	1	number	Number of endpoints in this interface, not including the default endpoint.
5	bInterfaceClass	1	TBD*	Image interface
6	bInterfaceSubClass	1	TBD*	Still Image Capture Device
7	bInterfaceProtocol	1	TBD*	This field indicates the protocol sup ported by this device. PIMA 15740 compliant devices use Bulk-only protocol
8	iInterface	1	index	Index of string describing this inter face. If set to zero (0), there is no inter face string.

**These fields will be assigned by the USB Working Group prior to the USB Still Image Device Class obtaining version 1.0 status. Until these numbers are assigned, the values used in these fields should be 0x00, and host implementations will need to rely upon external context for driver association.*

D.6.1.4 Data-In Endpoint Descriptor

Table D.12: Data-In Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	7	The length of this descriptor
1	bDescriptorType	1	5	Endpoint Descriptor Type
2	bEndpointAddress	1	0x8?	Any endpoint number not used by another endpoint. The direction shall be set to IN.
3	bmAttributes	1	0x02	The endpoint uses bulk transfers
4	wMaxPacketSize	2	number	Maximum packet size used by this endpoint. This must be less than or equal to 64 bytes.
6	bInterval	1	number	Ignored

D.6.1.5 Data-Out Endpoint Descriptor

Table D.13: Data-Out Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	The length of this descriptor
1	bDescriptorType	1	0x05	Endpoint Descriptor Type
2	bEndpointAddress	1	0x0?	Any endpoint number not used by another endpoint. The direction shall be set to OUT.
3	bmAttributes	1	0x02	The endpoint uses bulk transfers
4	wMaxPacketSize	2	number	Maximum packet size used by this endpoint. This must be less than or equal to 64 bytes.
6	bInterval	1	number	Ignored

D.6.1.6 Interrupt Endpoint Descriptor

Table D.14: Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	The length of this descriptor
1	bDescriptorType	1	0x05	Endpoint Descriptor Type
2	bEndpointAddress	1	0x8?	Any endpoint number not used by another endpoint. The direction shall be set to IN.
3	bmAttributes		0x03	The endpoint uses interrupt transfers
4	wMaxPacketSize	2	number	Maximum packet size used by this endpoint. This must be less than or equal to 64 bytes.
6	bInterval	1	number	Interval for polling endpoint for data transfers

D.6.1.7 String Descriptors

The device may include strings describing the manufacturer, product, serial number, configuration and interface. All string descriptors use the following format:

The Still Image Capture Device uses the following string descriptors to support the descriptor of a Still Image Capture Device.

D.6.1.7.1 Manufacturer ID Code Descriptor

The string descriptor of Index=*iManufacturer* shall support up to 120 UNICODE characters which describe the manufacturer's code name. Each character in the UNICODE string shall be alphanumeric and printable.

Table D.15: Manufacturer ID Code Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0xF2	The length of this descriptor
1	bDescriptorType	1	3	String Descriptor Type
2	wString1	2	uchar	Man ID Code UNICODE character 1
4	wString2	2	uchar	Man ID Code UNICODE character 2
240	wString120	2	uchar	Man ID Code UNICODE character 120

D.6.1.7.2 Product ID Code Descriptor

The string descriptor of Index=*iProduct* shall support up to 122 UNICODE characters that describe the Model name.

Table D.16: Product ID Code Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0xF2	The length of this descriptor
1	bDescriptorType	1	3	String Descriptor Type
2	wString1	2	uchar	Model UNICODE character 1
4	wString2	2	uchar	Model UNICODE character 2
122	wString120	2	uchar	Model UNICODE character 120

D.6.1.7.3 Serial Number Descriptor

The string descriptor of Index=*iSerialNumber* shall support up to 126 UNICODE characters which encode the serial number in a vendor specific format. Each character in the in the UNICODE string shall be alphanumeric and printable.

Table D.17: Serial Number Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0xFE	The length of this descriptor
1	bDescriptorType	1	3	String Descriptor Type
2	wString1	2	uchar	Serial Num UNICODE character 1
4	wString2	2	uchar	Serial Num UNICODE character 2
252	wString126	2	uchar	Serial Num UNICODE character 126

D.6.1.7.4 Language ID Descriptor

The string descriptor of Index=0, a Language ID Descriptor with a valid Language ID code, is mandatory for PIMA15740 compatible devices. String Descriptors are used to describe information required by PIMA15740.

Table D.18: Language ID Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0xFE	The length of this descriptor
1	bDescriptorType	1	3	String Descriptor Type
2	wLangID[0]	2	code	Language ID code 0
N+2	wLangID[N]	2	code	Language ID code N

D.6.1.7.5 Vendor Information Descriptor

The string descriptor of Index=*iVendorInformation* shall be used support PIMA15740 Vendor Dependent Information. A vendor may use up to 126 UNICODE characters in a vendor specific format. Typically, this field is used for firmware version information. Each of the UNICODE characters shall be alpha numeric and printable.

Table D.19: Vendor Information Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0xFE	The length of this descriptor
1	bDescriptorType	1	3	String Descriptor Type
2	wString1	2	uchar	Vendor UNICODE character 1
4	wString2	2	uchar	Vendor UNICODE character 2
252	wString126	2	uchar	Vendor UNICODE character 126

D.6.2 Class-Specific Descriptors

There are no Class-Specific Descriptors defined for a Still Image Capture Device.

D.6.3 Vendor-Specific Descriptors

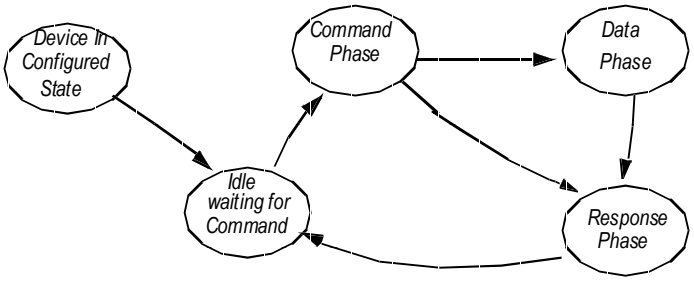
The device may support vendor-specific descriptors. The device shall return STALL if a request is received for an unrecognized or unsupported vendor-specific descriptor.

D.7 Still Image Capture Device Class-Specific Protocol

Still Image Capture Devices that are compatible to this annex support a Bulk-pipe interface used for image transfer or data transfer. This interface is used to provide transfer of coherent data objects such as image thumbnails, descriptors of the data objects, an image file, a code file, meta-data file, or any opaque data file. This interface is also used to manage stores and other items that affect data access in the device and it is used to manage the controls and modes of operation of the device. This contrasts with block data access provided by a mass storage interface.

A Still Image Capture Device that is compatible to PIMA15740 adheres to a protocol model that involves three phases of operation execution. These are *Command*, *Data*, and *Response*. This three-phase operation execution model applies to the Bulk-Only Protocol defined in this Class specification.

Figure D.3: Operation Phase State Diagram



In the *Command* phase the host transfers to the Still Image Capture Device a *Command Block* that defines the PIMA15740 operation that the host is requesting the device to perform. The specific contents of the *Command Block* correspond to an operation defined in the PIMA15740 specification. When a device is configured and idle, it is ready to receive a command. A device enters the *Command* phase when a *Command Block* is sent to the device. A device determines that a *Command Block* has been completely received when the device accepts from the host the number of bytes specified in the first four bytes of the *Command Block* that coincides with a short or null packet. A short packet or a NULL packet indicates the end of a *Command* phase. If the number of bytes specified in the first four bytes of the *Command Block* are an integral multiple of the wMaxPacketSize field of the Endpoint Descriptor the *Command* phase will end in a NULL packet. If the number of bytes transferred in the *Command* phase is less than that specified in the first four bytes of the *Command Block* then the device has received an invalid command and should STALL the Bulk-Pipe (refer to Clause 7.2).

After accepting and interpreting a command, a device optionally enters the *Data* phase. In the *Data* phase the data to be transferred is contained in a *Data Block.* The first four bytes of a *Data Block* describe the length in bytes of the data to be transferred. The operation code in the *Command Block* determines if the operation requires data transfer. The operation code

also determines the direction of data transfer (host to device - data out, or device to host - data in). The *Data* phase ends when the number of bytes transferred equals the number of bytes specified in the first four bytes of the *Data Block* that coincide with a short or a NULL packet. A short packet or a NULL packet indicates the end of the end of a *Data* phase. If the number of bytes specified in the first four bytes of the Data Block are an integral multiple of the wMaxPacketSize field of the Endpoint Descriptor the *Data* phase will end in a NULL packet.

If the number of bytes transferred in the *Data* phase is less than that specified in the first four bytes of the *Data Block* and the data receiver detects this condition before the initiation of the *Response* phase the data transfer may be cancelled (refer to Clause 7.2).

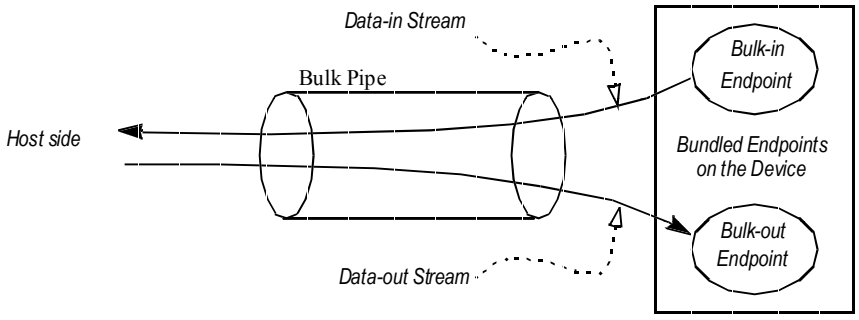
After the *Command* phase in operations without data transfer, or after the *Data* phase in operations with data transfer, the device enters the *Response* phase. The *Response* phase returns operation completion information to the host in a container called the *Response Block*. The first four bytes of a *Response Block* describe the length in bytes of the command completion data to be transferred.

The host determines that a *Response Block* has been completely received when the device sends a non-maximum length data packet. The maximum packet size is determined by the value set in the wMaxPacketSize field of the Endpoint Descriptor corresponding to the endpoint utilized in the status information transfer. Additionally, the host may also determine that a *Response Block* has been completely received when the device sends a NULL packet. This method will be used when the *Response Block* size in bytes is an integral number of maximum data packets.

Endpoint Utilization in Still Image Bulk-only Protocol

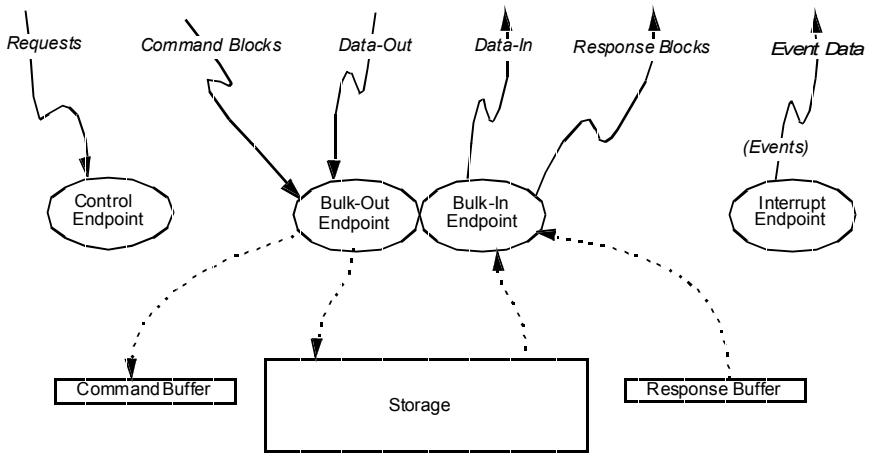
The Bulk-Only Protocol defined in this annex bundles together a Bulk-out Endpoint and a Bulk-in Endpoint into a single logical entity that services a thread of PIMA 15740 operations. This configuration may be abstracted to have a "Data-in Stream" that transfers bytes from the device to the host and a "Data-out Stream" that transfers bytes from the host to the device as shown in Figure D.7.0-2.

Figure D.4: Bulk-Only Protocol Streams



The Data-out Stream transfers operations, operation parameters, and any data from the host to the device (*Command Blocks* and *Data(out) Blocks*). The Data-in Stream transfers operation responses and any data from the device to the host (*Data(in) Blocks* and *Response Blocks*). A short packet or a NULL packet sent by the data source indicates the end of the transfer's phase.

Figure D.5: A USB Still Image Capture Device



D.7.1 Bulk-Pipe Containers

In a Still Image Capture Device the bulk pipe is used for the PIMA15740 operations that deal with file transfer and management. The bulk pipe has an input stream and an output stream connected to a common endpoint address. This common endpoint address is what leads to the abstraction of bundled endpoints. The PIMA15740 information is encapsulated into the streams using containers. A container has four types; *Command Block*, *Data Block*, *Response Block, and Event Block*. The *Event Blocks* are not used on the Bulk-Pipe but rather the Interrupt Pipe for encapsulating asynchronous event data.

D.7.1.1 Generic Container Structure

The generic structure of the containers used by a PIMA15740 compatible Still Image Capture Device follows.

Table D.20: Generic Container Structure

Byte Offset	Length (Bytes)	Field Name	Description
0	4	Container Length	This field encodes as an unsigned integer the number of bytes in this container. A Still Image Capture Device uses this field to determine the size of the container.
4	2	Container Type	This field describes the type of the container: 0 undefined 1 *Command Block* 2 *Data Block* 3 *Response Block* 4 *Event Block* otherwise reserved
6	2	Code	This field contains the PIMA15740 OperationCode, ResponseCode, or EventCode. The Data Block will use the OperationCode from the Command Block
8	4	TransactionID	This is a host generated number that associates all phases of an PIMA15740 operation.
12	??	Payload	The contents of this field depend on the operation and phase of the PIMA15740 operation.

Notes:

1. The OUT Stream can have *Command Blocks* and *Data Blocks*. The IN Stream can have *Data Blocks* and *Response Blocks*.

2. The Still Image Bulk-only Protocol, which is based on the PIMA15740 protocol, does not allow queuing of operations. Consequently when a *Command Block* optionally followed by a *Data Block* is sent by the host on the OUT Stream, a *Response Block* must be returned from the device on the IN Stream before the next *Command Block* can be set by the host to the device on the OUT Stream.

3. The data in the containers is in little endian format.

D.7.1.2 Command Block Payload Structure

The structure of the payload of a *Command Block* used by a Still Image Capture Device compatible with this annex follows.

Table D.21: Command Block Payload Structure

Relative Byte Offset	Length (Bytes)	Field Name	Description
0	4	Parameter 1	This field contains an operation parameter. The format and meaning of the parameter is described in the operation description of each operation (PIMA15740 Operation)..
??	4	Parameter N	This field contains an operation parameter. The format and meaning of the parameter is described in the operation description of each operation (PIMA15740 Operation).

Notes:

1. Parameters are always 4 bytes in length. The number of parameters can be determined from the Container length. (Length – 12)/4

2. The *Command Block* payload does not have a transfer length parameter. This prevents the responding device from knowing a priori the amount of any data associated with an operation.

D.7.1.3 Data Block Payload Structure

The structure of the payload of a *Data Block* used by a Still Image Capture Device compatible with this annex is not defined. The actual structure of the data in a *Data* phase depends on the operation associated with the data.

D.7.1.4 Response Block Payload Structure

The structure of the payload of a *Response Block* used by a Still Image Capture Device compatible with this annex follows.

Table D.22: Response Block Payload Structure

Relative Byte Offset	Length (Bytes)	Field Name	Description
0	4	Parameter 1	This field contains a response parameter. The format and meaning of the parameter is described in the response description of each operation (PIMA15740 Operation).
??	4	Parameter N	This field contains a response parameter. The format and meaning of the parameter is described in the response description of each operation (PIMA15740 Operation).

Notes:

1. Parameters are always 4 bytes in length. The number of parameters can be determined from the Container length. (Length – 12)/4

D.7.2 Still Image Bulk-only Protocol

This clause describes the implementation of Bulk-only protocol on a Still Image Capture Device.

Still Image Capture Devices defined in this annex support only a single thread of operations. The devices have one command buffer that processes commands sequentially. The bInterfaceProtocol field in the Still Image Interface Descriptor shall indicate Bulk-Only protocol.

D.7.2.1 Data Transfer Cancellation in Still Image Bulk-only Protocol

The information transfer over the Bulk-pipe can be cancelled at any time by either the host or the Still Image Capture Device. Normally only the *Data* phase is cancelled as the *Command* phase and the *Response* phase are short. An exception is when the device receives an invalid operation that will cause it to STALL the Bulk-Pipe. Consequently, a container may be only partially transferred by a stream or the data transfer may stop between containers. The Still Image Bulk-only Protocol enables detection of a cancelled information transfer condition allowing for recovery.

D.7.2.1.1 Rule Set

The cancellation data transfer over the Bulk-Pipe involves the use of two methods. One method is used by device-initiated cancels and the other method is used by host-initiated cancels.

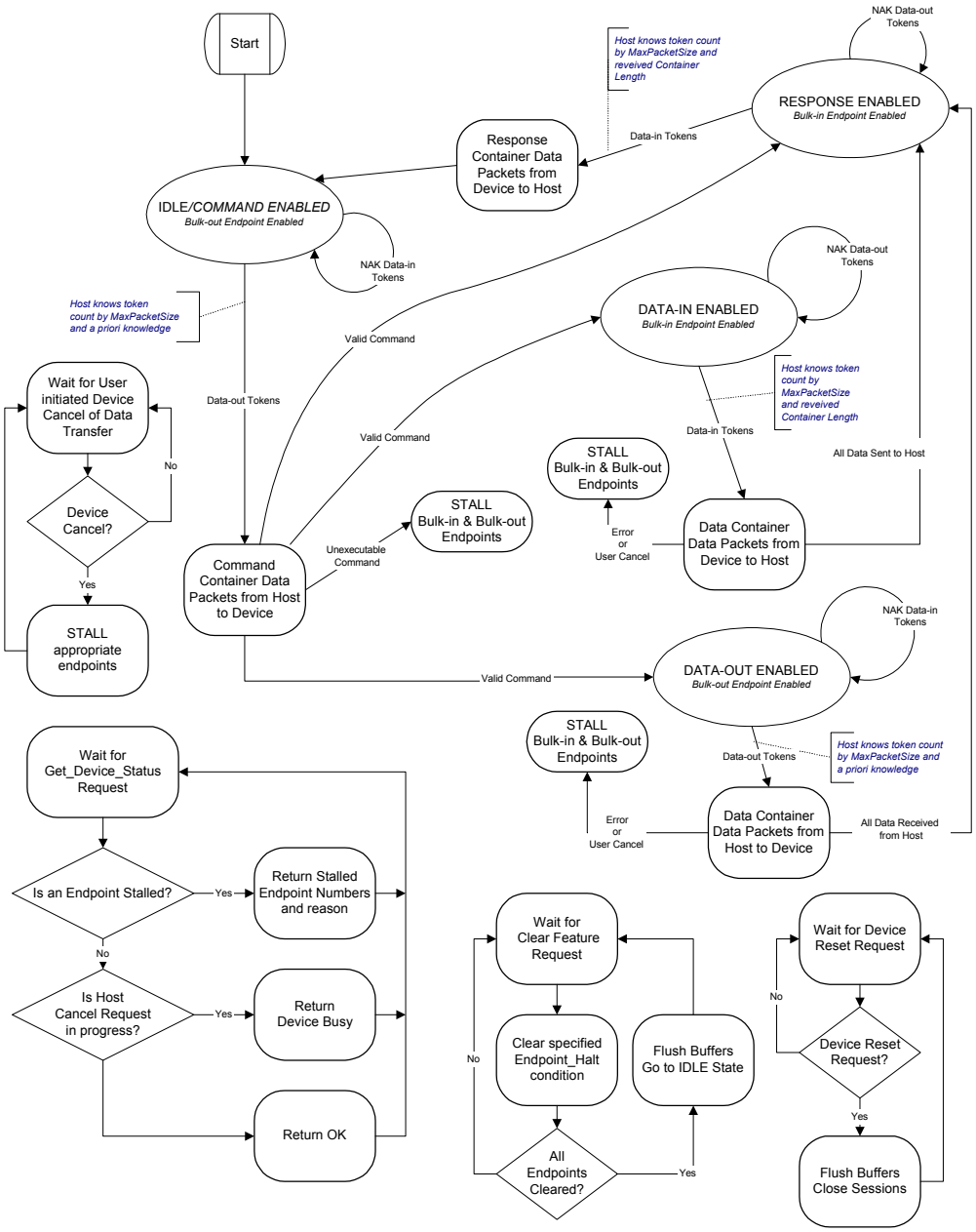
The Still Image Device shall cancel a data transfer by stalling the Bulk-Pipe endpoints. The host shall then determine the reason for the STALL (ENDPOINT_HALT condition) and the stalled endpoint numbers by issuing the class-specific Get_Device_Status Request (refer to Clause 5.2.4). The host shall then issue standard Clear_Feature Requests to clear the ENDPOINT_HALT condition on the affected endpoints. After the Clear_Feature Requests the device shall return OK status on subsequent Get_Device_Status Requests to indicate to the host that the device is ready to resume operations. The OK status corresponds to PIMA15740 Response Code 0x2001.

The host shall cancel a data transfer by no longer issuing tokens on the Bulk-Pipe and then issuing a class-specific Cancel Request to the device. The host shall then poll the device with Get_Device_Status Requests and when the device returns OK status commands may be resumed.

D.7.2.1.2 Device Behavior

The following flowchart describes the behavior of a device that implements Still Image Bulk-only Protocol.

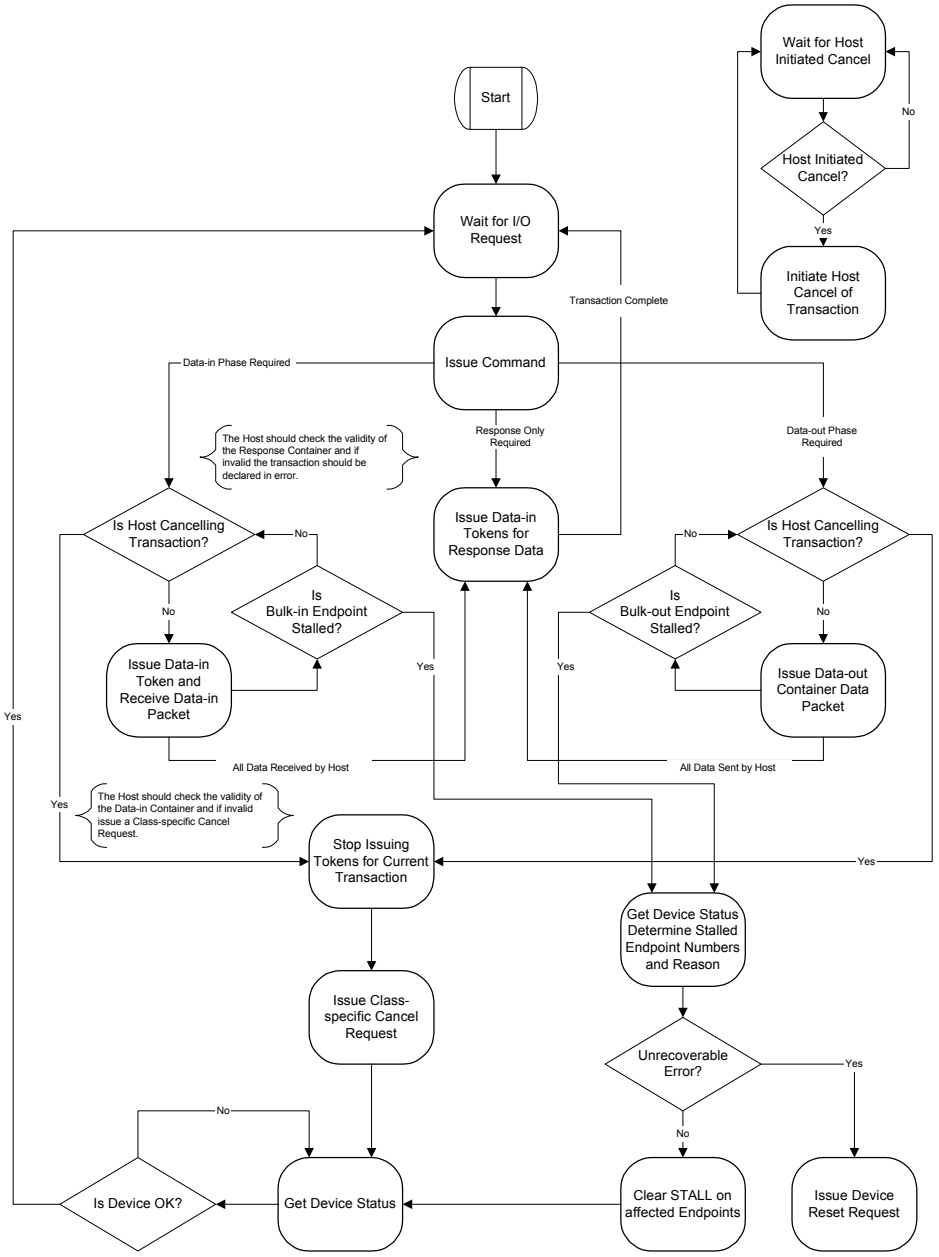
Figure D.6: Still Image Protocol Device Behavior



D.7.2.1.3 Host Behavior

The following flowchart describes the behavior of a host that implements Still Image Bulk-only Protocol.

Figure D.7: Still Image Protocol Host Behavior



D.7.3 Asynchronous Event Notification

A Still Image Capture Device that is compliant to PIMA 15740 shall provide a means to notify the host of the occurrence of certain events. Events such as the removal of a memory card while the still image capture device is actively connected to the host is an example. A USB Still Image Capture Device uses the interrupt endpoint associated with the Still Image Interface for this purpose. The device returns to the host interrupt data with zero or more (up to three) parameters that identify the asynchronous event by PIMA15740 Event Code. For Asynchronous Events that require a large amount of data, the class-specific Get Extended Event Data Request shall be used.

The Get Extended Event Data Request is issued by the host when a Check Device Condition Asynchronous Event is received by via the Interrupt Pipe.

D.7.3.1 Asynchronous Event Interrupt Data Format

The device shall return PIMA15740 interrupt data formatted as follows:

Table D.23: Format of Asynchronous Event Interrupt Data

Offset	Field	Size (Bytes)	Value	Description
0	Interrupt Data Length	4	number	This field indicates the length in bytes of the interrupt data.
4	Container Type	2	0x0004	Container Type = Event
6	Event Code	2	code	The PIMA15740 Event Code.
8	TransactionID	4	code	An unsigned 32-bit field containing the PIMA15740 TransactionID. This field is 0x00000000 if a TransactionID does not apply to this event.
12	Event Parameter 1	4	variable	This field contains the 1st parameter associated with the event if needed.
16	Event Parameter 2	4	variable	This field contains the 2nd parameter associated with the event if needed.
20	Event Parameter 3	4	variable	This field contains the 3rd parameter associated with the event if needed.

Notes:

1. Event parameters are always 4 bytes in length. The number of parameters can be determined from the Interrupt Data Length. (Length – 12)/4

D.8 Specific Structure of the PIMA15740 Datasets transferred in a Data Block.

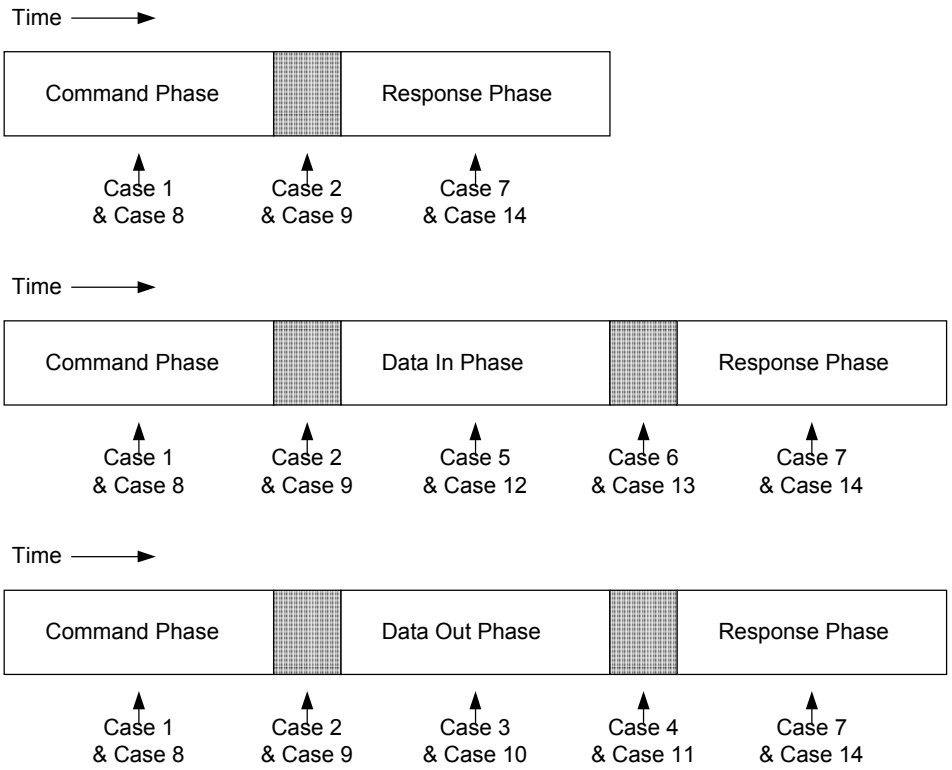
Four PIMA15740 datasets, namely the DeviceInfo Dataset, the ObjectInfo Dataset, the Device Property Describing Dataset, and the StorageInfo Dataset are transferred on the Bulk-pipe in data containers. All data items of these datasets are transferred together in one data container in one operation. Therefore, the binary structure of these datasets needs to be defined. One should refer to the PIMA15740 specification for the definitions and ordering of the dataset fields, determining the offsets of individual fields within each dataset using the relative order and length.

The PIMA15740 specification does not define byte significance within each field that is part of a dataset. This is commonly known as "endianness". Little-endian systems place the least significant byte of a multi-byte field first in a stream or at the lowest address in a memory buffer. Big-endian systems place the most significant byte of a multi-byte field first in a stream or at the lowest address of a memory buffer. The Universal Serial Bus is a little-endian system. All fields are packed accordingly.

D.9 Still Image Bulk-only Protocol Cancellation Examples

This clause presents specific cancellation examples in the Still Image Bulk-only Protocol. The different cases presented correspond to the indicated positions in the following figure that depicts the three phases of the Still Image Bulk-only Protocol.

Figure D.8: Cancellation Cases



Note: "just finished receiving" means "just received a short or NULL data packet" which terminates a Block.

Case 1: The HOST cancels when the DEVICE is receiving the Command Block, or

Case 2: The HOST cancels when the DEVICE has just finished receiving the Command Block, or

Case 3: The HOST cancels when the DEVICE is receiving the Data(out) Block, or

Case 4: The HOST cancels when the DEVICE has just finished receiving the Data(out) Block, or

Case 5: The HOST cancels when the HOST is receiving the Data(in) Block, or

Case 6: The HOST cancels when the HOST has just finished receiving the Data(in) Block.

Normally the Command Phase is not cancelled because it is short. If it is cancelled, then the following occurs:

1. The HOST stops issuing tokens.

2. The HOST sends to the DEVICE the class-specific Cancel Request.

3. The DEVICE enters a DEVICE_BUSY condition that is reported when the HOST issues a Get Device Status Request. DEVICE BUSY is reported by using the PIMA15740 Response Code 0.2019

4. The DEVICE clears its command buffer, goes to the *Idle/Command Enabled* state, and enters an OK condition.

5. The HOST polls the DEVICE with the Get Device Status Request. When the DEVICE returns OK status the cancel is complete.

Case 7: The HOST cancels when the HOST is receiving the Response Block.

1. The HOST does not cancel a Response Block as the Response Phase has the completion information of a completed transaction.

Case 8: The DEVICE cancels when the DEVICE is receiving the Command Block, or

Case 9: The DEVICE cancels when the DEVICE has just finished receiving the Command Block, or

Case 10: The DEVICE cancels when the DEVICE is receiving the Data(out) Block, or

Case 11: The DEVICE cancels when the DEVICE has just finished receiving the Data(out) Block, or

Case 12: The DEVICE cancels when the HOST is receiving the Data(in) Block, or

Case 13: The DEVICE cancels when the HOST has just finished receiving the Data(in) Block.

Since the Command Block is short and may only be one data packet in length the DEVICE usually does not cancel the Command Block. However, if an invalid command is received the DEVICE may cancel in Command phase.

1. The DEVICE places an Endpoint_Halt condition as indicated by the PIMA15740 Response Code, TransactionCancelled, 0x201F, on both the Bulk-in and Bulk-out endpoints. Consequently, any tokens issued by the HOST to these endpoints will return STALL.

2. The HOST issues a Get Device Status Request to determine the reason for the STALL and the endpoint numbers of the endpoints in an Endpoint_Halt condition.

3. The HOST may the issue Clear Feature Requests to clear the Endpoint_Halt condition on the endpoints returning STALL.

4. The DEVICE clears its command buffer, goes to the *Idle/Command Enabled* state, and enters an OK condition.

5. The HOST polls the DEVICE with the Get Device Status Request. When the DEVICE returns OK status the cancel is complete.

alternatively

3. The HOST sends to the DEVICE the class-specific Device Reset Request

4. The DEVICE clears its command buffer, closes all open sessions, and returns to the Configured State.

Case 14: The DEVICE cancels when the HOST is receiving the Response Block.

1. The Device shall not cancel a Response Block

Annex E: Bibliography

(informative)

Informative references relevant to the development of this standard are listed below.

[1] Design Rule for Camera File System, version 1.0, JEIDA-49-2-1998, Japan Electronic Industry Development Association, Dec. 1998.

[2] USB specification available via world-wide web at: http://www.teleport.com/~usb/

[3] IEEE1394 information available via world-wide web at: http://www.1394ta.org

[4] FlashPix information available from: http://www.digitalimaging.org/

[5] G3 colour fax information available from http://www.faximum.com/faqs/fax.info#1.2

[6] IrDA information available from http://www.irda.org

[7] ICC information available from http://www.color.org

[8] ISIS information available from http://www.pixtran.com

[9] TWAIN information available from http://www.twain.org

[10] IEEE P1394 Digital Camera Draft, Rev 1.03, Sony Corp, 1995.

[11] Melville, John, et. al., "An application programmer's interface for digital cameras", Proc. IS&T's 49th Annual Conference, May 1996, pp. 282-285.

[12] PNG information available at http://www.cdrom.com/pub/png/spec/

[13] PICT specification "Inside Macintosh: Imaging with QuickDraw," Addison Wesley Publishing Company, 1994 ISBN: 020163242X. PDF at http://developer.apple.com/techpubs/mac/QuickDraw/QuickDraw-2.html