

# Photography — Electronic still picture imaging — Picture transfer protocol (PTP) for digital still photography devices

ICS 37.040.99

# National foreword

This British Standard is the UK implementation of ISO 15740:2008. It supersedes BS ISO 15740:2005 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee CPW/42, Photography.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

**Compliance with a British Standard cannot confer immunity from legal obligations.**

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2008

© BSI 2008

ISBN 978 0 580 59154 9

## Amendments/corrigenda issued since publication

| Date | Comments |
|------|----------|
|      |          |
|      |          |
|      |          |
|      |          |

# INTERNATIONAL STANDARD

**ISO**  
**15740**

Second edition  
2008-05-01

---

---

## Photography — Electronic still picture imaging — Picture transfer protocol (PTP) for digital still photography devices

*Photographie — Imagerie des prises de vue électroniques — Protocole  
de transfert d'images (PTP) pour les appareils photographiques  
électroniques numériques*



Reference number  
ISO 15740:2008(E)

**AVA** Documents  
Document Sharing Hub



# Contents

Page

|                                                |    |
|------------------------------------------------|----|
| Foreword.....                                  | v  |
| Introduction .....                             | vi |
| 1 Scope .....                                  | 1  |
| 2 Normative references .....                   | 1  |
| 3 Terms and definitions.....                   | 1  |
| 4 Digital still photography device model.....  | 5  |
| 4.1 Overview .....                             | 5  |
| 4.2 Baseline requirements .....                | 6  |
| 5 Data format specification .....              | 7  |
| 5.1 General format .....                       | 7  |
| 5.2 Data types.....                            | 7  |
| 5.3 Simple types.....                          | 9  |
| 5.4 Arrays.....                                | 11 |
| 5.5 Datasets .....                             | 12 |
| 6 Image and data object formats .....          | 21 |
| 6.1 Object usage .....                         | 21 |
| 6.2 Thumbnail formats.....                     | 23 |
| 6.3 ObjectFormatCodes.....                     | 24 |
| 6.4 Object format version identification ..... | 24 |
| 6.5 Data object association.....               | 24 |
| 7 Transport requirements .....                 | 27 |
| 7.1 Disconnection events.....                  | 27 |
| 7.2 Reliable, error-free channel .....         | 27 |
| 7.3 Asynchronous event support.....            | 27 |
| 7.4 Device discovery and enumeration .....     | 27 |
| 7.5 Specific transports .....                  | 27 |
| 8 Persistent storage.....                      | 28 |
| 8.1 StorageID .....                            | 28 |
| 8.2 Data object referencing .....              | 28 |
| 8.3 Receiver object placement .....            | 29 |
| 9 Communication protocol .....                 | 30 |
| 9.1 Device roles .....                         | 30 |
| 9.2 Sessions .....                             | 30 |
| 9.3 Transactions.....                          | 30 |
| 9.4 Operation flow .....                       | 33 |
| 9.5 Vendor extensions.....                     | 33 |
| 10 Operations .....                            | 34 |
| 10.1 Operation overview.....                   | 34 |
| 10.2 Operation parameters.....                 | 34 |
| 10.3 OperationCode format.....                 | 35 |
| 10.4 OperationCode summary .....               | 35 |
| 10.5 Operation descriptions.....               | 35 |
| 11 Responses .....                             | 60 |
| 11.1 ResponseCode format.....                  | 60 |
| 11.2 ResponseCode summary.....                 | 61 |
| 11.3 Response descriptions .....               | 62 |

|                     |                                                              |            |
|---------------------|--------------------------------------------------------------|------------|
| <b>12</b>           | <b>Events.....</b>                                           | <b>66</b>  |
| <b>12.1</b>         | <b>Event usage .....</b>                                     | <b>66</b>  |
| <b>12.2</b>         | <b>Event types .....</b>                                     | <b>67</b>  |
| <b>12.3</b>         | <b>Event dataset .....</b>                                   | <b>67</b>  |
| <b>12.4</b>         | <b>EventCode format .....</b>                                | <b>68</b>  |
| <b>12.5</b>         | <b>EventCode summary .....</b>                               | <b>68</b>  |
| <b>12.6</b>         | <b>Event descriptions .....</b>                              | <b>68</b>  |
| <b>13</b>           | <b>Device properties .....</b>                               | <b>72</b>  |
| <b>13.1</b>         | <b>Device property usage.....</b>                            | <b>72</b>  |
| <b>13.2</b>         | <b>Values of a device property .....</b>                     | <b>72</b>  |
| <b>13.3</b>         | <b>Device property management requirements .....</b>         | <b>73</b>  |
| <b>13.4</b>         | <b>Device property identification.....</b>                   | <b>73</b>  |
| <b>13.5</b>         | <b>Device property descriptions.....</b>                     | <b>76</b>  |
| <b>14</b>           | <b>Streaming (PTP v1.1 only) .....</b>                       | <b>93</b>  |
| <b>14.1</b>         | <b>Streaming overview .....</b>                              | <b>93</b>  |
| <b>14.2</b>         | <b>Stream transfer.....</b>                                  | <b>93</b>  |
| <b>14.3</b>         | <b>Multiplexing .....</b>                                    | <b>94</b>  |
| <b>14.4</b>         | <b>Discovering and configuring stream capabilities.....</b>  | <b>94</b>  |
| <b>14.5</b>         | <b>Data transfer mechanism .....</b>                         | <b>94</b>  |
| <b>14.6</b>         | <b>Packet layout .....</b>                                   | <b>95</b>  |
| <b>14.7</b>         | <b>Frame layout.....</b>                                     | <b>96</b>  |
| <b>14.8</b>         | <b>Enumerating supported streams.....</b>                    | <b>96</b>  |
| <b>14.9</b>         | <b>Retrieving stream information .....</b>                   | <b>96</b>  |
| <b>15</b>           | <b>Conformance section .....</b>                             | <b>96</b>  |
| <b>Annex A</b>      | <b>(informative) Optional device features.....</b>           | <b>99</b>  |
| <b>Annex B</b>      | <b>(normative) Object referencing and format codes .....</b> | <b>101</b> |
| <b>Annex C</b>      | <b>(informative) Operation flow example scenarios .....</b>  | <b>103</b> |
| <b>Annex D</b>      | <b>(informative) Filesystem implementation examples.....</b> | <b>107</b> |
| <b>Annex E</b>      | <b>(informative) Reference to OSI model .....</b>            | <b>110</b> |
| <b>Annex F</b>      | <b>(informative) SendObject implementation example.....</b>  | <b>113</b> |
| <b>Bibliography</b> | <b>.....</b>                                                 | <b>116</b> |

## Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 15740 was prepared by Technical Committee ISO/TC 42, *Photography*.

This second edition cancels and replaces the first edition (ISO 15740:2005) which has been technically revised.

## Introduction

This second edition of ISO 15740 (hereinafter designated PTP v1.1) provides optional support for new increased performance and compatibility. All new constructs are fully backward compatible with the first edition (hereinafter designated PTP v1.0) and are optional. See 5.5.2 for standard version.

For the purposes of this International Standard, digital still photography devices (DSPDs) are defined as devices with persistent storage which 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 device, such as a printer. A number of high speed interface transports has been developed, including USB, TCP/IP and IEEE 1394 (FireWire). This International 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 behaviour, operations, responses, events, device properties, datasets and data formats to ensure interoperability. This International Standard also provides optional operations and formats, as well as extension mechanisms.

This International Standard specifies the following:

- behaviour requirements for DSPDs; this includes the baseline features a device needs to support in order to provide interoperability over conforming transports;
- functional requirements needed by a transport to facilitate the creation of a transport-dependent implementation specification that conforms to this International 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 of describing data object associations and filesystems and
  - mechanisms for implementing extensibility.

This International 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 International Standard is implemented;
- an application programming interface; this is left to the platforms upon which support for this International Standard is implemented.

This International 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<sup>[15]</sup> and ISO 12234-2, as well as the Design Rule for Camera Filesystem (DCF) and the Digital Print Order Format (DPOF).

The technical content of this International Standard is closely related to PIMA 15740:2000. The main difference is that PIMA 15740:2000 includes an informative annex describing a USB implementation of ISO 15740. This information is not included in this International Standard, which instead references the USB still device class document developed by the Device Working Group of the USB Implementers Forum.

PTP v1.1 provides optional support for new increased performance and compatibility. All new constructs are fully backward compatible with PTP v1.0 and are optional.

— Performance Enhancements:

- Support for retrieval of ObjectHandles in enumerated chunks, via specification of three new optional operations and a new response code. This may reduce long response times for some initiators that possess large numbers of objects.
- Support for optional arbitrary resizing prior to image transmission via specification of a new operation GetResizedImageObject. In PTP v1.0, image sizes might be requested in full-resolution or thumbnail size only.
- Support for arrays of datasets. This can be used to reduce the number of required transactions necessary for device characterization from being a function of the number of objects on the device to one.
- An optional fast file characterization operation called GetFilesystemManifest that exploits dataset arrays to request, in a single transaction, only the minimum data required to characterize a typical filesystem. Many initiators, particularly in printing scenarios, are interested in fast filesystem characterization for access to a specifically named file in a particular place. This capability can significantly improve end-user workflow latency. This single operation replaces the typical series of many GetObjectInfo requests with a binary filesystem manifest. This manifest is defined as a simple array of a subset of the standard ObjectInfo dataset called the ObjectFilesystemInfo dataset. This operation replaces the need for many GetObjectInfo calls, while also avoiding the need for responders to perform many internal file-opens on the fly, or to cache ObjectInfo image data that is often held persistently only “inside” internal image files (e.g. TIFF tags inside EXIF JPEGs), to quickly communicate only the fast filesystem information.

— Compatibility Enhancements:

- An optional mechanism to support multiple vendor extension sets. This is specified via the new VendorExtensionMap dataset, and two new optional operations that may be invoked outside of a session (GetVendorExtensionMaps and GetVendorDeviceInfo).
- The optional fast file characterization method GetFilesystemManifest natively supports extremely large objects, by requiring 8-bytes for object size (UINT64), as opposed to the standard 4-bytes.
- A new standard ObjectFormatCode to support the Digital Negative file format (DNG).

— Feature Enhancement:

- An optional mechanism for handling streaming content. This is specified via the new StreamInfo dataset, as well as the supporting GetStreamInfo and GetStream operations, as well as some optional new supporting DeviceProperties. This is described in a new Clause 14.



# Photography — Electronic still picture imaging — Picture transfer protocol (PTP) for digital still photography devices

## 1 Scope

This International Standard provides a common communication protocol for exchanging images with and between digital still photography devices (DSPDs). This includes communication between DSPDs and host computers, printers, other digital still devices, telecommunications kiosks and image storage and display devices.

This protocol is transport- and platform-independent.

## 2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO 12234-2, *Electronic still-picture imaging — Removable memory — Part 2: TIFF/EP image data format*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

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

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

## 3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

### 3.1

#### **album**

end-user-created object used to logically group data objects according to some user-defined criteria

**NOTE** An album might or might not be a physical folder in a filesystem. In this International Standard, an album is a type of association.

### 3.2

#### **association**

logical construct used to expose a relationship between discrete objects

**NOTE** Associations are used to indicate that separate data objects are related. Associations are represented like folders, and can be nested using a standard branched hierarchical tree structure.

**EXAMPLE** A time sequence, or user-defined groupings by content or capture session.

## 3.3

### **connection**

transport-provided mechanism for establishing paths for transferring data between devices

## 3.4

### **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

## 3.5

### **data object**

image or other type of data that typically exists in persistent storage of a DSPD or other device

## 3.6

### **dataset**

transport-independent collection of one or more individual data items with known interpretations

NOTE Datasets are not necessarily opaque nor atomic to transport implementations.

## 3.7

### **Design Rule for Camera Filesystem**

#### **DCF**

standard convention for camera filesystems which specifies the file format, foldering and naming conventions in order to promote file interoperability between conforming digital photography devices

## 3.8

### **device discovery**

act of determining the set of all devices present on a particular transport or platform that are physically or logically accessible

## 3.9

### **digital still photography device**

#### **DSPD**

device with persistent storage which captures a two-dimensional digital still image

## 3.10

### **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

NOTE DPOF also allows index prints, cropping, and text overlays to be specified.

## 3.11

### **enumeration**

act of creating an ordered increasing numerical list that contains one representative element for each member of a set

## 3.12

### **Exif/JPEG**

compressed file format for digital cameras in which the images are compressed using the baseline JPEG standard described in ISO 12234-2

NOTE In Exif, metadata and thumbnail images are stored using TIFF tags within an application segment at the beginning of the JPEG file.

## 3.13

### **folder**

optional sub-structure in a hierarchical storage area that can contain data objects

**3.14****FlashPix**

image file format, defined in *FlashPix Format Specification*, using a structured storage file containing metadata and a tiled, hierarchical image representation

NOTE The tiles in a FlashPix image are normally baseline JPEG images, and individual image tiles of a particular resolution can be easily accessed for rapid display and editing.

**3.15****IEEE 1394**

high-speed serial bus standardized by the IEEE (Institute of Electrical and Electronics Engineers) currently having clock rates of 100, 200 and 400 Mbits/s

NOTE IEEE 1394 is often referred to as FireWire.

**3.16****image aspect ratio**

ratio of the image width to the image height

**3.17****image capture device**

device for converting a scene or a fixed image, such as a print, film or transparency, to digital image data

**3.18****image output device**

device that can render a digital image to hardcopy or softcopy media

**3.19****in-band event**

event transmitted on the same logical connection as operations and responses

NOTE Events are only asynchronous to the degree of data precision for which the transport implementation allows event interleaving.

**3.20****initiator**

device that initiates a conversation by opening a session, and issues all formal operations to the responder

NOTE The initiator is analogous to the client in the client/server paradigm.

**3.21****International Imaging Industry Association****I3A**

organization that serves to represent the common interests among manufacturers of imaging technology products

NOTE See <http://www.i3a.org>.

**3.22****Infrared Data Association****IrDA**

infrared wireless communication system that currently supports wireless communication at data rates between 9 600 bps and 4 Mbps.

**3.23****Joint Photographic Experts Group****JPEG**

specific image compression method defined in ISO/IEC 10918-1

## 3.24

### **LogicalStorageID**

least significant sixteen bits of a StorageID

NOTE This value uniquely identifies one logical storage area within the physical store indicated in the PhysicalStorageID.

## 3.25

### **Most Significant Nibble**

#### **MSN**

most significant four bits of the most significant byte

## 3.26

### **object aggregation**

act of taking one or more location-specific lists of objects that exist on a particular device and grouping them together in one set

## 3.27

### **ObjectHandle**

device-unique 32-bit unsigned integer assigned by a device to each data object in local persistent storage which is provided to external devices

NOTE External recipients of an ObjectHandle must use it to reference that piece of data in subsequent transactions. ObjectHandles are guaranteed to be persistent over at least a session.

## 3.28

### **out-of-band event**

event transmitted on a different logical connection to that for operations and responses

NOTE Out-of-band events are asynchronous from operation transactions.

## 3.29

### **personal computer**

#### **PC**

any personal computing device, which may employ various hardware architectures and operating systems

## 3.30

### **PhysicalStorageID**

most significant sixteen bits of a StorageID

NOTE This value uniquely identifies one physical storage area on a device, although there may be more than one logical store per physical store.

## 3.31

### **Portable Network Graphics**

#### **PNG**

extensible file format for lossless, portable, compressed storage of raster images

NOTE PNG supports indexed colour, greyscale, truecolour and an optional alpha channel.

## 3.32

### **protocol**

defined mechanisms for exchanging data between devices

## 3.33

### **pull model**

use paradigm for DSPDs where the object receiver initiates the operation requests to transfer data objects from the sender

**3.34****push model**

use paradigm for DSPDs where the object sender initiates the operation requests to transfer data objects to the receiver

**3.35****QuickDraw picture**

file format consisting of sequences of saved drawing commands

NOTE QuickDraw files are commonly referred to as PICT files.

**3.36****responder**

device that responds to operations from the initiator

NOTE The responder is analogous to a server in the client/server paradigm.

**3.37****session**

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

**3.38****square pixel sampling**

image having equal sample spacing in the two orthogonal sampling directions

**3.39****StorageID**

device-specific four-byte unsigned integer (UINT32) that represents a unique storage area that may contain data objects

NOTE The most significant sixteen bits of a StorageID represent the PhysicalStorageID, whilst the least significant sixteen bits of a StorageID represent the LogicalStorageID.

**3.40****transport aggregation**

act of taking one or more transport-specific list of conforming devices that are logically or physically accessible in a system and grouping them in one set that spans all transports across the particular system

**3.41****transport**

means of attaching the digital capture device to some other digital device including a physical wire or a wireless connection

**3.42****Universal Serial Bus****USB**

digital interface for connecting up to 127 devices in a tiered-star topology

NOTE 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 a 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 International Standard supports devices with a wide range of potential features. However, a small number of features is required for conformance with this International Standard, while many others are optional. Subclause 4.2 describes the required features and functionality. Annex A describes features that are not required for conformance but which should be implementable using this International Standard and its extension mechanisms.

Standard data formats for datatypes and datasets are described in Clause 5.

Clause 6 describes required and optional support for particular image and non-image formats and metadata. Clause 6 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 describes these requirements.

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

Clause 9 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 lists the standard operations, their corresponding optional operation codes and their usages. Standard responses to operations are defined in Clause 11. The use of events is mandatory in order to ensure synchronization between devices. Clause 12 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.

Clause 15 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**

### **4.2.1 General**

The requirements listed in 4.2.2 to 4.2.5 shall be met in order for a DSPD to conform to this International Standard.

### **4.2.2 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 7.

### **4.2.3 Thumbnail support**

The DSPD shall provide support for thumbnails as described in 6.2.

### **4.2.4 Standard image and data reference behaviour**

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.

### **4.2.5 Asynchronous event support**

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

## 5 Data format specification

### 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 International Standard are numbered so that the least significant bit is at the zero position, holding the right-most position in the field; e.g. 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 International Standard uses hexadecimal notation as a means of concisely describing multibyte fields. All hexadecimal byte fields 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 Data types

#### 5.2.1 Datatype summary

The types of data that are defined in this International Standard as having specific interpretations of their data content are listed in Table 1.

Table 1 — Datatype summary

| Name                 | Size (bytes) | Format                            | PTP Version |
|----------------------|--------------|-----------------------------------|-------------|
| OperationCode        | 2            | Datacode (UINT16)                 | 1.0+        |
| ResponseCode         | 2            | Datacode (UINT16)                 | 1.0+        |
| EventCode            | 2            | Datacode (UINT16)                 | 1.0+        |
| DevicePropCode       | 2            | Datacode (UINT16)                 | 1.0+        |
| ObjectFormatCode     | 2            | Datacode (UINT16)                 | 1.0+        |
| StorageID            | 4            | Special (UINT32)                  | 1.0+        |
| ObjectHandle         | 4            | Handle (UINT32)                   | 1.0+        |
| DateTime             | Variable     | String                            | 1.0+        |
| DeviceInfo           | Variable     | Dataset                           | 1.0+        |
| StorageInfo          | Variable     | Dataset                           | 1.0+        |
| ObjectInfo           | Variable     | Dataset                           | 1.0+        |
| DevicePropDesc       | Variable     | Dataset                           | 1.0+        |
| DevicePropDescEnum   | Variable     | Enumerated form of DevicePropDesc | 1.0+        |
| DevicePropDescRange  | Variable     | Range form of DevicePropDesc      | 1.0+        |
| Object               | Variable     | Variable                          | 1.0+        |
| VendorExtensionMap   | 8            | Dataset                           | 1.1+        |
| ObjectFilesystemInfo | Variable     | Dataset                           | 1.1+        |
| StreamInfo           | 36           | Dataset                           | 1.1+        |
| EnumID               | 4            | Special (UINT32)                  | 1.1+        |

### 5.2.2 Datacodes

Datacodes are 16-bit unsigned integers (UINT16) with specified interpretations, used for the purposes of enumeration. In order to aid in visual interpretation and 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 International 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 International 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, each having the same usage and definition as those specified in this International Standard. This allows for transport-aggregating abstractions in host software to use the codes defined in this International 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 International Standard, and therefore one operation code may not be sufficient.

For example, if a transport does not use the 16-bit OperationCodes, it should still provide an equivalent mechanism for the GetObject operation that supports the same usage defined in this International Standard. Another example is 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. See Table 2.

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 International 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 International 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 5.5.2.

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

## 5.3 Simple types

### 5.3.1 Simple type summary

The generic datatypes that may be used in this International Standard are listed in Table 3.

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

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 with bit 15 set to 0 | Undefined      | Reserved                           |
| All other values with bit 15 set to 1 | Vendor-defined | Vendor-defined                     |

### 5.3.2 Integers

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

### 5.3.3 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 that 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 5.5.3.

### 5.3.4 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 are typically contained inside a data object that is opaque to this protocol.

### 5.3.5 Strings

#### 5.3.5.1 String format

All strings shall consist of standard 2-byte unicode characters (USC-2) encoded with UTF-16 as described in ISO/IEC 10646, 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 Table 4. An unused string field shall be represented by specifying a length field of zero (0x00) and including no actual string bytes, resulting in a total PTP-encoded string length of 1 byte.

Table 4 — String format

| Dataset field | Size (bytes) | Datatype                       |
|---------------|--------------|--------------------------------|
| 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. Permitted 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.

#### 5.3.5.2 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 that the time is relative to a time zone. Appending neither indicates that the time zone is unspecified.

## 5.4 Arrays

### 5.4.1 Simple arrays

A simple 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. See Table 5. 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<br>(bytes) | Format  |
|---------------------------|-----------------|---------|
| NumElements               | 4               | UINT32  |
| ArrayEntry[0]             | ElementSize     | Special |
| ArrayEntry[1]             | ElementSize     | Special |
| ...                       | ElementSize     | ...     |
| ArrayEntry[NumElements-1] | ElementSize     | 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 four 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.

#### 5.4.2 Dataset arrays (PTP v1.1)

PTP v1.1 provides a standard for encoding arrays of datasets, and relies upon their usage for some optional performance and compatibility enhancements, such as those provided by the GetFilesystemManifest and GetVendorExtensionMaps operations.

Arrays of datasets are encoded in a manner similar to fixed-length types described in 5.4.1, having a “header” field that holds the number of objects in the array. A single dataset array may hold any number of objects of a single dataset type, within the constraints of 8 bytes. Arrays for datasets, being new to PTP v1.1, are slightly different in that they use 64 bits (i.e. UINT64 instead of UINT32) to hold the number of objects. As with the PTP v1.0 standard, the datatype, element size and interpretation of the individual elements of the datasets in the array shall be context-specific, and therefore not explicit within the array structure. See Table 6.

If the individual datasets in the array correspond to particular objects on the device, each of the datasets will require a field to hold an ObjectHandle, in order to distinguish which dataset describes which object.

Table 6 — Dataset array format

| Field                     | Size<br>(bytes)   | Format  |
|---------------------------|-------------------|---------|
| NumElements               | 8                 | UINT64  |
| ArrayEntry[0]             | Fixed or Variable | Dataset |
| ArrayEntry[1]             | Fixed or Variable | Dataset |
| ...                       | Fixed or Variable | ...     |
| ArrayEntry[NumElements-1] | Fixed or Variable | Dataset |

## 5.5 Datasets

### 5.5.1 Dataset usage

For the purposes of this International 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 be 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 into multiple pieces, pack the fields a particular way or pass different fields using different mechanisms.

In order to avoid inconsistencies when defining their implementations of the datasets, transport implementations should reference the field ordering in this specification. 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 which are unused are considered to have no default value, and should be set to zero.

If datasets are changed in future versions of this International Standard, fields will only be added, never deleted. No existing fields will be redefined in future versions of this International 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.

### 5.5.2 DeviceInfo dataset

See Table 7.

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 describe 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; e.g. 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 change (due to a FunctionalMode change), a DeviceInfoChanged event shall be issued to all sessions in order to indicate how its capabilities have changed.

All optional strings that are not provided should consist of an empty string. Subclause 5.3.5 describes the use of strings.

**Table 7 — 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                 |

**StandardVersion:** 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). The version value used for PTP v1.0 is 1.00 (i.e. 100), whereas the version used for PTP v1.1 is 1.10 (i.e. 110).

**NOTE** The use of 1.00 (i.e. 100) for PTP v1.0 was not formally specified in PTP v1.0, and was assumed (i.e. neither the original PIMA 15740:2000 nor ISO 15740:2005 specified the value of this field). As part of the standardization of PTP v1.1, all PTP initiators should interpret any value for Standard Version less than 110 (0, 1, 2, ..., 108, 109) as PTP v1.0, and anything larger than 110 as undefined.

**VendorExtensionID:** provides the context for interpretation of the default set of vendor extensions used by this device. These IDs are assigned by PIMA, as described in 9.5. 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. If exactly one set of vendor extensions is supported, the device would not support the PTP v1.1 GetVendorExtensionMaps or GetVendorDeviceInfo operations, and this value would be interpreted as previously defined in PTP v1.0 (i.e. as the “one and only” “VendorExtensionID”). If multiple vendor extension sets are supported, those operations would both be supported, and this field is interpreted as the default, or “native” set, with all other supported sets being mapped into unused portions of the native set, as communicated to the initiator by the two new optional operations.

**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. Table 8 describes the standard functional modes.

The functional mode information is held by the device as a device property. This property is described in 13.5.2. 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.

**Table 8 — 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 |

**OperationsSupported:** This field is an array of OperationCodes representing operations that the device is currently supporting, given the FunctionalMode indicated. Subclause 10.3 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. Subclause 12.6 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. Subclause 13.5 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 in which a device can capture data 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.

**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.

### 5.5.3 ObjectInfo dataset

This dataset is used to define the information about data objects in persistent storage, 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. See Table 9.

**Table 9 — 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     | Date Time String | No             | No                   |
| ModificationDate     | 18          | Variable     | Date Time String | No             | No                   |
| Keywords             | 19          | Variable     | String           | No             | No                   |

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

**ObjectFormat:** indicates ObjectFormatCode of the object. See 6.3 for a list of these codes.

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

**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 International Standard without first separately removing their protection status in a separate transaction. The values are enumerated according to Table 10.

**Table 10 — 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 from the AccessCapability field present in the StorageInfo dataset described in 5.5.4. If an attempt is made to delete an object, 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 5.5.4 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 6.3 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 are 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 are 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 are 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 colour 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.

used. This field should not be used for memory allocation purposes, but is strictly information that is typically inside an image object that may affect whether or not a user wishes to transfer the image, and is therefore 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 6.5 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 association. If unused, this field should be set to 0x00000000. Refer to 6.5 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 5.5.4 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 5.3.5.2.

**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 5.3.5.2.

**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.

#### 5.5.4 StorageInfo dataset

This dataset is used to hold the state information for a storage device. See Table 11.

Table 11 — 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. See Table 12.

All undefined values are reserved for future use.

**Table 12 — Storage types**

| Code value       | Storage type  |
|------------------|---------------|
| 0x0000           | Undefined     |
| 0x0001           | Fixed ROM     |
| 0x0002           | Removable ROM |
| 0x0003           | Fixed RAM     |
| 0x0004           | Removable RAM |
| All other values | Reserved      |

**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. See Table 13. If the storage device is DCF-conformant, it shall be so indicated here.

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.

**Table 13 — 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       |

**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 permitted values are described in Table 14. Read-write is only valid if the StorageType is non-ROM, as described in the StorageType field in Table 12.

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.

**Table 14 — StorageInfo AccessCapability values**

| Value            | Description                       |
|------------------|-----------------------------------|
| 0x0000           | Read-write                        |
| 0x0001           | Read-only without object deletion |
| 0x0002           | Read-only with object deletion    |
| All other values | Reserved                          |

**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 in 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.

### 5.5.5 VendorExtensionMap dataset (PTP v1.1)

This dataset is an optional dataset defined in PTP v1.1, that is used to support multiple vendor extensions. This dataset contains the mapping information for a single supported 16-bit (i.e. UINT16) vendor extended datacode of any of the standard types (Operation, Response, Event, DeviceProperty and ObjectFormat Codes). The mutual exclusion of all code category types means that the type of category may be determined from inspecting the value of the code as described in Table 2. All native and mapped code values should have the MSB (Most Significant Bit) set to 1 to indicate vendor extended status in the typical fashion of all vendor extended codes in this International Standard. See Table 15.

These datasets are typically communicated in a dataset array, as the response to the GetVendorExtensionMaps operation, which returns an entire mapping of one or more additional sets or subsets of vendor extensions, for devices that support multiple vendor extension sets, as described by Table 24. When used in this fashion, the “version” of the vendor extension set that is used for interpretation for a particular code is specified in the DeviceInfo dataset retrieved for that particular extension set (i.e. using the new optional GetVendorDeviceInfo operation in 10.5.33).

**Table 15 — VendorExtensionMap dataset (PTP v1.1)**

| Dataset field           | Field order | Length (bytes) | Data type |
|-------------------------|-------------|----------------|-----------|
| NativeCode              | 1           | 2              | UINT16    |
| MappedCode              | 2           | 2              | UINT16    |
| MappedVendorExtensionID | 3           | 4              | UINT32    |

**NativeCode:** The datacode in the “unused” portion of the “default” VendorExtension space, as determined by the responder. This is a single UINT16 Operation, Response, Event, ObjectFormat, or DeviceProp datacode with the vendor extended most significant bit 15 set to one, indicating its vendor-extended status. The specific type of datacode in the NativeCode field must match the type of datacode in the following MappedCode field of the same VendorExtensionMap dataset instance.

**MappedCode:** The datacode of the non-default VendorExtension code in the “native” value with respect to its VendorExtensionID, as defined in that vendor extension specification. This is a single UINT16 Operation, Response, Event, ObjectFormat or DeviceProp datacode with the vendor extended most-significant bit 15 set to one, indicating its vendor extended status. The specific type of datacode in the MappedCode field must match the type of datacode in the proceeding NativeCode field of the same VendorExtensionMap dataset instance.

**MappedVendorExtensionID:** The VendorExtensionID (defined for the default set of vendor extensions and for PTP v1.0 devices by the UINT32 field in the standard DeviceInfo dataset) for which this particular datacode is in context.

### 5.5.6 ObjectFilesystemInfo dataset (PTP v1.1)

See Table 16.

This optional dataset is a subset of the standard ObjectInfo dataset, containing only the fields required to minimally characterize most basic filesystems. This dataset is typically retrieved in an array using the optional GetFilesystemManifest operation to obtain, in a single transaction, all of the filesystem information available from a responder, while avoiding any image-specific metadata that may require time to acquire. Objects that support this optional operation must still support the standard GetObjectInfo operation, which still can and will be called subsequently for more information on particular objects at the initiator's discretion, as well as PTP v1.0 or performance insensitive initiators.

**Table 16 — ObjectFilesystemInfo dataset (PTP v1.1)**

| Dataset field          | Field order | Size (bytes) | Datatype         | Assoc. specific | Equivalent ObjectInfo dataset field |
|------------------------|-------------|--------------|------------------|-----------------|-------------------------------------|
| ObjectHandle           | 1           | 4            | ObjectHandle     | No              | N/A                                 |
| StorageID              | 2           | 4            | StorageID        | No              | 1                                   |
| ObjectFormat           | 3           | 2            | ObjectFormatCode | No              | 2                                   |
| ProtectionStatus       | 4           | 2            | UINT16           | No              | 3                                   |
| ObjectCompressedSize64 | 5           | 8            | UINT64           | No              | 4 <sup>a</sup>                      |
| ParentObject           | 6           | 4            | ObjectHandle     | No              | 12                                  |
| AssociationType        | 7           | 2            | AssociationCode  | Yes             | 13                                  |
| AssociationDesc        | 8           | 4            | AssociationDesc  | Yes             | 14                                  |
| SequenceNumber         | 9           | 4            | UINT32           | No              | 15                                  |
| Filename               | 10          | Variable     | String           | No              | 16                                  |
| ModificationDate       | 11          | Variable     | DateTime String  | No              | 18                                  |

<sup>a</sup> The ObjectCompressedSize64 field is equivalent to the standard ObjectCompressedSize field in the ObjectInfo dataset, except that it uses a 64-bit integer in order to support objects of potentially very large size.

### 5.5.7 StreamInfo dataset (PTP v1.1)

See Table 17.

**Table 17 — StreamInfo dataset (PTP v1.1)**

| Dataset field    | Field order | Size (bytes) | Datatype |
|------------------|-------------|--------------|----------|
| DatasetSize      | 1           | 8            | UINT32   |
| TimeResolution   | 2           | 8            | UINT32   |
| FrameHeaderSize  | 3           | 4            | UINT32   |
| FrameMaxSize     | 4           | 4            | UINT32   |
| PacketHeaderSize | 5           | 4            | UINT32   |
| PacketMaxSize    | 6           | 4            | UINT32   |
| PacketAlignment  | 7           | 4            | UINT32   |

**TimeResolution:** Contains the value in nanoseconds of one unit of the time stamp in the frame header. Thus, to calculate the time interval between two frames the following formula is used:

$$\text{ElapsedTime} = (\text{TimeStamp2} - \text{TimeStamp1}) \times \text{TimeResolution} [\text{ns}]$$

**FrameHeaderSize:** Contains the size, in bytes, of the frame header. The initiator should read the frame payload at this offset. For this specification, the size of the header is 8 bytes. This can be changed in future versions; therefore, this header will point to the payload, thus enabling the existing clients to work with the new protocol.

**FrameMaxSize:** The maximum size in bytes of the frame the initiator can expect including the header. The responder should not send frames larger than this. This is not the same as the entire length of the video stream, but is a format-specific construct abstracted for use in PTP. This will always be a fixed number.

**PacketHeaderSize:** Contains the size in bytes of the packet header. The initiator should read the packet payload at this offset. This is provided in order to enable future extensions to this protocol and make sure that the existing clients will work properly.

**PacketMaxSize:** The maximum size in bytes of the packet the initiator can expect including the header. The responder should not send packets larger than this. If the frame is built up of multiple packets then all packets excluding the last should be of the same PacketMaxSize.

**PacketAlignment:** The number of bytes to which the packet should be aligned. The alignment may be required by some transport or device responder implementations in order to achieve optimal performance.

## 6 Image and data object formats

### 6.1 Object usage

A data object is defined as an image or other type of data that exists in persistent storage of a DSPD or other device. This International 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 International 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 2.2, the preferred format,
- one of the permitted image file formats listed in Table 18 or
- a proprietary image file format.

All devices conforming to this International Standard which 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 5.5.3. The setting of 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 the transfer of non-standard file formats.

Table 18 — 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)      |
| 0x300D                                                      | A    | QuickTime      | Apple QuickTime Video Format                     |
| 0x300E                                                      | A    | XML            | eXtensible Markup Language file                  |
| 0x3800                                                      | I    | Undefined      | Unknown image object                             |
| 0x3801                                                      | I    | Exif/JPEG      | Exchangeable File Format, JEITA 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 Image Technology       |
| 0x380F                                                      | I    | JP2            | JPEG2000 Baseline File Format                    |
| 0x3810                                                      | I    | JPX            | JPEG2000 Extended File Format                    |
| 0x3811                                                      | I    | DNG            | Digital Negative Format (PTP v1.1)               |
| 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) |      |                |                                                  |

## 6.2 Thumbnail formats

### 6.2.1 Permitted 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.

### 6.2.2 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 10918-1:1994. This requires that the image file use the following:

- DCT-based image compression process;
- 8-bit samples within each component;
- a 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 the following:

- square pixel sampling;
- a single interleaved scan (e.g. Y, Cr, Nb interleaved blocks);
- sRGB colour space as defined in IEC 61966-2-1;
- 3 colour components, Y, Cr, Nb, derived from the sRGB signals as follows, as specified in ITU-R BT.601<sup>1)</sup>:
  - $Y = 0,299 R + 0,587 G + 0,114 B$ ,
  - $Nb = (-0,299 R - 0,587 G + 0,886 B) \times 0,564 + \text{offset}$ ,
  - $Cr = (0,701 R - 0,587 G - 0,114 B) \times 0,713 + \text{offset}$ ,
  - or a single (luminance) component;
- either 4:2:2 or 4:2:0 Y:Nb:Cr spatially centred colour subsampling or Y:Nb:Cr co-sited subsampling. Writers may use either option. Readers shall support both options.

NOTE These features are all requirements of the Exif version 2.2 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 2.0. These requirements include the following:

- JPEG compression using 4:2:2 chrominance sampling and the “typical” Huffman table provided in ISO/IEC 10918-1:1994;
- 160 × 120 pixel image record with a 4:3 aspect ratio.

---

1) Digital Video Standard (formerly CCIR 601).

### 6.2.3 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. The thumbnail image shall have a compression tag value = 1 (no compression) with the thumbnail image contained within IFD0.

## 6.3 ObjectFormatCodes

A list of object formats is given in Table 18. An ObjectFormatCode is a 16-bit unsigned integer (UINT16) 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.

## 6.4 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.

## 6.5 Data object association

### 6.5.1 Association usage

This International 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 shall 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.

### 6.5.2 Association sub-types

There are different sub-types of associations. The sub-type is specified in the AssociationType field of the ObjectInfo dataset of the association object. Table 19 lists the various associations that may be used.

**Table 19 — Association sub-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 sub-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 sub-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 International Standard and shall be set to 0x00000000 for devices that support this version of the specification.

**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, in order 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; e.g. four images would have SequenceNumbers assigned as shown in Figure 1.

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|

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; e.g. four images would have SequenceNumbers assigned as shown in Figure 2.

|   |
|---|
| 4 |
| 3 |
| 2 |
| 1 |

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; e.g. sixteen images arranged in a 4 × 4 2DPanoramic would have SequenceNumbers assigned as shown in Figure 3.

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

Figure 3 — 2DPanoramic SequenceNumber example

**AncillaryData:** indicate that the association represents one or more non-image objects being associated with an image object; e.g. 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 SequenceNumbers for those objects should be set to 0x00000000. In this case, the AssociationDesc fields for those objects are unused, and should be set to 0x00000000.

### 6.5.3 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.

### 6.5.4 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 need 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 shall be unique.

### 6.5.5 Associations as filesystem folders

Unordered associations may be used to represent filesystem 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, see D.2.

## 7 Transport requirements

### 7.1 Disconnection events

The transport shall be capable of notifying the overlying agents or system software that a 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 the 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 of discovering and enumerating connected devices. Any transport may support a v1.0 or v1.1 PTP implementation.

### 7.5 Specific transports

#### 7.5.1 USB

Devices using this transport shall conform to the following specifications:

- *USB Specification*, Version 1.1, USB Implementer's Forum, 23 September, 1998<sup>[2]</sup>;
- *Universal Serial Bus Still Image Capture Device Definition*, Version 1.0, USB Implementer's Forum, 11 July, 2000<sup>[3]</sup>.

#### 7.5.2 IEEE 1394

Devices using the IEEE 1394 transport shall conform to the following specifications:

- IEEE 1394-1995<sup>[4]</sup>, *IEEE Standard for High Performance Serial Bus*;
- PTP-on-SBP (*Picture Transfer Protocol on Serial Bus Protocol*), IEEE 1394 Trade Association<sup>[6]</sup>.

#### 7.5.3 TCP/IP

Devices using the transport TCP/IP shall conform to the following specifications:

- *Transmission Control Protocol*, Internet Engineering Task Force (IETF) RFC 793<sup>[13]</sup>, by Information Sciences Institute, University of Southern California, September 1981; <http://ietf.org/rfc/rfc793.txt>;
- *PTP-IP IP Picture Transfer Protocol*, Version 1.0, Camera & Imaging Products Association (CIPA)<sup>[14]</sup>, Japan, Copyright 2005 by FotoNation Inc.

## 8 Persistent storage

### 8.1 StorageID

Every local persistent 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 sixteen least significant bits represent a logical storage area within a physical store.

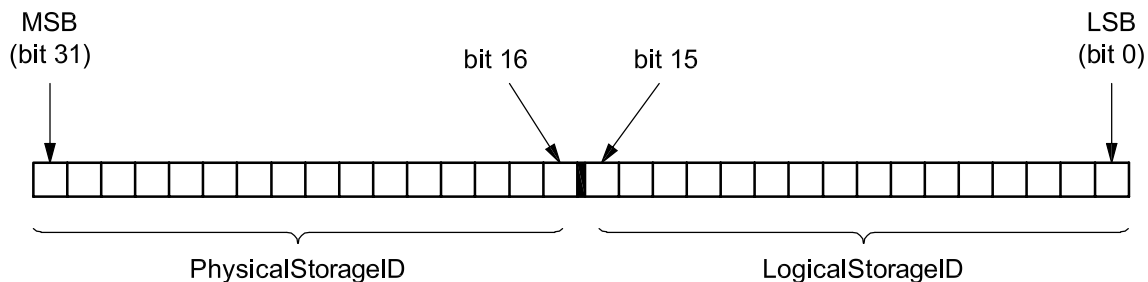


Figure 4 — StorageID Layout

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 16 bits shall be identical). PhysicalStorageIDs are unique at all times for any particular device. In cases where there is more than one logical store within a physical store, all LogicalStorageIDs (i.e. the lower 16 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

##### 8.2.1.1 Interoperability

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, see 5.3.3.

##### 8.2.1.2 ObjectHandle assignment

All conforming DSPDs shall allow data object referencing via the following behaviour.

- ObjectHandles shall be globally unique across all physical and logical storage areas on the device.
- The values 0x00000000 and 0xFFFFFFFF shall be considered undefined or have context-specific meanings, and shall not be assigned to any valid object.
- 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 in 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 which does not conflict with any other currently assigned handles;
  - 2) broadcast an appropriate ObjectAdded event to all open sessions in accordance with Clause 12; 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 broadcast an appropriate ObjectRemoved event to all open sessions in accordance with Clause 12. If the data object is deleted as a 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 which 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 permitted values, see 5.5.4. For a description of individual data object protection, refer to the description of the ObjectInfo dataset in 5.5.3.

### 8.3 Receiver object placement

The object receiver shall 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 in which it will be placed 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 behaviour. 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.

- a) The initiator does not specify receiver location for the 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.
- b) 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:
  - 1) 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;