ISO/IEC JTC 1/SC 29

Date: 2003-12-12

ISO/IEC FCD 2.0 15444-9 Study Text

ISO/IEC JTC 1/SC 29/WG 1

Secretariat: JISC

Information technology — JPEG 2000 image coding system — Part 9: Interactivity tools, APIs and protocols

Technologie de l'information — JPEG 2000 système de codage d'image — Partie 9: Outils d'interactivité, APIs et protocoles

Error! AutoText entry not defined.

Document type: **Error! Reference source not found.**
Document subtype: **Error! Reference source not found.**
Document stage: **Error! Reference source not found.**
Document language: **Error! Reference source not found.**

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national 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 and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 15444–9 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of Audio, Picture, Multimedia and Hypermedia Information*.

ISO/IEC 15444 consists of the following parts, under the general title *Information technology — JPEG 2000 image coding system*:

— *Part 1: Core coding system*

— *Part 2: Extensions*

— *Part 3: Motion JPEG 2000*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Compound image file format*

— *Part 8: Secure JPEG 2000*

— *Part 9: Interactivity tools, APIs and protocols*

— *Part 10: 3-D and floating point data*

— *Part 11: Wireless*

— *Part 12: ISO base media file format*

Introduction

ITU–T Rec. T.800 | ISO/IEC 15444–1 (JPEG 2000) is a specification that describes an image compression system that allows great flexibility, not only for the compression of images but also for access into the codestream. The codestream provides a number of mechanisms for locating and extracting portions of the compressed image data for the purpose of retransmission, storage, display, or editing. This access allows storage and retrieval of compressed image data appropriate for a given application without decoding.

The purpose of this International Standard is to provide a network protocol that allows for the interactive and progressive transmission of JPEG 2000 coded data and files from a server to a client. This protocol allows a client to request only the portions of an image (by region, quality or resolution level) that are applicable to the client's needs. The protocol also allows the client to access metadata or other content from the file.

Any organization contemplating the use of this International Standard should carefully consider the constraints on their applicability.

Information technology — JPEG 2000 image coding system — Part 9: Interactivity tools, APIs and protocols

1 Scope

This International Standard defines, in an extensible manner, syntaxes and methods for the remote interrogation and optional alteration of JPEG 2000 codestreams and files in accordance with their definition in the following parts of ISO/IEC 15444:

— ITU–T Rec. T.800 | ISO/IEC 15444–1:2002 and its definition of a JPEG 2000 codestream and JP2 file format.

— the JPEG 2000 family of file formats as defined in further parts of ISO/IEC 15444.

In this International Standard, the defined syntaxes and methods are referred to as the JPEG 2000 Interactive Protocol, “JPIP”, and interactive applications using JPIP are referred to as “JPIP systems.”

JPIP specifies a protocol consisting of a structured series of interactions between a client and a server by means of which image file metadata, structure and partial or whole image codestreams may be exchanged in a communications efficient manner. This International Standard includes definitions of the semantics and values to be exchanged, and suggests how these may be passed using a variety of existing network transports.

With JPIP, the following tasks may be accomplished in varying, compatible ways:

— the exchange of capabilities

— the negotiation of capabilities to use in a session

— the request and transfer of the following elements from a variety of containers, such as JPEG 2000 family files, JPEG 2000 codestreams and other container files

  — selective data segments

  — selective and defined structures

  — parts of an image or its related metadata

Further, JPIP provides the capability for ‘fallback,’ such that the protocol can deliver similar results using differing levels of awareness of JPEG 2000 file and codestream structures at the client and the server. JPIP can be used over a variety of networks and communications media having different characteristics and quality of service characteristics. It can use a number of methods to communicate between client and server, based on existing protocols and network transports, which this International Standard extends to provide additional JPEG 2000 related functionality. Information that is user or session related can also be exchanged.

JPIP can be tailored via the various extensions to the JPEG 2000 file format, as defined in ITU–T Rec. T.801 | ISO/IEC 15444–2, ISO/IEC 15444–3 and ISO/IEC 15444–6. However, to achieve a simple level of interactivity that allows portions of a single JPEG 2000 file or codestream to be transferred, these other capabilities are not mandated.

Although the terms ‘client’ and ‘server’ are typically used in this International Standard to refer to the image receiving and delivering applications respectively, it is intended that JPIP can be used within both hierarchical and peer to peer networks, for data delivery in either direction, and for machine to machine as well as user to machine or user to user applications. It is also intended for use as an adjunct to an alternative, more comprehensive, protocol for image delivery, and for the delivery of non-JPEG 2000 coded information. While some features of JPIP may be applied to codestreams that are not JPEG 2000 compliant, such use is not mandated or required by JPIP systems conforming to this International Standard.

The use of JPIP in an Internet or intranet environment is addressed. Further facilities and features may be available when JPIP is used on top of a network transport protocol such as TCP/IP, UDP or HTTP.

In addition, this International Standard defines two media-types: JPP-stream and JPT-stream.

2 Conformance

[Editor’s note: Add paragraph.]

3 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.

ITU–T Rec. T.800 | ISO/IEC 15444–1:2002, JPEG 2000 Image Coding System 2nd edition

ITU–T Rec. T.801 | ISO/IEC 15444–2:2001, JPEG 2000 Image Coding System: Extensions

ISO/IEC 15444–3:2002, JPEG 2000 Image Coding System: Motion JPEG 2000

ISO/IEC 15444–6:2003, JPEG 2000 Image Coding System: Compound Image File Format

IETF RFC 793, Transmission Control Protocol, September 1981

IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996

IETF RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996

IETF RFC 2616, Hypertext Transfer Protocol — HTTP/1.1, June 1997

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply. The definitions defined in ITU–T Rec. T.800 | ISO/IEC 15444–1:2000 Clause 3 and ITU–T Rec. T.801 | ISO/IEC 15444–2 Clause 3 also apply to this International Standard.

4.1 HTTP definitions

The following definitions are intended to match HTTP/1.1. In the case of any difference, these definitions should be used.

4.1.1
connection
A transport layer virtual circuit established between two programs for the purpose of communication.

4.1.2
entity
The information transferred as the payload of a request or response. An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body.

4.1.3
origin server
The server on which a given resource resides or is to be created.

4.1.4
proxy
An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers.

4.2 JPIP definitions

The following definitions are used within this standard. In some cases, these definitions differ from those used in other standards and/or recommendations.

4.2.1
cache (client-side)
The cache on the Client is the storage of the JPIP data-bins. The Client may have a limited cache and may have to purge cached JPIP data-bins from time to time.

4.2.2
cacheable
A response is cacheable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. Even if a resource is cacheable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

4.2.3
cache-model (server-side)
The server's estimation of the portions of the data-bins available to client. The server may add items to its estimation of the clients cache because it assumes successfully delivery or because it has received acknowledgements of transmitted data, or because of cache-model update statements.

4.2.4
channel
A mechanism for grouping requests and responses such that only one request/response is active at a time within the group. Multiple simultaneous requests and responses require multiple channels.

4.2.5
client
A program that establishes connections for the purpose of sending requests.

4.2.6
data-bin
A set of bytes of the same type of data which may be partially delivered.

4.2.7
JPIP index table
A file format box which provides information about the location of portions of a file or codestream.

4.2.8
logical target
A data object available from a server. Such targets have a unique identifier.

4.2.9
message
A set of bytes from a single data-bin and the header identifying those bytes and the data-bin.

4.2.10
request
A group of fields and values sent from the client to the server to obtain portions of an image or metadata.

4.2.11
response
The bytes sent from the server to the client after receiving a request.

4.2.12
resource
A network data object or service that can be identified by a URI. A HTTP target.

4.2.13
server
An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general.

4.2.14
session
A collection of requests and responses applying to the same resource for which the server maintains a cache model.

4.2.15
stateful
Where the server maintains a cache model.

4.2.16
stateless
A single request where the server does not make use of a cache-model in determining the response.

4.2.17
target
The logical identification of JPIP data.

Note: JPEG2000 files or codestreams may be available in multiple representations (e.g. return type, precinct size) or vary in other ways, each identified as a unique logical target.

4.2.18
tile header
All tile-part headers for a specific tile.

4.2.19
view-window
The portion of the image data the client desires. The view-window is often smaller than the whole image as indicated by a spatial offset, a limited extent, or a lower resolution.

5 Symbols (and abbreviated terms)

For the purposes of this International Standard, the following symbols apply. The symbols defined in ITU–T Rec. T.800 | ISO/IEC 15444–1 Clause 4, ITU–T Rec. T.801 | ISO/IEC 15444–2 Clause 4 and HTTP/1.1 also apply to this International Standard.

BNF: Backus-Naur Form

JP3D: JPEG 2000 Part 10: 3-D and floating point data

JPIP: JPEG 2000 Interactive Protocol

JPP: JPIP Precinct

JPSEC: JPEG 2000 Part 8: Secure JPEG 2000

JPT: JPIP Tile-part

JPWL: JPEG 2000 Part 11: Wireless

SVG: Scalable Vector Graphics

UUID: Universal Unique Identifier

VBAS: Variable-length Byte Aligned Segment

XML: Extensible Markup Language

6 General description

6.1 JPIP protocol

This International Standard describes the syntaxes and methods that are used when a client is accessing JPEG 2000 compressed imagery and imagery related data residing on a JPIP-enabled server. This International Standard enables the flexibility and functionality intended in ITU–T Rec. T.800 | ISO/IEC 15444–1:2002 to be realized across multiple client/server transports.

JPIP defines the interactive protocol to achieve the efficient exchange of JPEG 2000 imagery and imagery related data. The protocol defines the Client-Server interactions based on a client request and server response as shown in Figure 1. The client uses a View-Window request to define the resolution, size, location, components, layers, and other parameters for the image and imagery related data that is requested by the client. The server response delivers imagery and imagery related data with precinct-based streams, tile-based streams, or whole images. The protocol also allows for the negotiation of client and server capabilities and limitations. The client may request information about an image as defined in index tables from the server, which enables the client to refine its View-Window request to image specific parameters (e.g., byte range requests). The server's cache model is based on the capabilities defined by the client and the statefulness of the session.

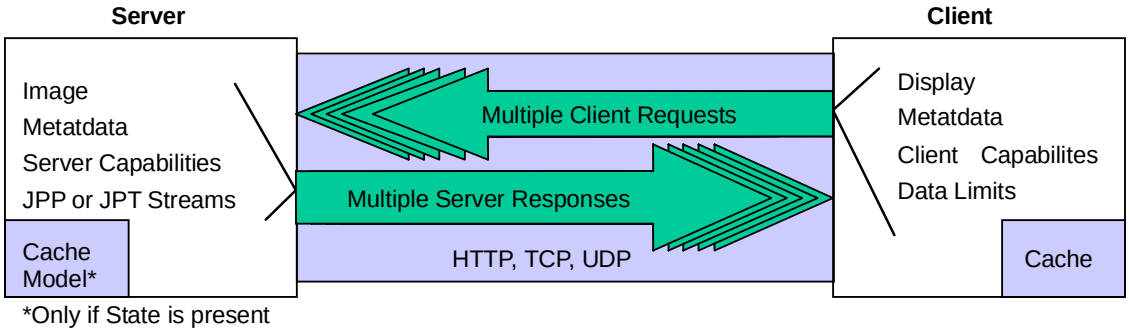

Figure 1 — JPIP Protocol overview

This protocol can be used over several different transports as shown in Figure 2. This standard includes informative annexes on the use of JPIP Protocol over HTTP and TCP, and provides suggestions for other example implementations.

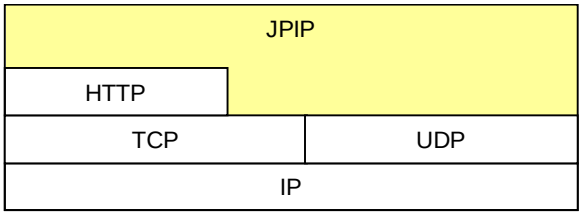

Figure 2 — JPIP protocol stack

Provisions have been included for the extension of the JPIP protocol to support the current JPEG 2000 Standards ISO/IEC 15444–3 Motion JPEG 2000 and ISO/IEC 15444–6 Compound Documents and the future parts of JPEG 2000 (currently JP3D, JPSEC, and JPWL).

6.2 Purpose

This International Standard defines the syntax and methods required for both the client and server. Each Annex defines a component that is required to achieve interoperability and functionality between the client and server over several transports. Each Annex may be a requirement of the client, server, or both and are described below.

— Annex A describes the tile based and precinct based streams that are required for both the client and the server. The server is required to produce compliant JPP- and JPT-streams and understand uploaded JPP- and JPT-streams. The client is required to understand and properly decode these streams and is responsible for producing compliant streams when uploading partial imagery to the server.

— Annex B describes the session and cache modelling of a client/server session and is required for both the client and server.

— Annex C defines the client request syntax. The client shall produce compliant versions of this and the server shall be able to understand and respond to all compliant-requests.

— Annex D defines the server response syntax. The server shall produce compliant versions of this and the client shall be able to understand compliant responses.

— Annex E defines syntax and methods to upload a partial image for systems which use JPIP for upload.

— Annex F, Annex G, and Annex H define the methods and procedures for JPIP client/server interactions over several different transport protocols.

— Annex I defines the indexing information syntax contained in a JPEG 2000 box that can be used by a client and server to more efficiently access imagery and imagery related data.

— Annex J describes several examples of using this Recommendation | International Standard for several different applications.

— Annex K describes an example of an API implementation of the JPIP system.

7 Common BNF definitions

7.1 General

The following BNF definitions are used and referenced throughout this International Standard.

7.2 Integers and character sets

```
LOWER   = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" |
          "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" |
          "u" | "v" | "w" | "x" | "y" | "z"

UPPER   = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" |
          "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" |
          "U" | "V" | "W" | "X" | "Y" | "Z"

ALPHA   = LOWER | UPPER

DIGIT   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

NZDIGIT = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

HEXDIGIT = DIGIT
         | "A" | "B" | "C" | "D" | "E" | "F"
         | "a" | "b" | "c" | "d" | "e" | "f"

UINT    = 1*DIGIT

NONZERO = 0*"0" NZDIGIT 0*DIGIT

UINT-RANGE = UINT ["-" [UINT]]

UFLOAT = 1*DIGIT ["." 1*DIGIT]

ENCODED-CHAR = "%" HEXDIGIT HEXDIGIT

CR = 0x0D

LF = 0x0A

CRLF = CR LF

LWS = Optional linear white space

UUID = 16(HEXDIGIT)

TOKEN= 1*(ALPHA | DIGIT | "/" | "." | "_")
```

[Editor's note: Should it be "\" | "/" | "." | "_" – i.e. include forward and backward slash?]

UINT-RANGE specifies a range of integer values. The first integer in the range specifies the beginning of the range. If two values are specified, the first and second values specify the inclusive beginning and ending limits to the range. If only the first value and the "-" character are specified, the range includes all values greater than or equal to the first value.

A numerical value immediately preceding a BNF element refers to a repetition of the parameter that follows the number for the number of times given by the numerical value, with no intervening spaces between each occurrence.

The construct “1#” refers to one or more repetitions of the parameter that follows, each occurrence of which is separated by a comma.

The construct “*n**” refers to a repetitions of the parameter that follows with no intervening spaces between each occurrence, where there are at least *n* occurrences of that parameter. *n* may be zero.

The construct “1$” refers to one or more repetitions of the parameter that follows, each occurrence of which is separated by a semicolon.

The construct “*n*m*” refers to a repetition of the parameters that follows with no intervening spaces between each occurrence, where there are at least *n* occurrences and no more than *m* occurrences.

The construct “0*” means zero or more repetitions of the parameter that follows, with no intervening spaces between each occurrence.

7.3 File format elements

```
compatibility-code = 4(ALPHA | DIGIT | "_" | ENCODED-CHAR)

box-type = 4(ALPHA | DIGIT | "_" | ENCODED-CHAR)

box-type-list = "*" | 1#(box-type)

box-desc = box-type ["[" UINT "]"]
```

`box-type` specifies the four characters of the box type. For each character in the box type, if the character is alpha-numeric (A..Z, a..z or 0..9), the character is written directly into the string. If the character is a space (0x20), then that character shall be encoded as the underscore character (“_”). For any other character, a 3-character string is written in its place, consisting of an percent character (“%”) followed by two hexadecimal digits representing the value of the character from the box type in hexadecimal. The `compatibility-code` is encoded the same way that a `box-type` is encoded.

`box-type-list` specifies a list of box types. If the value of a `box-type-list` field is “*”, then the field refers to all box types.

`box-desc` describes a specific box at a given level of the logical target, by specifying the type of that box and the count of how many boxes of that type are found before the box to be described. For example, the 3rd association box at a given level in the logical target would be described as “asoc[2]”. If the count value is 0, the count may be omitted. For example, the first XML box may be described as either “xml_[0]” or “xml_”.

Annex A
(normative)

The JPP-stream and JPT-stream media types

A.1 Introduction

JPP-stream and JPT-stream are media types useful for presenting JPEG 2000 codestreams and file format data in an arbitrary order. Each media type consists of a concatenated sequence of messages, where each message contains a portion of a single data-bin preceded by a message header. Data-bins contain portions of a JPEG 2000 compressed image representation, such that it is possible to construct a stream that completely represents the information present in a JPEG 2000 file or codestream. Each message is completely self-describing, so that the sequence of messages may be terminated at any point and messages may be re-ordered subject to minimal constraints without losing their meaning. For these reasons, JPP-stream and JPT-stream media types are useful for JPIP servers and the JPIP protocol is designed with these media types particularly in mind. This Annex defines the JPP-stream and JPT-stream media types without reference to the JPIP protocol.

A.2 Message Header Structure

A.2.1 General

Each message represents a portion of exactly one data-bin. The message header consists of a sequence of variable-length byte-aligned segments (VBAS). Each VBAS consists of a sequence of bytes, all but the last of which has a most significant bit (bit 7) of 1, as seen in Figure A.1. The least significant 7 bits of each VBAS are concatenated to form a bit stream which is used in different ways for different VBAS's.

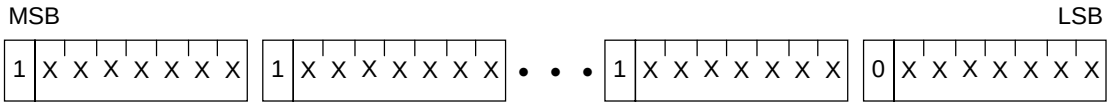

Figure A.A1 — VBAS structure

The message header serves to identify the particular data-bin and byte range which is represented by the message body. Message headers can take an independent form and a dependent form. The independent form is a long form where the message headers are completely self-describing; their interpretation is independent of any other message headers. The optional shorter dependent form message headers make use of information in the headers of previous messages; their decoding is dependant on the previous message. Applications may choose to use the long form message headers; these messages can be rearranged in any arbitrary order. Alternatively, applications may use the shorter form message headers that do depend on previous message headers; these are shorter messages but will create erroneous results if the messages are not arranged in the correct sequence when decoded. It is an application decision whether or not the sequence ordering of received messages can be assumed to be reliable, and if so, whether to make use of the shorter form message headers.

The message header consists of the following VBAS's (optional VBAS's identified by the use of square brackets):

Bin-ID [, Class] [, CSn], Msg-Offset, Msg-Length [, Aux]

The existence of the Class and CSn VBASs are determined by examining the Bin-ID VBAS. The existence of the Aux VBAS is determined by the Class VBAS or the previous Class VBAS, if there is no Class VBAS in the current message header.

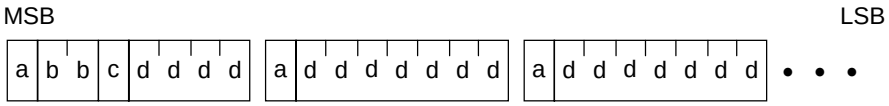

Figure A.A2 — Bin-ID VBAS structure

The Bin-ID VBAS serves several roles. Bits 6 & 5 of the first byte of the Bin-ID VBAS, labelled ‘b’ in Figure A.2, indicate whether the Class and CSn VBASs are present in the message header.

Bit 4 of the first byte of the Bin-ID VBAS, labelled ‘c’ in Figure A.2, indicates whether or not this message contains the last byte in the associated data-bin: ‘0’ means it is not the last byte in the data-bin; ‘1’ indicates that it is the last byte in the data-bin. Receiving a message with this bit set allows determination of the length of the complete data-bin, although it does not imply that the complete JPP-stream or JPT-stream contains sufficient messages to assemble all of the bytes from that data-bin.

The remaining 4 bits of the first byte and the 7 low order bits of any remaining bytes in the Bin-ID VBAS (labelled `d’ in Figure A.2) form an “in-class identifier”, which is used to uniquely identify the data-bin within its class, in the manner described in Annex A.2.3.

Table A.A1 — Bin-ID Additional VBAS indication

Indicator Bits ‘bb’	Meaning
00	Prohibited.
01	No Class or CSn VBAS is present in message header
10	Class VBAS is present but CSn is not present in message header
11	Class and CSn VBAS are both present in the message header.

The Class VBAS, if present, provides a message class identifier. The message class identifier is a non-negative integer, formed by concatenating the least significant 7 bits of each byte of the VBAS in big-endian order. If the Class VBAS is not present, the message class identifier is unchanged from that associated with the previous message. If the Class VBAS is not present and there is no previous message, the message class identifier is 0. Valid message class identifiers are described in Annex A.2.2.

The CSn VBAS, if present, identifies the index (starting from 0) of the codestream to which the data-bin belongs. The codestream index is formed by concatenating the least significant 7 bits of each byte of the VBAS in big-endian order. If the CSn VBAS is not present, the codestream index is unchanged from the previous message. If CSn VBAS is not present and there is no previous message, the codestream index is 0.

The Msg-Offset and Msg-Length VBAS’s each represent integer values, formed by concatenating the least significant 7 bits of each byte in the VBAS in big-endian order. The Msg-Offset integer identifies the offset of the data in this message from the start of the data-bin. The Msg-Length integer identifies the total number of bytes in the body of the message.

An auxiliary VBAS may be present. Its presence, and meaning if present, are determined by the message class identifier found within the Bin-ID VBAS, as explained in the Annex A.2.2. If present, the auxiliary VBAS represents an integer, formed by concatenating the least significant 7 bits of each byte in the VBAS in big-endian order.

NOTE The information in the auxiliary VBAS cannot affect the length of the message body.

A.2.2 Message Class Identifiers

The message class identifiers defined by this document are the non-negative integers shown in the Table A.2. The interpretation of the data-bin classes to which they refer is described in Annex A.3. All other values of message class identifier are reserved, and the associated messages should be skipped by decoders not recognizing the value.

Class identifiers are chosen such that an auxiliary VBAS is present if and only if the identifier is odd. This property allows unrecognized message headers to be correctly parsed and the contents skipped.

Table A.A2 — Class identifiers for different data-bin message classes

Class Identifier	Message Class	Data-Bin Class	Stream Type
0	Precinct data-bin message	Precinct data-bin	JPP-stream only
1	Extended precinct data-bin message	Precinct data-bin	JPP-stream only
2	Tile header data-bin message	Tile header data-bin	JPP-stream only
4	Tile data-bin message	Tile data-bin	JPT-stream only
5	Extended tile data-bin message	Tile data-bin	JPT-stream only
6	Main header data-bin message	Main header data-bin	JPP- and JPT-stream
8	Metadata-bin message	Metadata-bin	JPP- and JPT-stream

Extended precinct data-bin messages have exactly the same interpretation as non-extended precinct data-bin messages and they refer to exactly the same precinct data-bins. The extended precinct messages include an auxiliary VBAS which identifies the number of complete packets (quality layers) which would be available for the precinct if the bytes in this message were combined with all previous bytes of the same precinct. If this message also contains the last byte of the data-bin, the auxiliary VBAS indicates the total number of quality layers associated with the precinct in the original codestream. Otherwise, the auxiliary VBAS indicates the quality layer to which the byte immediately following the last byte in the message belongs. The information in the auxiliary VBAS may be useful to certain clients.

Extended tile data-bin messages have exactly the same interpretation as non-extended tile data-bin messages and they refer to exactly the same tile data-bins. The extended tile messages include an auxiliary VBAS which identifies the number of complete resolutions starting from the lowest resolution which would be available for all components in the tile if the bytes in this message were combined with all previous bytes of the same tile. Thus the value 1 is used when the lowest resolution is complete, and the value is the number of wavelet transform levels plus one when all resolutions have been delivered. Because resolutions do not necessarily appear in order in a tile some resolutions above the value given in the VBAS may have been completed, but this cannot be determined from the message header. The information in the auxiliary VBAS may be useful to certain clients.

A.2.3 In-class identifiers

The least significant 4 bits of the first byte and the least significant 7 bits of all other bytes from the Bin-ID VBAS are concatenated in big-endian order to form a single word, having $7k-3$ bits, where k is the number of bytes in the VBAS. This word represents an unsigned integer which serves to uniquely identify the data-bin within its class and codestream. Annex A.3 provides a description of the various data-bin classes, along with the corresponding in-class identifiers.

A.3 Data-bins

A.3.1 Introduction

Data-bins contain portions of a JPEG2000 file or codestream data. These may be based on imagery elements, such as precint-based data, tile-based data, and headers. They may also be based on metadata. Whatever the content of a data-bin, each data-bin is treated as an individual bit-stream.

A.3.2 Precinct data-bins

A.3.2.1 Precinct data-bin format

Precinct data-bins appear only within the JPP-stream media-type. Each precinct data-bin corresponds to a single precinct within a single codestream. The in-class identifier, *I*, uniquely identifies the precinct within its codestream. It is defined by

$$I = t + (c + s \times \text{num\_components}) \times \text{num\_tiles}$$

where t is the index (starting from 0) of the tile to which the precinct belongs, c is the index (starting from 0) of the image component to which the precinct belongs, and *s* is a sequence number which identifies the precinct within its tile-component. Within each tile-component, precincts are assigned contiguous sequence numbers, s, as follows. All precincts of the lowest resolution level (that containing only the LL subband samples) are sequenced first, starting from 0, following a raster-scan order. The precincts from each successive resolution level are sequenced in turn, again following a raster-scan order within their resolution level.

It follows that a precinct identifier of 0 refers to the upper left hand precinct from the LL subband of image component 0 in tile 0.

Each precinct data-bin corresponds to the string of bytes formed by concatenating all codestream packets, complete with all relevant packet headers, which belong to the precinct. It is conceivable that packet headers will be packed into PPM or PPT marker segments which shall then belong to main header or tile header data-bins, in which case the precinct data-bin would hold only packet bodies. In any event, the precinct data stream should coincide with the contiguous segment of bytes that would be found within a JPEG 2000 codestream having one of the layer-subordinate progression sequences (CPRL, PCRL or RPCL).

A.3.2.2 Precinct data-bin example (informative)

Figure A.3 shows an example precinct data-bin (in-class identifier 3) with 4 quality layers (or packets)

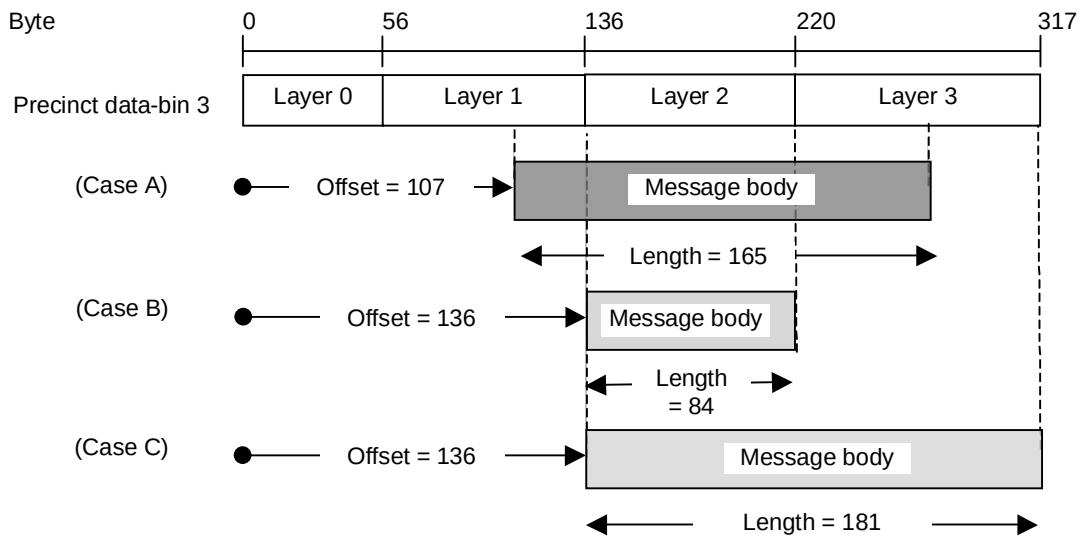

Figure A.A3 — Example precinct data-bin

For Case A, B and C, the message header is shown below, based on the extended and non-extended precinct data-bin message structures. The underlined data denotes the auxiliary VBAS to identify the number of layers which are completed by the message.

(Case A)

Non-extended header: `00100011 01101011 10000001 00100101 xxxxxxxx …`

The initial 0 bit indicates only one byte is used in the Bin-ID VBAS. The next two bits (“01”) indicate that no Class or CSn VBAS is present. The next “0” bit indicates that the data-bin is not completed by this message. The remaining bits of the first byte (“0011”) indicate that the bin-ID is 3. The first bit of the second byte indicates that there is only one byte used in the Msg-Offset VBAS. The next 7 bits (“1101011”) mean that the offset is 107. The first bit of the 3rd byte indicates that both this byte and at least the next byte are part of the Msg-Length VBAS. The 0 bit starting the 4th byte indicates that it is the last byte of the Msg-Length VBAS. Thus all the low order bits from the 3rd and 4th bytes are concatenated to determine the length. In this case, “0000001 0100101” = 165.

Extended header: `01000011 00000001 01101011 10000001 00100101 00000011 xxxxxxxx …`

(Case B)

Non-extended header: `00100011 10000001 00001000 01010100 xxxxxxxx …`

Extended header: `01000011 00000001 10000001 00001000 01010100 00000011 xxxxxxxx …`

(Case C)

Non-extended header: `00110011 10000001 00001000 10000001 00110101 xxxxxxxx …`

Extended header: `01010011 00000001 10000001 00001000 10000001 00110101 00000100 xxxxxxxx …`

Note that since the return data contains the last byte of the data-bin in Case C, the Bin-ID VBAS indicates that it is a "completed" message.

A.3.3 Tile header data-bins

Tile header data-bins appear only within the JPP-stream media type. For data-bins belonging to this class, the in-class identifier holds the index (starting from 0) of the tile to which the data-bin refers. This data-bin consists of markers and marker segments for tile n. It shall not contain an SOT marker segment. Inclusion of SOD markers is optional. This data bin may be formed from a legal codestream, by concatenating all marker segments except SOT and POC in all tile-part headers for tile n.

A.3.4 Tile data-bins

Tile data-bins shall be used only with the JPT-stream media type. For data-bins belonging to this class, the in-class identifier is the index (starting from 0) of the tile to which the data-bin belongs. Each tile data-bin corresponds to the string of bytes formed by concatenating all tile-parts belonging to the tile, in order, complete with their SOT, SOD and all other relevant marker segments.

A.3.5 Main header data-bin

Both JPP- and JPT-stream media types use the main header data-bin. For data-bins belonging to the codestream main header class (completed or non-completed variations), the in-class identifier shall be 0. This data-bin consists of a concatenated list of all markers and marker segments in the main header, starting from the SOC marker. It contains no SOT, SOD or EOC markers.

A.3.6 Metadata-bins

A.3.6.1 Introduction to metadata-bins

Both JPP- and JPT-stream media types use metadata-bins. Metadata-bins are used to convey metadata from the logical target that contains the codestream or codestreams whose elements may be referenced by other data-bins associated with the JPP-stream or JPT-stream. For the purpose of this document, the term “metadata” refers to any collection of “boxes” from a JPEG 2000 family file. The codestream index shall be ignored in any message which has the metadata-bin class identifier.

Unlike the numerical ID’s used for other types of data-bins, metadata-bin ID’s do not map algorithmically to some file format construct or byte offset. The server is free to choose any numeric ID for any particular metadata bin. The one and only one exception of this is that the metadata-bin containing the root of the logical target shall be given the ID of 0.

Note: The mechanism for assignment is implementation dependent, however, it is an informative suggestion that servers assign bin-ID’s using consecutive numbers.

A.3.6.2 Division of a logical target containing a JPEG 2000 file into metadata-bins

All metadata could conceivably be included in metadata-bin 0. In this case, all boxes from the logical target would belong to metadata-bin 0, appearing in their original order. Since JPEG 2000 family file formats consist of nothing but a sequence of boxes, this effectively means that metadata-bin 0 would consist of the entire logical target. More generally, however, it is useful to break the logical target into pieces that can be transmitted in a manageable fashion. This allows image servers to deliberately omit portions of the logical target that are not currently required by a client. To this end, JPIP defines a new special box type, known as the “Placeholder box.” The Placeholder box serves to identify the size and type of a box from the logical target, while pointing to another data-bin that holds that box’s contents. Placeholders are also able to represent codestreams from the logical target. This is particularly significant in view of the fact that the compressed data represented by any given codestream may be delivered incrementally via the other data-bin types (header data-bins and precinct data-bins or tile data-bins).

Formally, metadata-bin 0 consists of all boxes from the logical target, appearing in their original order, with the exception that a placeholder may replace any given box. The Placeholder box contains the original header of the box that has been replaced, together with the identifier of the metadata-bin that holds that box’s contents, not including the header itself. Every metadata-bin, other than metadata-bin 0, shall consist of the contents of

some box, whose header appears in the placeholder that references that data-bin. These box contents may themselves include sub-boxes, any of which may be replaced by further placeholders.

The following colour scheme will be used for metadata-bin example illustrations (Figure A.4):

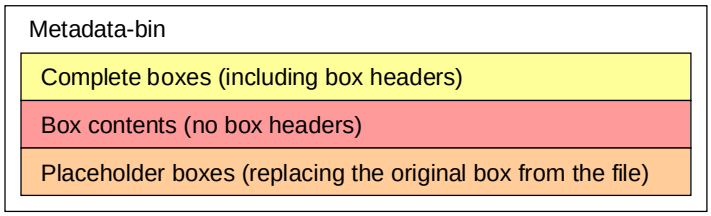

Figure A.A4 — Metadata-bin example colour scheme

As an example, consider a simple JP2 file with the following box structure (Figure A.5):

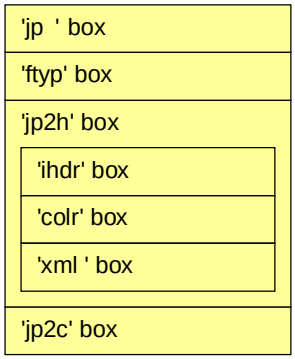

Figure A.A5 — A sample JP2 file

This file may be divided up into three metadata-bins: one to represent the top-level of the original file (data-bin 0); one to represent the JP2 Header box; and one to represent the codestream. This division is shown in Figure A.6.

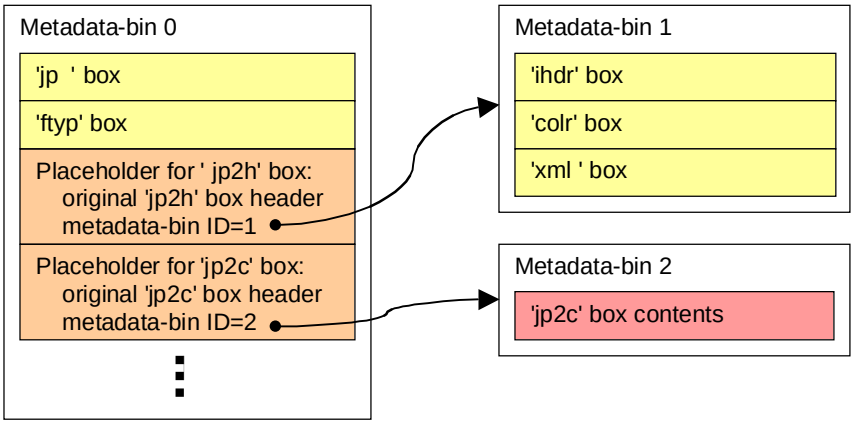

Figure A.A6 — A sample JP2 file divided into three metadata-bins

While the contents of any metadata-bin shall be the contents of the box or file represented by that bin, the actuall data contained in those contents may conceptually vary depending on the type of box. For example, in Metadata-bin 1 in Figure A.6, representing the contents of the JP2 Header box, the contents of that box is literally a series of other complete boxes, as the JP2 Header box is a superbox. No data other than the series of those complete boxes may be found within Metadata-bin 1, as there is no other data in the JP2 Header box.

In contrast, the data inside Metadata-bin 2 is the raw contents of the Contiguous Codestream box, with no box headers, because that box is not a superbox.

One point of particular interest to note from the example in Figure A.6 is that access to codestream data may be provided in two ways. The second placeholder bin is used to replace the contiguous codestream box (jp2c) in the original file. It identifies metadata-bin 2 as holding the original contents of this box, i.e., the raw codestream itself. The placeholder may also provide a codestream identifier. Any data-bins belonging to the main header, tile header, precinct or tile data-bin classes, having this same codestream identifier, convey compressed data associated with the same codestream as that found in metadata-bin 2. For convenience of description in this document, the terms “raw codestream” and “incremental codestream” will be used to distinguish between the representation which may appear in a metadata-bin (raw) and the representation which may appear in header, precinct and tile data-bins (incremental).

In general, placeholders that reference codestream data may do so either by referencing a separate metadata-bin (raw codestream), or by providing a codestream identifier (incremental codestream), or both. Even if both methods are provided, the JPP-stream or JPT-stream data available at a client or image-rendering agent might only have the contents of the raw codestream, or only have data from the incremental codestream. Moreover, if both the raw and incremental versions of the same codestream are available, there is no guarantee that the two representations will have compatible coding parameters. Only the reconstructed image samples associated with the two representations are guaranteed to be consistent.

It is also possible to use Placeholder boxes to associate multiple codestreams with a single original box. The interpretation of such an association is dependent upon the box being replaced. Further discussion of this topic appears in A.3.5.4.

In the simple example of Figure A.5, Placeholder boxes appear only at the top-level of the file, in metadata-bin 0. As already noted, however, placeholders may be used to replace any box, in any metadata-bin. This allows complex files to be decomposed in hierarchical fashion. As such, a single original file may be encapsulated in a variety of different metadata-bin structures, depending on how placeholders are used. **However, a single JPP-stream or JPT-stream shall adopt only one such encapsulation.** In client-server applications, the server will generally determine a suitable metadata-bin structure for the file, assigning a unique identifier to the resulting stream, and using the same metadata-bin structure in all communication with all clients which reference this same unique identifier.

When a placeholder relocates a box into a new metadata-bin, the header of that box (LBox, TBox and XLBox fields) is stored, unmodified, in the Placeholder box. If a client or rendering agent needs to map particular boxes to their original file offsets, it may do so using the original box headers that appear in the Placeholder boxes. This information ultimately allows any location in the original file to be mapped to a particular location in a particular metadata-bin, if the contents of that data-bin exist. This is important since some JPEG 2000 family files contain boxes that reference other boxes through their location within the file.

While considerable freedom exists in deciding how best to divide a file into metadata-bins, there is one restriction. Any Placeholder box that appears within a metadata-bin shall replace a top-level box within that data-bin. Equivalently, wherever a sub-box is to be replaced with a placeholder, its immediate containing super-box shall reside within its own metadata-bin. For example, in the sample file shown in Figure A.5, the XML data contained within the JP2 Header box may be placed in a separate data bin from the other boxes. This allows a server to deliver only those data-bins that are actually required for decoding and display of the image, unless XML data is explicitly requested. A suitable data-bin structure is shown in Figure A.7.

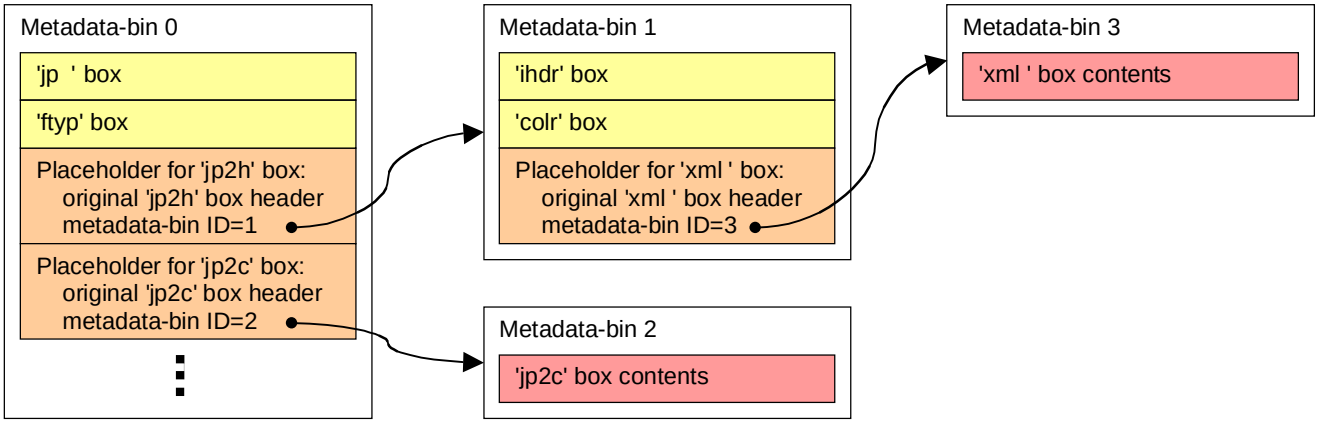

Figure A.A7 — A superbox with a referenced metadata-bin

It would not be legal, however, for the JP2 Header box to be left in metadata-bin 0, as shown in Figure A.8:

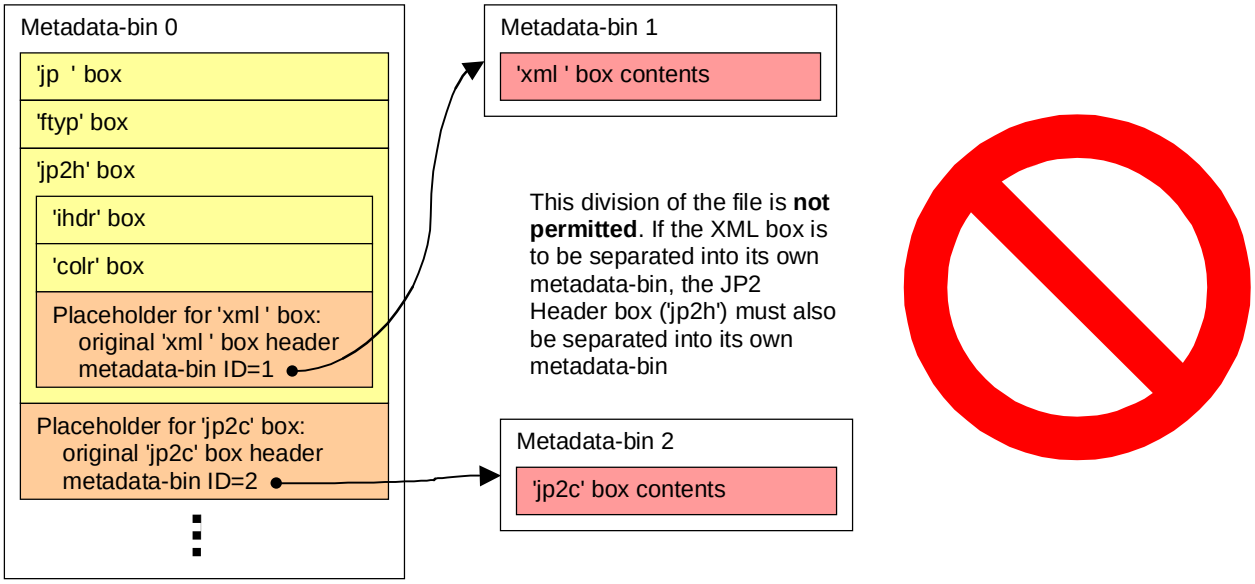

Figure A.A8 — An illegal division of the file into metadata-bins

NOTE An equivalent way to express this same restriction is as follows. Wherever a placeholder replaces a sub-box, a placeholder shall also replace its containing box. This restriction ensures that it is always possible for a client or rendering agent to recover the lengths and locations of the original boxes within the file, even if some of the boxes are not understood by the client.

In addition to providing the original contents of a box in a separate metadata-bin, JPP- and JPT-streams are also permitted to provide alternate representations of the box, which did not explicitly appear within the original file. These alternate representations are known as “stream equivalents.” For example, the original file might contain a Cross-reference box whose fragment list box collects one or more fragments of the file to reconstitute a Colourspace Specification box. While a client or rendering agent should be able to follow the relevant file pointers to reconstruct the Colourspace Specification box, a more convenient JPP- or JPT-stream representation might contain a placeholder which references a data-bin containing the Colourspace Specification box as a stream equivalent. To do this, the placeholder includes a box header for the stream equivalent, together with the identifier of the metadata-bin that holds the contents of the stream equivalent box.

The following example (shown in Figure A.9) illustrates the use of stream equivalents for Cross-reference boxes. In this case, the data-bin that holds the stream-equivalent contents is also referenced as holding the original contents of another box. While this is likely to be a common situation where the original file contained

cross-reference boxes, there is no need for the stream-equivalent to point to a metadata-bin that is connected to the original file hierarchy. The stream equivalent box's contents may be created from scratch or they may refer to content which originally existed within other files. This allows Cross-reference boxes whose fragment list references other files or URLs to be fully encapsulated within a single JPP- or JPT-stream.

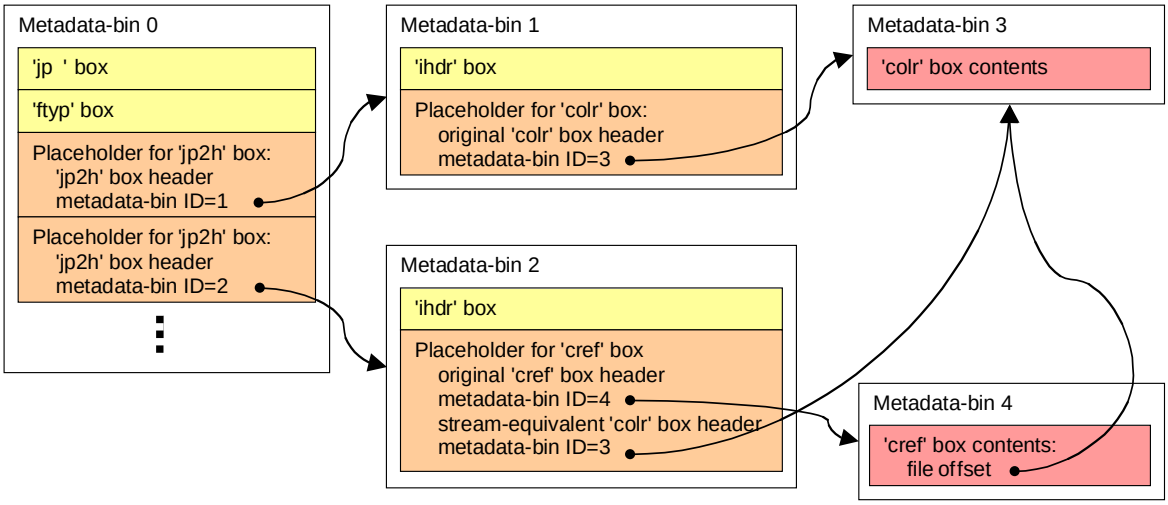

Figure A.A9 — Example of the use of stream equivalents

Stream equivalents may be used in any situation where the server can create an alternate form of the contents of a box that provide some benefit to the client; they are not just for providing access to explicitly cross-referenced data.

In addition to pointing to actual or equivalent box data, a placeholder box can point to one or more codestreams where the replaced box is equivalent to those codestreams. For example, the Contiguous Codestream box may be replaced by a placeholder box that references the ID of the incremental codestream contained within that Contiguous Codestream box. Another example would be to replace the Chunk Offset box in a Motion JPEG 2000 file with a placeholder that specifies an array of codestream ID's. Those codestream ID's refer to the codestreams that are pointed to by the Chunk Offset box.

A.3.6.3 Placeholder box format

Figure A.10 shows the format of a Placeholder box, including the box header (unlike the definition of most boxes in Annex I and other parts of this standard); it is specified this way to emphasise that the use of the length field in the box header for a Placeholder box is more restrictive than for other boxes.

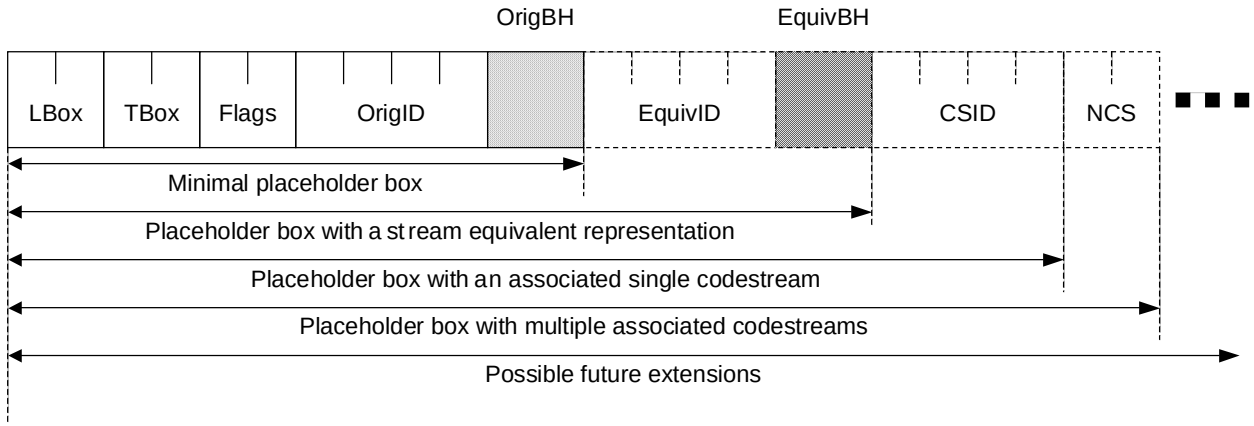

Figure A.A10 — Placeholder box structure

LBox: This is the standard 4-byte big-endian length field for a box. The value shall not be 1 for a Placeholder box, meaning that the XLBox field shall not be present.

TBox: This is the standard 4-byte box type field for a box. The type value for a Placeholder box shall be 'phld' (0x7068 6c64).

Flags: This field specifies what elements of the Placeholder box contain valid data. This field is encoded as a 4-byte big-endian integer. Legal values for the Flags field are specified in Table A.3.

OrigID: This field specifies the metadata-bin ID of the bin containing the contents of the original box represented by this Placeholder box. It is encoded as an 8-byte big-endian unsigned integer.

OrigBH: This field specifies the original header (LBox, TBox and XLBox, as needed) of the original box referenced by this Placeholder box. The length of this field is 8 bytes if the original box header's LBox field is not equal to 1 and 16 bytes otherwise.

EquivID: This field specifies the metadata-bin ID of the bin that contains a stream-equivalent form of the contents of this box. This field is encoded as an 8-byte big-endian unsigned integer.

EquivBH: This field specifies the header of the stream-equivalent box (LBox, TBox and XLBox as needed) of the box referenced by this Placeholder box. The length of this field is 8 bytes if the equivalent box header's LBox field is not equal to 1 and 16 bytes otherwise.

CSID: This field specifies the ID of the first codestream associated with the replaced box. This is the ID that is associated with all header, precinct and/or tile data-bins used to incrementally communicate the contents of the first codestream associated with the replaced box. This field is encoded as an 8-byte big-endian unsigned integer.

NCS: This field specifies the number of codestreams in the array of codestreams that is equivalent to the replaced box. The codestream ID values of these codestreams run contiguously from the value specified by the CSID field. This field is encoded as a 4-byte big-endian unsigned integer.

ExtendedBoxList: This field is not specifically shown in Figure A.10. The NCS field may be followed by a sequence of boxes containing extended information from the server. The existence of any box following the NCS field shall be specified through a bit in the Flags field. However, no extended boxes, nor any additional bit flags, are defined by this International Standard. Clients shall ignore any box in ExtendedBoxList that is not understood.

Table A.A3 — Legal values for the Flags field of a Placeholder box

Value	Meaning
yyyy yyyy yyyy yyyy yyyy yyyy yyyy xxx1	Access is provided to the original contents of this box through the metadata-bin specified in the OrigID field
yyyy yyyy yyyy yyyy yyyy yyyy yyyy xxx0	No access is provided to the original contents of this box, and the value of the OrigID field shall be ignored
yyyy yyyy yyyy yyyy yyyy yyyy yyyy xx1x	A stream-equivalent box is provided, whose contents are in the metadata-bin specified by the EquivID field.
yyyy yyyy yyyy yyyy yyyy yyyy yyyy xx0x	No stream-equivalent box is provided, and the value of any EquivID and EquivBH fields shall be ignored
yyyy yyyy yyyy yyyy yyyy yyyy yyyy 01xx	Access to the image represented by this box is provided by a single incremental codestream, which is identified by the CSID field. The value of the NCS field shall be treated as if was set to "1" regardless of the actual value of that field.
yyyy yyyy yyyy yyyy yyyy yyyy yyyy 11xx	Access to the image represented by this box is provided by one or more incremental codestreams, as specified by the CSID and NCS fields.

Table A.A3 — Legal values for the Flags field of a Placeholder box

Value	Meaning
yyyy yyyy yyyy yyyy yyyy yyyy yyyy x0xx	This placeholder does not provide access to an image representing the original box as an incremental codestream; the CSID and NCS fields shall be ignored.
Other values	Reserved for ISO use

A bit value of "x" in Table A.3 indicates that the specified value includes cases where that bit is set to either "1" or "0". Bits indicated as "y" are unused by this standard and shall be set to 0 by servers and ignored by clients.

Not all of the fields defined in for a Placeholder box need appear in every Placeholder box. As suggested by the arrows in Figure A.10, if no box equivalent or incremental codestream ID is provided, the box may be terminated at the end of the OrigBH field. Similarly, if no incremental codestream ID is provided, the box may be terminated at the end of the EquivBH field, and if no more than one incremental codestream ID is provided, the box may be terminated at the end of the CSID field.

A.3.6.4 Referencing of incremental codestreams with placeholders

Wherever header, precinct or tile data-bins exist, their codestream ID shall appear in a Placeholder box within an appropriate metadata-bin. The only exception to this requirement is for unwrapped JPEG 2000 codestreams, which are not embedded within a JP2-family file format.

The codestream ID values that appear within the relevant Placeholder box shall conform to any requirements imposed by the containing file format. For example, JPX files formally assign a sequence number to each codestream that appears at the top level of the file, either through a Contiguous Codestream box or a Fragment Table box. The first top-level codestream in the logical target shall have a codestream ID of 0; the next shall have a codestream ID of 1; and so forth.

Placeholders that reference multiple codestream ID's may be used only where the meaning of those codestreams is well defined by the type of the box that is being replaced. This International Standard defines only two such box types, for use with Motion JPEG 2000 (MJ2) files. Specifically, either the chunk offset box ('stco') or the chunk large offset box ('co64') may be replaced by a Placeholder box which identifies multiple codestream ID's.

A.3.6.5 Using Placeholder boxes with MJ2

Each video track in an MJ2 file contains exactly one chunk offset box (either 'stco' or 'co64') that, in combination with the sample to chunk box ('stsc'), serves to identify the locations of all of the contiguous codestream boxes that belong to the video track. If the chunk offset box is replaced by a placeholder that provides one or more codestream ID's, there shall be exactly one codestream ID for each contiguous codestream box in the video track. If the visual sample entry box ('mjp2') identifies a field count of 2, there shall be 2*N* codestream ID's in the range provided by the Placeholder box, where *N* is the number of video samples (i.e., *N* is the number of frames). Otherwise, there shall be only *N* codestream ID's in the range provided by the Placeholder box. The codestream ID's shall be sequenced by sample number (frame number) and by field number within each sample.

NOTE For MJ2 files in a JPP-stream or a JPT-stream representation, there is no need for the stream to contain the contents of the original chunk offset box, the sample to chunk box ('stsc'), or the sample size box ('stsz'). This indexing information can be regenerated if needed if the stream representation is converted to an MJ2 file.

A.4 Conventions for parsing and delivery of JPP-Stream and JPT-Streams (informative)

Placeholder boxes create additional flexibility and some potential ambiguity for both clients and servers in how they parse or deliver JPP- and JPT-streams. A server may choose to partition original boxes from a JP2-family file into metadata-bins using any of a wide range of strategies, by introducing Placeholder boxes at appropriate points. The server shall do this in a consistent way so that the data-bins associated with a JPP- or JPT-stream have the same nominal contents for all clients which access the same logical target (possibly qualified by a unique target ID), whenever they access it.

More significantly, however, Placeholder boxes allow servers to construct a single JPP- or JPT-stream whose data-bins provide multiple alternate representations of the same original content. This can happen when a streaming equivalent is identified within a placeholder, and/or when an incremental codestream ID is identified within a placeholder. In these cases, an original JP2 box might be made available in a metadata-bin, while also being made available as a stream equivalent in yet another metadata-bin, and/or also being made available as an incremental codestream via header, precinct or tile data-bins. While servers might distribute the contents of all data-bins that represent an original box, for efficiency reasons servers would be expected to distribute only sufficient information to convey the original content, unless explicitly asked to distribute redundant data-bins. Client-side parsers of JPP- or JPT-streams, when confronted with multiple representations of an original box, might choose to ignore all but one of the representations. The expected client convention should have a significant impact on which metadata-bins the server chooses to actually send to a client.

In view of this, this International Standard recommends the following conventions:

— Unless a server has reason to believe otherwise, it shall assume that the client parser will parse a stream equivalent box in preference to the original box if the presence of both box types has been signalled to the client by placeholders.

— Unless a server has reason to believe otherwise, it shall assume that the client parser will use the incremental codestream representation (header, precinct or tile data-bins) in preference to a raw codestream if the presence of both box types has been signalled to the client by placeholders.

A.5 Conventions for JPP-Stream or JPT-Stream Interoperability (informative)

This convention describes the exchange file format for JPP-Stream and JPT-Stream, herein termed jpp-file and jpt-file respectively. Such a file may contain the received JPEG2000 data from a JPIP session (the client's cache for example), or a subset thereof. It is possible for another JPIP client to read and use this file because JPP-Stream and JPT-Stream are self-describing media types.

These files are formed by concatenation of JPT-Stream or JPP-Stream messages. For example, they may be formed by the simple concatenation of all such messages received by a client in a single session or from multiple sessions. An improved situation would be where clients generated a legal JPT-Stream or JPP-stream using a single Message Header and Message per data-bin.

It is recommended that the ".jpp" and ".jpt" extensions be used for these files and, if appropriate, that the file name includes a reference to a relevant JPIP `target` token or `target-id` token.

This convention does not specify the implementation or structure of the JPIP Cache for a client. For example, a client may use a database to serve as its implementation of the cache function rather than a file-based cache system.

Annex B
(normative)

Sessions, Channels, Cache Model and Model-sets

B.1 Requests Within a Sessions vs. Stateless Requests

The JPIP protocol makes a clear distinction between two different types of requests: stateless requests and requests which belong to a session.

The purpose of sessions is to reduce the amount of explicit communication required between the client and server. Within a session, the server is expected to remember client capabilities and preferences supplied in previous requests so that this information need not be sent in each and every request. Even more importantly, the server would typically keep track of the information it has already sent to the client in response to previous requests, so that this information need not be re-transmitted in response to future requests. Unless explicitly instructed otherwise, the server may assume that the client caches the responses to all requests issued within a session, and may model the client's cache, sending only those portions of the compressed image data or metadata which the client does not already have in its cache.

Stateless requests are not associated with any session and so shall be entirely self-contained. It should be noted that the term “stateless” applies only to the server, not the client. As for sessions, the client should generally cache the responses from previous requests associated with the same logical target. Clients that issue multiple stateless requests for the same target should generally include information about their cache contents with each request, so as to avoid the transmission of redundant data. Thus, the benefits of sessions are smaller, less complex requests and/or less redundant response data from the server. The benefit of stateless communication is that the server need not maintain state information between requests; usually, this means that the same host need not ultimately serve all requests for a single target image that emanate from a single client.

B.2 Channels and Sessions

Associated with each session are the following elements:

— One or more logical targets (usually image files), whose content does not change over the session.

— A single image data return type for each logical target associated with the session.

— For each logical target associated with the session, a model of the client’s cache contents shall be maintained wherever the data return type is one of “jpp-stream” or “jpt-stream”. Note, however, that this model need not perfectly reflect the actual state of the client’s cache. Rules governing the maintenance of cache models are outlined in B.3.

— One or more JPIP channels. Clients may generally open multiple channels within the same session. Each JPIP channel may be associated with a separate underlying transport channel (e.g., a separate TCP connection), although this might not be the case. Multiple channels allow clients to issue simultaneous requests for multiple image regions, with the expectation that the server will respond to these requests concurrently. Channels also allow for intelligent bandwidth allocation amongst different types of requests either within a single target image or across multiple targets.

— Where multiple channels are associated with the same logical target, the session cache model applies across all channels. Multiple clients may open JPIP channels within the same session, although this might have undesirable side effects if the channels refer to the same logical target.

Associated with each channel are the following elements:

— A single logical target (usually an image file).

— A server-assigned identifier that shall be included with each request. JPIP does not define a separate session identifier, since the channel identifier is sufficient to associate the request with its session.

— A record of the client's capabilities and preferences, which may be adjusted through appropriate request fields.

— To the extent that the server queues requests, it should provide a separate queue for each JPIP channel.

B.3 Cache Model Management

As already noted, one of the principle roles played by sessions is that of modelling the client cache. Unless explicitly informed otherwise, the server **may** assume that the client has cached all information that it has sent in response to requests within the session. This information **need not** be re-transmitted. Note, however, that the server is not obliged to maintain a complete cache model, or indeed any cache model at all. In this case, some redundant data may be transmitted in response to future requests, but this is not illegal.

In addition to the impact of transmitted data, explicit cache model manipulation statements in client requests may update the server's cache model. These statements are to be processed before determining the data that should be returned to the client in response to its request. There are two types of cache model manipulation statements: additive and subtractive.

Additive cache model manipulation statements serve to augment the server's cache model, adding data-bins, or portions of data-bins to the existing model. These provide a mechanism for a client to inform the server about information which it received in a previous session, or using previous stateless requests. A server **should** attempt to exploit any additive cache model manipulation statements that appear in client requests. However, servers are not obliged to maintain a complete cache model, so a server may disregard, or partially disregard, additive cache model manipulation statements.

Subtractive statements serve to remove data-bins, or portions of data-bins from the server's cache model. A client might issue subtractive cache model manipulation statements in order to inform the server that it has discarded some data which was previously sent by the server, rather than caching that data. The server is otherwise free to assume that the client has cached all data transmitted during the session. The server **shall** remove all information identified by a subtractive cache model statement from any cache model (complete or otherwise) that it is maintaining.

Session-based JPIP requests have side effects, which may affect the response to future requests. This is true also of requests that contain cache model manipulation statements – the effects of cache model manipulation are persistent. Moreover, the side effects of a request arriving on one JPIP channel are reflected in the response to any requests that might belong to a different JPIP channel which is associated with the same logical target. This follows from the fact that there is only cache model for each logical target in each session.

B.4 Interrogation and Manipulation of Model-Sets

Where a logical target associated with a session contains a large number of codestreams (e.g. a video target), or a client remains connected for a long period of time, partial cache modelling becomes an increasingly likely strategy for practical server implementations. It also becomes increasingly likely that clients will be unable to cache all information sent by the server. To avoid communication inefficiencies in such circumstances, the concept of a "mset" is introduced. The "mset" is the collection of codestreams for which client cache contents are being modelled by the server.

In any request, the client may instruct the server to limit its “mset” to a particular set of codestreams. This provides a convenient mechanism for clients to discard whole codestreams from their cache without running the risk that the server will generate incomplete responses to future requests for those codestreams.

Model-set requests also result in server responses which indicate the actual set of codestreams for which cache model information is being maintained. This allows clients to determine whether or not cache model manipulation statements which refer to a variety of codestreams will be disregarded by the server.

In the absence of any explicit “mset” manipulation or interrogation, the client may assume only that the server’s “mset” includes all codestreams for which response data is generated to its request. Since servers generally have the right to limit the scope of a client’s request to a smaller number of codestreams than the number which was originally specified, there is no guarantee that the server’s “mset” will include all of the codestreams mentioned in a request, unless the request mentioned only one codestream. These matters are explained further in C.9.4.

Annex C
(normative)

Client request

C.1 Request syntax

Ed Note: Make the following paragraph non-hanging text

This annex describes all possible elements in a JPIP request. Each major section describes a group of fields and possible values for those fields. In general, a request will consist of fields from more than one group, but some groups are incompatible. Further, within each group, some request fields are incompatible. Some otherwise legal requests may not be valid for use in some situations (e.g. sessions), even though this is not indicated by the BNF syntax. Finally, even with a legal request, a server may not support all possible request fields or combinations there-of.

C.1.1 Request structure

The JPIP request consists of the following fields:

— Target identification fields

— Session and channel management fields

— View-window request fields

— Metadata field

— Data limiting request fields

— Server control request fields

— Cache management request fields

— Upload request fields

— Client capability and preference fields

The elements in the request shall be sent in compliance with the seleced transport protocol. For example, in http, the requests are expressed as the characters listed in the BNF syntax, multiple parameters are joined with an “&” character, and the requests may be part of the query field of a GET request, or the body of a POST request. See Annex F, Annex G and Annex H for details.

```
jpip-request-field = target-field
                   | channel-field
                   | view-window-field
                   | metadata-field
                   | data-limit-field
                   | server-control-field
                   | cache-management-field
                   | upload-field
                   | client-cap-pref-field
```

```
target-field       = target                    ; Annex C.2.2
                   | subtarget                 ; Annex C.2.3
                   | tid                       ; Annex C.2.4

channel-field      = cid                       ; Annex C.1.1
                   | cnew                      ; Annex C.3.2
                   | cclose                    ; Annex C.3.3
                   | qid                       ; Annex C.3.4


view-window-field  = fsiz                      ; Annex C.4.2
                   | roff                      ; Annex C.4.3
                   | rsiz                      ; Annex C.4.4
                   | comps                     ; Annex C.4.5
                   | stream                    ; Annex C.4.6
                   | context                   ; Annex C.4.7
                   | srate                     ; Annex C.4.8
                   | roi                       ; Annex C.4.9
                   | layers                    ; Annex C.4.10

metadata-field     = metareq                   ; Annex C.5.2

data-limit-field   = len                       ; Annex C.6.1
                   | quality                   ; Annex C.6.2

server-control-field = align                   ; Annex C.7.1
                   | wait                      ; Annex C.7.2
                   | type                      ; Annex C.7.3
                   | drate                     ; Annex C.7.4

cache-management-field = model                 ; Annex C.8.1
                   | tpmodel                   ; Annex C.8.3
                   | need                      ; Annex C.8.4
                   | tpneed                    ; Annex C.8.5

                   | mset                      ; Annex C.8.6

upload-field       = upload                    ; Annex C.9.1

client-cap-pref-field = cap                    ; Annex C.10.1
                   | pref                      ; Annex C.10.2
                   | csf                       ; Annex C.10.3
```

C.1.2 Restrictions on combining request fields

Each JPIP request field shall occur no more than once in a single request.

In general, requests for image data (view-window requests) may be combined with requests for additional metadata. However, there are restrictions, as follows, on how the request fields may be combined:

— The upload request field shall not be combined with `metadata-field`, `data-limit-field`, or `server-control-field`.

C.2 Target identification fields

C.2.1 Introduction to logical targets

Each JPIP request has a logical target. The target is usually a file on the server, but it might refer to a collection of files or an image or images that are dynamically constructed by the server as required. The logical target may also be a byte range of a file.

In some cases, it might not be necessary to explicitly specify the logical target. An important example of this is when the request includes a Channel ID request field, since channels are always associated with exactly one logical target, as explained in Annex B.

To explicitly specify a logical target, the main target name may be provided using the Target request field. This may be further qualified by the provision of a Sub-target request field, which provides a byte range for the file (or other entity) associated with the main target name. The main target name might also be specified by other means, such as the path component of the resource name supplied in an HTTP request.

Where a logical target is to be served with JPP-stream or JPT-stream messages, the associated data-bins shall remain consistent throughout all responses which are issued within the same session. Where the server, or a related server, also issues a Target ID, the data-bins shall remain consistent across all responses issued with the same Target ID, whether they are issued within the same session or not.

C.2.2 Target (target)

```
target = "target" "=" TOKEN
```

This field is used to specify the main target name (often the name of a file on the server). If the Target request field is missing then the main target name is determined by other means.

NOTE For example, the main target name may be obtained from the path component of the URL used for a JPIP request carried by HTTP. In the URL “http://one.jpeg.org/images/picture.jp2?fsiz=200,200”, the main target name is “/images/picture.jp2”. Another server might identify the same main target name with a URL of the form “http://two.jpeg.org/cgi-bin/imagerserver.cgi?target=/images/picture.jp2&fsiz=200,200”.

C.2.3 Sub-target (subtarget)

```
subtarget = "subtarget" "=" byte-range

byte-range = UINT-RANGE
```

This field may be used to qualify the main target name. The logical target is to be interpreted as the indicated byte range of the main target identified either by the Target request field or by other means. Where the server assigns a unique target ID, that target ID refers to the complete logical target, not just the main target name. Similarly, where the server creates channels, each channel is associated with a particular logical target, not all logical targets with the same main target name.

The lower and upper bounds of the supplied byte-range are inclusive, where 0 refers to the first byte of the target file.

C.2.4 Target ID (tid)

```
tid = "tid" "=" target-id

target-id = TOKEN
```

This field may be used to supply a `target-id` string, which was previously generated by the server to absolutely identify the logical target that is being accessed. The logical target name is not necessarily unique

and does not necessarily correspond to a single encoding of its contents, whereas the `target-id` string should absolutely identify both the imagery and its encoding.

If `target-id` is "0", the client is requesting the server to explicitly inform it of the assigned target-id, if there is one. In this case, the server shall include a Target ID header in its response.

`target-id` shall not exceed 255 characters in length.

C.3 Fields for working with sessions and channels

C.3.1 Channel ID (cid)

```
cid = "cid" "=" channel-id

channel-id = TOKEN
```

This request field is used to associate the request with a particular JPIP channel, and hence the session to which the channel belongs. A request is deemed to be stateless unless one or both of the following conditions occur:

— The request includes a valid Channel ID field

— The request includes a New Channel field (see below), and the server response includes a New Channel response header with a newly issued `channel-id`.

C.3.2 New Channel (cnew)

```
cnew = "cnew" "=" 1#(transport-name)

transport-name = TOKEN
```

This request field is used to request a new JPIP channel. If no Channel ID request field is present, the request is for a new session. Otherwise, the request is for a new channel in the same session as the channel identified by the Channel ID request field.

The value string identifies the names of one or more transport protocols that the client is willing to accept. This standard defines only the transport names "http" and "http-tcp," although it is anticipated that other transports might be defined elsewhere. Details of the use of JPIP over the "http" transport appear in Annex F, while details of the use of JPIP over the "http-tcp" transport appear in Annex G.

If the server is willing to open a new channel, using one of the indicated transport protocols, it shall return the new channel identifier token using the New Channel response header (see Annex D). In this case, the present request is the first request within the new channel.

It is possible for a client to open a channel to a new logical target within the same session. To do this, the client's request shall identify both an existing Channel ID, and a logical target. When opening a new channel to the same logical target which is associated with an existing channel, there is no need to specify the logical target explicitly.

If the server is not willing to open a new channel, it shall not return a New Channel response header, but the request shall be serviced as though the New Channel request field had not been included. This means that a request that specifies an existing Channel ID shall be treated as a request within that channel, while a request that includes no Channel ID request field shall be treated as a stateless request. In the event that the New Channel request identifies a different logical target to that which is associated with the supplied existing Channel ID, the server will not be able to respond to the request without either issuing a new Channel ID or returning an error code.

EXAMPLE 1 "target=nice.jp2&cnew=http" requests the first channel of a new session to the image "nice.jp2", using the "http" transport. If no channel is assigned by the server, the request will be treated as stateless.

EXAMPLE 2 "cid=013ac8&cnew=http-tcp" requests a new channel within the same session which is associated with Channel ID 013ac8. The new channel is to use the "http-tcp" transport and refers to the same logical target as Channel ID 013ac8. A single cache model is shared by these channels. If no channel is assigned by the server, the request will be treated as though the New Channel request field had been omitted.

EXAMPLE 3 "target=nice.jp2&cid=013ac8&cnew=http" requests a new channel within the same session which is associated with Channel ID 013ac8. The new channel is to use the "http" transport. The logical target associated with the new channel is distinct from that associated with Channel ID 013ac8 and a separate cache model is created for the new channel.

C.3.3 Channel Close (cclose)

```
cclose = "cclose" "=" "*" | 1#(channel-id)
```

This request field is used to close one or more open channels to a session. If the value field contains one or more channel-id tokens, they shall all belong to the same session. In this case, the Channel ID request field is not necessary, but if provided it shall also reference a channel belonging to the same session.

If the value field is "*", all channels associated with the session will be closed. In this case, the session shall be identified by the inclusion of a Channel ID request field.

The server shall complete its response to any request that includes a Channel Close request field before it closes the channel.

C.3.4 Request ID (qid)

```
qid = "qid" "=" UINT
```

Each channel has its own request queue, with its own request ID counter. Requests which are received within any given channel (as indicated by the Channel ID value) shall be processed in the order of their Request ID values, where the Request ID field is used. The server may process requests which do not contain a Request ID field on a first-come-first-served basis. However, it may not process a request which arrives with a Request ID value of *n* until it has processed a request with a Request ID value of *n*-1 which is associated with the same channel, unless *n*=0. The client shall not issue a request which specifies the same Channel ID value as any other request associated with the same channel.

C.4 View-window request fields

C.4.1 Mapping view-window requests to codestream image resolutions and regions

The purpose of JPIP is to provide portions of a JPEG 2000 image and associated metadata in response to requests from a client. This is done via a sequence of requests and responses. For the image portion, the data requested may be less than the full image in terms of image size, region, quality, and/or components.

In the simplest case, the image portion in question is defined directly with respect to the high resolution reference grid of the JPEG2000 codestream(s) identified in the request, not the sampled grid of any particular image component. More generally, however, clients may request higher level image objects (e.g., JPX compositing layers or MJ2 video tracks) via the Codestream Context request field. In this case, the requested image portion may need to be subjected to a coordinate transformation, in order to determine the portion of each associated codestream which is being requested. These coordinate transformations are described in C.4.7, and they may be understood in terms of the following description of codestream image regions.

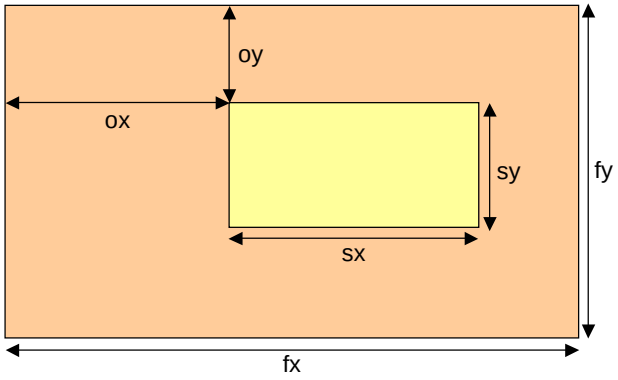

Figure C.C1 — Desired region within an image

Codestream image regions are described using 3 two-dimensional parameters, as shown in Figure C.1. The size parameters (`sx` and `sy`) and the offset parameters (`ox` and `oy`) specify the width and height of the desired codestream image region and the top-left corner of that region, with respect to a whole image that has the given frame size (`fx` and `fy`). For example, a client wishing to fill a 640 × 480 display with the whole image could make a request as follows "`fsiz=640,480&rsiz=640,480&roff=0,0`". Note that this can be done regardless of the original size of the image (and indeed without knowing the original size of the image).

When none of the available image resolution in the JPEG 2000 codestream correspond exactly to the requested frame size, the returned image data may be larger or smaller than the requested frame size, and may even differ in aspect ratio. The server shall determine a suitable codestream image resolution, denoted by size parameters `fx'` and `fy'`, and a suitable region on the codestream, denoted by parameters `rx'`, `ry'`, `ox'` and `oy'`. Although the client may specify the direction for rounding, as part of the Frame Size request field, the client shall be prepared to deal with returned data that does not match the requested parameters exactly.

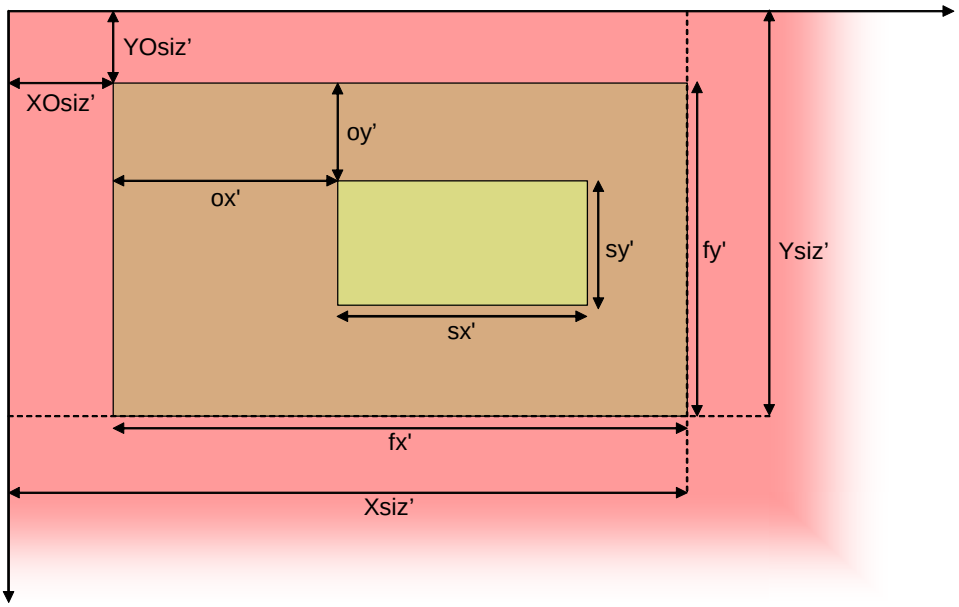

Figure C.C2 — Desired region with respect to the reference grid

As shown in Figure C.2, the size of the requested codestream image resolution is given by fx'=Xsiz'-XOsiz' and fy'=Ysiz'-YOsiz', where XOsiz', YOsiz', Xsiz', and Ysiz' are derived using equation (1).

equation (1) $$XOsiz' = \left\lceil \frac{XOsiz}{2^r} \right\rceil; \quad YOsiz' = \left\lceil \frac{YOsiz}{2^r} \right\rceil; \quad Xsiz' = \left\lceil \frac{Xsiz}{2^r} \right\rceil; \quad Ysiz' = \left\lceil \frac{Ysiz}{2^r} \right\rceil$$

Here, XOsiz, YOsiz, Xsiz and Ysiz are taken from the relevant codestream's SIZ marker segment, and *r* is determined by the server in order to match the requested image size (fx and fy) as closely as possible, subject to any rounding preferences supplied via the Frame Size request field. It is natural to interpret *r* as a number of discarded highest DWT (wavelet transform) levels, and indeed *r* must be no less than 0. However, the value of *r* is not limited by the number of DWT levels which were used to compress any tile-component in the codestream.

Once fx' and fy' have been found, the size and offset associated with the codestream image region are determined by equation (2).

equation (2) $$ox' = \left\lfloor ox \cdot \frac{fx'}{fx} \right\rfloor; \quad oy' = \left\lfloor oy \cdot \frac{fy'}{fy} \right\rfloor; \quad sx' = \left\lceil sx \cdot \frac{fx'}{fx} \right\rceil; \quad sy' = \left\lceil sy \cdot \frac{fy'}{fy} \right\rceil$$

For example, suppose the requested Frame Size is 128 × 128, and the image on the codestream's high resolution reference grid is described by XOsiz=127, Xsiz=648, YOsiz=0 and Ysiz=504. Suppose also that 3 levels of wavelet transform exist for all image components in the codestream. The available codestream image sizes are then:

$$521 \times 504 \quad \left(\left\lceil \frac{648}{1} \right\rceil - \left\lceil \frac{127}{1} \right\rceil \text{ by } \left\lceil \frac{504}{1} \right\rceil - 0 \right)$$

$$260 \times 252 \quad \left(\left\lceil \frac{648}{2} \right\rceil - \left\lceil \frac{127}{2} \right\rceil \text{ by } \left\lceil \frac{504}{2} \right\rceil - 0 \right)$$

$$130 \times 126 \quad \left(\left\lceil \frac{648}{4} \right\rceil - \left\lceil \frac{127}{4} \right\rceil \text{ by } \left\lceil \frac{504}{4} \right\rceil - 0 \right)$$

$$65 \times 63 \quad \left(\left\lceil \frac{648}{8} \right\rceil - \left\lceil \frac{127}{8} \right\rceil \text{ by } \left\lceil \frac{504}{8} \right\rceil - 0 \right)$$

Thus if the request is for a larger frame size (round-direction is round-up) the returned frame size will be 260 × 252. If the request is for a smaller frame size (round-direction is round-down), then a 65 × 63 frame size will be used. Note that, as in this example, the available codestream image frame sizes are not generally exact powers of 2.

Subsampling of an image component, as specified by XRsiz and YRsiz, has no effect on the interpretation of the requested image region or image resolution within any requested codestream. For example, a request for a 256 × 256 region from the upper left corner of a 512 × 512 image can be made with:

```
fsiz=512,512&rsiz=256,256
```

Suppose the codestream contains an image subsampled in components 1 and 2 but not in component 0. Specifically, suppose Xsiz=1024, Ysiz=1024, XOsiz=0, YOsiz=0, and $XRsiz_0$=1, $YRsiz_0$=1, $XRsiz_1$=2, $YRsiz_1$=2, $XRsiz_2$=2, and $YRsiz_2$=2. The server would leave out the highest resolution level of all three components, and return tiles or precincts sufficient to provide 256 × 256 samples of component 0, but only 128 × 128 samples of components 1 and 2. The client thus has data to display the upper left corner at half the size of the full image and still subsampled. If the client desires to display non-subsampled chroma components, it could issue an additional request such as

```
fsiz=1024,1024&rsiz=512,512&comps=1,2
```

The server would then return sufficient data to provide 256 × 256 samples of components 1 and 2, which could be combined with the component 0 data already received to obtain a non-subsampled but half sized image.

If all three components had been subsampled, the server would provide only 128 × 128 samples of all three components for the original request (`fsiz=512,512&rsiz=256,256`) since image resolution and image regions are assessed with respect to the reference grid of each requested codestream.

C.4.2 Frame Size (fsiz)

```
fsiz = "fsiz" "=" fx "," fy ["," round-direction]

fx = UINT

fy = UINT

round-direction = "round-up" | "round-down" | "closest"
```

This field is used to identify the resolution associated with the requested view-window. The values `fx` and `fy` specify the dimensions of the desired image resolution. The round-direction value specifies how an available codestream image resolution shall be selected for each requested codestream, if the requested image resolution is not availablewithin that codestream. The requested frame size is mapped to a codestream image resolution, following the procedure described in Section C.4.1, possibly with the addition of coordinate transformations requested via a Codestream Context request field. A client wishing to control the exact number of samples received for a particular image component may need to increase the requested frame size, as explained in Section C.4.1. The `round-direction` options defined by this International Standard are described in Table C.1.

Table C.C1 — Round direction options

Round-direction	Meaning
"round-up"	For each requested codestream, the smallest codestream image resolution whose width and height are both greater than or equal to the specified size shall be selected. If there is none, then the largest available codestream image resolution shall be used (*r*=0).
"round-down"	For each requested codestream, the largest codestream image resolution whose width and height are both less than or equal to the specified size shall be selected. This is the default value when the round-direction parameter is not specified.
"closest"	For each requested codestream, the codestream image resolution that is closest to the specified size in area (where area = fx x fy) shall be selected. Where two codestream image resolutions have areas which are equidistant from fx x fy, the larger of the two shall be selected.

If the Frame Size request field is omitted from a view-window request and `metadata-only` is not specified in a metadata request field (C.5.1), the requested view-window includes no compressed image data and no tile-specific headers, but it does include all other header (codestream and file format) information that would have been returned had the client included the Frame Size request field. See C.5.1 for further information on the file format information (metadata) which is implicitly requested along with the view-window request.

C.4.3 Offset (roff)

```
roff = "roff" "=" ox "," oy
```

```
ox = UINT

oy = UINT
```

This field is used to identify the upper left hand corner (offset) of the spatial region associated with the requested view-window; if not present, the offsets default to 0. The actual displacement of a codestream image region from the upper left hand corner of the image, at the actual codestream image resolution selected by the server, is obtained following the procedure described in C.4.1, possibly with the addition of coordinate transformations requested via a Codestream Context request field.

Use of the Offset field is legal only in conjunction with the Frame Size request field.

If a codestream image region specified using Region Size and/or Offset turns out to be empty (no area), the server's response should not include any compressed image data for that codestream. In particular, responses of type JPP-stream or JPT-stream should contain no messages which reference precinct, tile or tile-header data-bins of that codestream. The server may, at its discretion, opt to return main header or metadata-bin messages that would have been returned in response to a request that omitted the Frame Size request field.

C.4.4 Region Size (rsiz)

```
rsiz = "rsiz" "=" sx "," sy

sx = UINT

sy = UINT
```

This field is used to identify the horizontal and vertical extent (size) of the spatial region associated with the requested view-window in reference grid units; if not present, the region extends to the lower right hand corner of the image. The actual dimensions of a codestream image region, at the actual codestream image resolution selected by the server, are computed following the procedure described in C.4.1, possibly with the addition of coordinate transformations requested via a Codestream Context request field. A requested codestream image region need not necessarily be fully contained within the codestream, in which case the server simply takes the intersection between the available codestream image region and the requested region.

Use of the Region Size request field is legal only in conjunction with the Frame Size request field.

If a codestream image region turns out to be empty (no area), the server's response should not include any compressed image data for that codestream. In particular, responses of type JPP-stream or JPT-stream should contain no messages which reference precinct, tile or tile-header data-bins of that codestream. The server may, at its discretion, opt to return main header or metadata-bin messages that would have been returned in response to a request that omitted the Frame Size request field.

C.4.5 Components (comps)

```
comps = "comps" "=" 1#(UINT-RANGE)
```

This field is used to identify the image components that are to be included in the requested view window; if not present, the request is understood to include all available image components of all codestreams identified via the Codestream request field, and all relevant components of all codestreams requested via the Codestream Context request field. These "relevant" components are those which are involved in the reproduction of the image entities (e.g., JPX compositing layers or MJ2 video tracks) which are specified via the Codestream request field.

The values in this request field represent the indices of the image components of interest. Image component indices start from 0, and have the interpretation assigned to them by the JPEG 2000 codestream syntax, as described in ITU–T Rec. T.800 | ISO/IEC 15444–1, but note that these are the components which are obtained by decoding and inverse wavelet transforming the compressed data, prior to the application of the inverse RCT or ICT component transform. For codestreams conforming to ISO/IEC 15444–2, the components

identified here are those identified as “spatial components”, i.e., those obtained by decoding and inverse wavelet transforming the compressed data, prior to the application of any inverse multi-component transform, dependency component transform, or multi-component wavelet transform.

Non-existent components in any of the requested codestreams shall be disregarded.

C.4.6 Codestream (stream)

```
stream = "stream" "=" 1#(sampled-range)

sampled-range = UINT-RANGE [":" sampling-factor]

sampling-factor = UINT
```

This field is used to identify which codestream or codestreams belong to the requested view-window. If the field is omitted and the codestream(s) cannot be determined by other means, the default is the single codestream with identifier 0. Note that the Codestream Context request field provides an additional means for requesting codestreams.

For JPEG 2000 family targets, codestream indices are those which are embedded in the corresponding Placeholder box that appears within the appropriate metadata-bin, as described in A.3.6. For file formats which have implied codestream identifiers, those identifiers should agree with the indices used here.

Where a range of codestreams is identified, the absence of an upper bound means that the range extends to all codestreams with larger identifiers. Where an upper bound is provided, the upper bound provides the absolute identifier of the last codestream in the range.

Whether or not an upper bound is provided, a codestream range may be qualified by an additional sampling-factor. The sampling-factor, if provided, shall be a strictly positive integer, *F*. The range then includes all codestream identifiers *L+Fk* which lie within the unqualified range, where *L* is the identifier of the first codestream in the range.

C.4.7 Codestream Context (context)

```
context = "context" "=" 1#(context-range)

context-range = jpxl-context-range | mj2t-context | reserved-context

jpxl-context-range = "jpxl" "<" jpx-layers ">" [ "[" jpxl-geometry "]" ]

jpx-layers = sampled-range

jpxl-geometry = "s" jpx-iset "i" jpx-inum

jpx-iset = UINT

jpx-inum = UINT

mj2t-context = "mj2t" "<" mj2-track ">" [ "[" mj2t-geometry "]" ]

mj2-track = NONZERO ["+" "now" ]

mj2t-geometry = "track" | "movie"

reserved-context = 1*( TOKEN | "<" | ">" | "[" | "]" | "-" | ":" | "+" )
```

This request field may be used to request codestreams indirectly via “higher level” image entities. This International Standard defines contexts corresponding to JPX compositing layers (a JPX compositing layer

may involve one or more codestreams) and MJ2 video tracks; however, the mechanism is designed with extensibility in mind.

If a Codestream Context request field is supplied, the requested view-window includes each of the codestreams which are associated with the requested context(s), in addition to any codestreams requested via the Codestream request field.

The body of a Codestream Context request field consists of one or more context-range values. Each context-range is associated with a set of codestreams which can be determined by the server. A context-range may also identify coordinate remapping transformations which shall be applied to the Frame Size, Region Size and Offset parameters in order to determine the codestream image resolution and codestream image region for each of the codestreams associated with that context-range. Where the server is prepared to process a context-range, it shall identify the codestreams which are associated with that context-range by means of a Codestream Context response header.

This International Standard defines two specific types of context-range, which are intended to address the needs of the JPX and MJ2 file formats. The first of these context-range types, jpxl-context-range, is used to identify one or more JPX compositing layers. The indices of the compositing layers associated with a jpxl-context-range are supplied in the form of a sampled-range, following the same semantics as sampled codestream ranges in the Codestream request field. Where a jpxl-context-range is processed by the server, the codestreams belonging to the corresponding compositing layer(s) shall be identified within a Codestream Context response header.

A jpxl-context-range may identify an optional coordinate remapping transformation, to be used in deducing the codestream image resolution and the codestream image region for each of its codestreams. This coordinate remapping transformation is determined by two non-negative integers, `jpx-iset` and `jpx-inum`. Together, these two integers identify a specific compositing instruction within a JPX composition (comp) box, found within the scope of the logical target. The specific instruction in question is located in the instruction set (iset) box whose ordinal position (starting from 0) within the composition box is given by the jpx-iset value. The jpx-inum value gives the ordinal position (starting from 0) of the instruction within that instruction set box. The interpretation of these indices is independent of repeat counts which may appear within a JPX composition box.

When jpx-iset and jpx-inum values are processed by the server, the requested frame size and region parameters fx, fy, sx, sy, ox and oy, shall first be mapped to modified frame size and region parameters $\overline{\mathrm{fx}}, \overline{\mathrm{fy}}\,\overline{\mathrm{sx}}, \overline{\mathrm{sy}}, \overline{\mathrm{ox}}, \overline{\mathrm{oy}}$, using the expressions in equation (3). These modified region parameters shall be calculated separately for each requested codestream and shall then be used in place of fx, fy, sx, sy, ox and oy when determining the codestream image resolution and the codestream image region following the procedure described in C.4.1.

equation (3)

$$\overline{\mathrm{fx}} = \left\lceil \mathrm{fx} \cdot \frac{\mathrm{XR}_{\mathrm{reg}}}{\mathrm{XS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Wt}_{\mathrm{inst}}}{\mathrm{Ws}_{\mathrm{inst}}} \cdot \frac{\mathrm{Xsiz}}{\mathrm{XS}_{\mathrm{comp}}} \right\rceil; \quad \overline{\mathrm{fy}} = \left\lceil \mathrm{fy} \cdot \frac{\mathrm{YR}_{\mathrm{reg}}}{\mathrm{YS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Ht}_{\mathrm{inst}}}{\mathrm{Hs}_{\mathrm{inst}}} \cdot \frac{\mathrm{Ysiz}}{\mathrm{YS}_{\mathrm{comp}}} \right\rceil$$

$$\overline{\mathrm{ox}} = \left\lfloor \max\{\mathrm{ox}, x_{\min}\} - x_{\mathrm{off}} \right\rfloor; \quad \overline{\mathrm{oy}} = \left\lfloor \max\{\mathrm{oy}, y_{\min}\} - y_{\mathrm{off}} \right\rfloor$$

$$\overline{\mathrm{sx}} = \left\lceil \min\{(\mathrm{ox} + \mathrm{sx}), x_{\mathrm{lim}}\} \right\rceil - \overline{\mathrm{ox}}; \quad \overline{\mathrm{sy}} = \left\lceil \min\{(\mathrm{oy} + \mathrm{sy}), y_{\mathrm{lim}}\} \right\rceil - \overline{\mathrm{oy}}; \quad \text{where}$$

$$x_{\mathrm{off}} = \frac{\mathrm{XO}_{\mathrm{reg}}}{\mathrm{XS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Wt}_{\mathrm{inst}}}{\mathrm{Ws}_{\mathrm{inst}}} \cdot \frac{\mathrm{fx}}{\mathrm{XS}_{\mathrm{comp}}}; \quad y_{\mathrm{off}} = \frac{\mathrm{YO}_{\mathrm{reg}}}{\mathrm{YS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Ht}_{\mathrm{inst}}}{\mathrm{Hs}_{\mathrm{inst}}} \cdot \frac{\mathrm{fy}}{\mathrm{YS}_{\mathrm{comp}}}$$

$$x_{\min} = x_{\mathrm{off}} + \mathrm{XO}_{\mathrm{inst}} \cdot \frac{\mathrm{Wt}_{\mathrm{inst}}}{\mathrm{Ws}_{\mathrm{inst}}} \cdot \frac{\mathrm{fx}}{\mathrm{XS}_{\mathrm{comp}}}; \quad y_{\min} = y_{\mathrm{off}} + \mathrm{YO}_{\mathrm{inst}} \cdot \frac{\mathrm{Ht}_{\mathrm{inst}}}{\mathrm{Hs}_{\mathrm{inst}}} \cdot \frac{\mathrm{fy}}{\mathrm{YS}_{\mathrm{comp}}}$$

$$x_{\mathrm{lim}} = x_{\min} + \mathrm{Wt}_{\mathrm{inst}} \cdot \frac{\mathrm{fx}}{\mathrm{XS}_{\mathrm{comp}}}; \quad \text{and} \quad y_{\mathrm{lim}} = y_{\min} + \mathrm{Ht}_{\mathrm{inst}} \cdot \frac{\mathrm{fy}}{\mathrm{YS}_{\mathrm{comp}}}$$

Here, XS_{comp} and YS_{comp} are the dimensions of the composited result, supplied in the JPX composition options box (see IS 15444-2, Annex M.11.10.1); XC_{inst} and YC_{inst} are the cropping offsets supplied via the relevant

instruction (see IS 15444-2, Annex M.11.10.2.1); Ws_{inst} and Hs_{inst} are the cropped width and height and Wt_{inst} and Ht_{inst} are the composited width and height, again described via the relevant compositing instruction (see IS 15444-2, Annex M.11.10.2.1); XS_{reg} and YS_{reg} are the registration precisions described at the beginning of any codestream registration box (see IS 15444-2, Annex M.11.7.7); XO_{reg} and YO_{reg} are the codestream registration offsets and XR_{reg} and YR_{reg} the codestream registration sampling factors, described at the beginning of any codestream registration box (see IS 15444-2, Annex M.11.7.7).

If `jpx-iset` and `jpx-inum` values are not supplied, the modified region parameters to be used in place of fx, fy, sx, sy, ox and oy are given by the expressions in equation (4). As before, these modified parameters shall be used when determining the codestream image resolution and the codestream image region, following the procedure in C.4.1.

equation (4)

$$\overline{\mathrm{fx}} = \left\lceil \mathrm{fx} \cdot \frac{\mathrm{XR}_{\mathrm{reg}}}{\mathrm{XS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Xsiz}}{\mathrm{W}_{\mathrm{reg}}} \right\rceil; \quad \overline{\mathrm{fy}} = \left\lceil \mathrm{fy} \cdot \frac{\mathrm{YR}_{\mathrm{reg}}}{\mathrm{YS}_{\mathrm{reg}}} \cdot \frac{\mathrm{Ysiz}}{\mathrm{H}_{\mathrm{reg}}} \right\rceil$$

$$\overline{\mathrm{ox}} = \left\lfloor \mathrm{ox} - x_{\mathrm{off}} \right\rfloor; \quad \overline{\mathrm{oy}} = \left\lfloor \mathrm{oy} - y_{\mathrm{off}} \right\rfloor$$

$$\overline{\mathrm{sx}} = \mathrm{sx} + \mathrm{ox} - \overline{\mathrm{ox}}; \quad \overline{\mathrm{sy}} = \mathrm{sy} + \mathrm{oy} - \overline{\mathrm{oy}}; \quad \text{where}$$

$$x_{\mathrm{off}} = \frac{\mathrm{XO}_{\mathrm{reg}}}{\mathrm{XS}_{\mathrm{reg}}} \cdot \frac{\mathrm{fx}}{\mathrm{W}_{\mathrm{reg}}}; \quad y_{\mathrm{off}} = \frac{\mathrm{YO}_{\mathrm{reg}}}{\mathrm{YS}_{\mathrm{reg}}} \cdot \frac{\mathrm{fy}}{\mathrm{H}_{\mathrm{reg}}}$$

Here, W_{reg} and H_{reg} are the width and height of the compositing layer, as it appears on the compositing layer registration grid. The expressions in equation (4) may equivalently be obtained by setting $XS_{comp}=Ws_{inst}=Wt_{inst}=W_{reg}$, $YS_{comp}=Hs_{inst}=Ht_{inst}=H_{reg}$ and $XO_{inst}=YO_{inst}$ in equation (3).

The second type of context-range described by this International Standard, mj2t-context, allows clients to request specific tracks from an MJ2 file. The mj2-track identifier must be a strictly positive integer, since 1 is the smallest allowable track identifier permitted within an MJ2 file. If an mj2-track identifier includes the optional “+now” suffix, the mj2t-context consists of all codestreams belonging to the MJ2 video track, starting with the codestream whose capture time corresponds to the time at which the request is received. This is useful when the source is a live video stream. Otherwise, the server may associate “now” with any codestream it sees fit. If the “+now” suffix is not included, the mj2-context consists of all codestreams belonging to the MJ2 video track.

An mj2t-context may specify a coordinate remapping transformation, to be used in deducing codestream image resolutions and codestream image regions for each of its codestreams. If not present, the frame size and region parameters supplied via Frame Size, Offset and Region Size request fields shall be interpreted directly following the procedure outlined in C.4.1. Otherwise, one of two types of coordinate transformation is being requested, as identified by the appearance of one of the “track” or “movie” tokens.

Where “track” is specified, the Frame Size, Offset and Region Size request fields are being used to identify a desired presentation size and a desired rectangular region within the smallest bounding rectangle which contains the track’s presentation, at this desired presentation size. The geometric transformations described by the MJ2 Track Header (tkhd) box shall be applied to determine a corresponding image resolution and region on each codestream associated with the track.

Where “movie” is specified, the Frame Size, Offset and Region Size request fields are being used to identify a desired size for the entire (possibly composited) reproduced movie, and a desired rectangular region within the smallest bounding rectangle which contains the movie, at this desired size. The geometric transformations described by the MJ2 Track Header (tkhd) box shall be combined with the geometric transformations described by the Movie Header (mvhd) box and applied to determine a corresponding image resolution and region on each codestream associated with the track.

In the event that a server is unable to apply any of the mj2t-context geometric transformations described above, it provide a modified mj2t-context string in its Codestream Context response header.

Note: the use of the Codestream Context request field together with the Codestream request field may result in a codestream being requested multiple times with different geometric transformations of the Frame Size,

Region Size and Offset request fields. Where this happens, multiple disjoint or overlapping image portions of that codestream are effectively being requested.

EXAMPLE 1 “context=jpxl<0-4:2>[s5i2]”
In this case, the server is requested to return the codestreams which are used by JPX compositing layers 0, 2 and 4, remapping the requested frame size and image region according to the geometric adjustments represented by the third instruction of the sixth instruction set box within the composition box (JPX files have at most one composition box).

EXAMPLE 2 “stream=0&context=mj2t<1+now>[track]”
In this case, the server is requested to return codestream 0, as well as all codestreams belonging to the first track of an MJ2 file, starting from the codestream whose sampling time corresponds to the current time. Moreover, the server is requested to remap the requested frame size and image region according to the geometric adjustments described in the Track Header box, disregarding any additional geometric adjustments which may be described in the Movie Header box.

C.4.8 Sampling Rate (srate)

```
srate = "srate" "=" streams-per-second

streams-per-second = UFLOAT
```

If this request field is supplied, the codestreams which belong to the view-window are obtained by sub-sampling those mentioned by the Codestream request field, in addition to those expanded from context-range values in the Codestream Context request field, so as to achieve an average sampling rate no greater than the streams-per-second value. This is possible only if the codestreams have associated timing information (e.g., if they belong to a logical target conforming to the MJ2 file format).

This request field serves only to determine which codestreams should be considered to belong to the view-window. The server shall scan through all codestreams which would otherwise be included in the view-window, discarding codestreams as required to ensure that the average separation between codestream source times is no less than the reciprocal of the streams-per-second value. This international standard does not prescribe an algorithm for subsampling, or a precise interpretation for the term “average separation.”

If no source timing information is available the view-window will consist of all codestreams identified via the Codestream request field the Codestream Context request field, but this request field may nonetheless affect the interpretation of a Delivery Rate request field, if present.

C.4.9 ROI (roi)

```
roi = "roi" "=" region-name

region-name = 1*(DIGIT | ALPHA | "_")
            | "dynamic"
```

This field specifies the desired spatial region of the image through a name rather than through coordinates. The mapping between `region-name` and a specific spatial region of the image may come from several places; it may be defined within an ROI description box within the logical target, or it may be defined within the implementation of the server itself.

A `region-name` value of “dynamic” (a dynamic ROI) is reserved to represent a non-constant region within the image that is mapped to a spatial region independently for each and every request. The server may use any information about the client and any other request parameters when it determines what spatial region it will provide for that particular request. For example, if the server knows that the physical display on the client is very small, it may choose to provide only the foreground region of the image at a higher resolution rather than the entire region of the image at a lower resolution. Servers are not required to support dynamic ROI’s.

If an ROI field exists, and the server knows how to handle the ROI request, then the ROI field takes precedence over the Offset request field and the Region Size fields, which shall be ignored by the server. If an

ROI field exists, but the server does not know how to handle it for any reason, the server shall ignore the ROI field and use the Offset and Region Size fields. If these fields are omitted, the default values of those fields shall be used.

If the client specifies a Frame Size as well as an ROI, and the server understands the ROI specified, the value of the Frame Size request field determines the image resolution at which the ROI is requested.

C.4.10 Layers (layers)

```
layers = "layers" "=" UINT
```

This field may be used to restrict the number of codestream quality layers that belong to the View-Window request. By default, all available layers are of interest. The value specifies the number of initial quality layers that are of interest. The server should not attempt to augment any precinct data-bins beyond the relevant layer boundary. The server should not attempt to augment any tile data-bins beyond the point at which all remaining contents lie beyond the relevant layer boundary. Due to the order of data within a tile, it may be necessary for the server to return data beyond the boundary of the requested layer for JPT-stream requests only.

C.5 Metadata request fields

C.5.1 Metadata requested implicitly with view-window requests

The Codestream request field and the Codestream Context request field identify one or more codestreams which are associated with the requested view-window. Even if neither of these request fields, the view-window is associated with at least one codestream, as mentioned in C.4.6. Moreover, as noted in C.4.2, even if the Frame Size request field is omitted, the requested view-window includes at least the main codestream header for each requested codestream. The only exception to this is when `metadata-only` is specified in a metadata request field (see C.5.2). Except in this case, the client is also implicitly requesting whatever metadata boxes may be required from the file format, if any, in order to utilize the imagery represented by the requested codestreams. To ensure interoperability between client and server components, this section identifies a minimal set of metadata which servers shall regard as being implicitly requested along with the view-window. Where the server is aware of additional relevant metadata elements, it may deliver these as well.

For JP2 and JPX files, the following metadata elements shall be considered to be requested along with the view-window.

1. The entire contents of metadata-bin 0
2. The entire contents of each of the following boxes, wherever they are found at the top level of the file:
    a. JP2 Signature (“jP ”)
    b. File Type (“ftyp”)
    c. Reader Requirements (“rreq”)
    d. Composition (“comp”)
3. All immediate sub-box headers from each of the following superboxes:
    a. any JP2 Header (jp2h) box;
    b. any Codestream Header (jpch) box associated with a requested codestream;

c. any Compositing Layer Header (jplh) box associated with a JPX compositing layer requested via the Codestream Context request field;

4. The entire contents of each of the following boxes, wherever these boxes are found within one of the superboxes mentioned above:

   a. Image Header (“ihdr”)

   b. Bits per Component (“bpcc”)

   c. Palette (“pclr”)

   d. Component Mapping (“cmap”)

   e. Channel Definition (“cdef”)

   f. Resolution (“res ”)

   g. Codestream Registration (“creg”)

   h. Opacity (“opct”)

5. For JP2 and JP2 compatible files, the first Colour Description (“colr”) box of the JP2 Header box

6. At least one of the Colour Description (“colr”) boxes associated with each JPX compositing layer requested via the Codestream Context request field.

[DST Note: We should probably try augmenting the above list to accommodate MJ2 files]

The server is requested to return an initial prefix of each metadata-bin which contains any of the metadata mentioned above, extending from the first byte of the metadata-bin and continuing to the end of all requested metadata from that metadata-bin. As a result, the actual amount of metadata returned by the server may depend upon the particular way in which the logical target has been partitioned into metadata-bins. A discussion of these issues may be found in A.3.6.2.

C.5.2 Metadata Request (metareq)

```
metareq = "metareq" "=" 1#("[" 1$(req-box-prop) "]" [root-bin] [max-depth])
                        [metadata-only]

req-box-prop = box-type [limit] [metareq-qualifier] [priority]

limit = ":" (UINT | "r")

metareq-qualifier = "/" 1*("w" | "s" | "g" | "a")

priority = "!"

root-bin = "R" UINT

max-depth = "D" UINT

metadata-only = "!!"
```

The Metadata Request request field specifies what metadata is desired in response to this request, in addition to any metadata required for the client to decode or interpret requested image data (see C.5.1). The value string in this request field is a list of independent requests; however, the server may handle the requests as a group, and there may be overlap between the requests.

Each request is relative to the data-bin specified by its `root-bin` value. If a `root-bin` value is not specified, the root is metadata-bin 0. The request pertains only to data within or referenced (through Placeholder boxes) by that particular data-bin.

If a value for `max-depth` is specified, then only boxes contained within the root metadata-bin, and those no more than `max-depth` levels in the file hierarchy below that box are requested. If a value for `max-depth` is not specified, there is no limit on the depth of the file hierarchy for this request.

The `req-box-prop` portion of the request specifies a list of box types that are of interest to the client. The special string "*" may be substituted for the box type, in which case all box types are implied. Each box type (or "*") may be followed by any combination of three attributes: a limit value, a metareq qualifier, and a priority flag.

`limit` specifies what type of information, and how much of the box contents the client is requesting for that box type. The limit parameter takes the form of a colon followed by a value, which shall either be an integer or the character "r".

If the value of `limit` is an integer *n* greater than zero, then the server is requested to return at least the first *n* bytes of the contents of relevant boxes of that box type, in addition to the box headers. If the value of `limit` is 0, then only the box headers for boxes of that type are requested. If `limit` is not specified, then the client is requesting the entire contents of all boxes of that box type which match other aspects of the request, regardless of whether this box is a superbox or not. Also, in the case of a numeric or not specified `limit` value on a superbox, the server is requested to provide the amount of data requested by `limit` regardless of whether or not the hierarchy contained within that superbox is deeper than would have been reached based on the values of `root-bin` and `max-depth`, and regardless of the box types of sub-boxes found within the superbox.

If the value of `limit` is "r" then the server is requested to send the box header for any box with the indicated box type, as well as all of its descendant sub-boxes (regardless of their box type), down to the maximum depth specified in the request. This is in effect a request for a skeleton of that portion of the box hierarchy. If the server is unable to determine whether or not a box is a superbox, it might not be able to recurse into the box's sub-boxes, so that it might not respond completely to some metadata requests. Servers should be able to recognize the superbox status of all boxes defined by the file formats they are intended to support.

While a limit value of "r" means that the client is requesting a skeleton of the box structure, consisting of the box headers, the division of the logical target into metadata-bins may force the server to return additional data, including the contents of some boxes and the headers and/or contents of other non-requested boxes. This is because the server is requested to return all bytes from the start of each metadata-bin which contains requested box bytes up to the last requested box byte.

The `metareq-qualifier` takes the form of a "/" followed by one or more of the flags "g", "s", "w" and "a". Each flag identifies a context from which boxes which match the request shall be drawn. The interpretation for each of these contexts is supplied in Table C.2. If more than one of the flags is provided, the union of the corresponding contexts shall be taken. If no `metareq-qualifier` is provided, the union of the "g", "s" and "w" contexts shall be used. By way of clarification, note that contexts "g", "s" and "w" are mutually exclusive, but their union is generally smaller than the catch-all context "a".

Table C.2 — Metadata request qualifier flags

Flag	Interpretation
"w"	This metareq context includes all boxes which are known to be associated with a specific spatial image region within one or more codestreams which belong to the view-window, where the spatial region, resolutions and the image components to which the boxes relate intersect with those of the view-window. Such an association might, for example, be established by an "asoc" box in a JPX file.
"s"	This metareq context includes all boxes which are known to be

	associated with one or more codestreams which belong to the view-window, or with one or more of the requested codestream contexts (e.g. JPX compositing layers or MJ2 video tracks), where these boxes are not solely associated with particular spatial regions. Such an association might be established by an "asoc" box in a JPX file, for example.
“g”	This metareq context includes all boxes which are relevant to the requested view-window, taking into account the requested codestreams and the requested codestream contexts, excluding those boxes which are included in the “w” and “s” metareq contexts.
“a”	This metareq context includes all boxes in the logical target, without exception. NOTE This metareq context is suitable for requests that wish to interrogate the file structure independently of the view-window.

If the `priority` flag is specified, then the client is requesting that boxes of type `box-type` which match other elements of the request be delivered with higher priority than the image data.

For any box type not specified in the `req-box-prop` list, no data is requested for boxes of that box type.

If `metadata-only` is specified at the end of the metadata request field, the client is requesting that the server’s response consist only of metadata, without any image data or codestream headers, regardless of whether view-window request fields such as Frame Size have been used. For JPP-stream and JPT-stream return types, this means that the returned JPIP messages will all be metadata-bin messages.

EXAMPLE 1 “metareq=[*]R31D4”
In this case, the server is requested to return the entire contents of all boxes it finds in the contents of bin 31. While a restriction on the desired depth has been specified, the server shall ignore that restriction because the contents of those boxes were not limited through the `limit` parameter.

EXAMPLE 2 “metareq=[*:r,drep]R31D4”
The “*:r” means that the server has been requested to return box headers (in the form of Placeholder boxes) for all boxes contained in metadata-bin 31 and any bins referenced by placeholders contained within that bin, up to a depth of 4 levels from the contents of bin 31, but not to include the contents of those boxes. The additional “drep” `req-box-prop` specifies that the server is requested to return the entire contents of any “drep” box contained within metadata-bin 31 and any bins referenced by placeholders within that bin, up to a depth of 4 levels from the contents of bin 31.

EXAMPLE 3 “metareq=[drep]R31D4”
In this case, the server is still requested to return the entire contents of any “drep” boxes it finds in the contents of bin 31 or any bins referenced by that bin, up to a depth of four levels from the contents of bin 31. However, because no other boxes were specified, the server is requested to send only as much other data (generally in the form of Placeholder boxes) in order to specify the position of any “drep” box in the file hierarchy with respect to the box contained within metadata-bin 31.

Regardless of the box specifications provided via the Metadata Request field, the server may send other data, either because it has determined that the other data is required for the client to decode or interpret the requested image data, or because the server had previously divided the logical target into data-bins using different criteria and additional data shall be sent in order to provide a consistent and meaningful view of the metadata-bins for this logical target.

C.6 Data limiting request fields

C.6.1 Maximum Response Length (len)

```
len = "len" "=" UINT
```

This field specifies a restriction on the amount of data the client wants the server to send in response to this request. The unit shall be bytes. If not present, the server should send image data to the client until such

point as all of the relevant data has been sent, a quality limit is reached (see C.6.2), or the response is interrupted by the arrival of a new request that does not include a Wait request field with a value of “yes” (see C.7.2).

C.6.2 Quality (quality)

```
quality = "quality" "=" (1*2DIGIT | "100")        ; 0 to 100
```

This field may be used to limit data transmission to a quality level (between 0 for lowest quality and 100 for highest quality) associated with the image. Quality limits are difficult to formulate in a reliable manner, and the server may ignore this request by responding with a value “-1”. (See D.2.15). Nevertheless, it is useful to allow the client to provide some indication of the maximum image quality that might be of interest. The quality factor may attempt to approximate the ad-hoc Quality commonly used to control JPEG compression. The client should expect that the returned data size is monotonically non-decreasing with increasing quality, i.e., increasing the quality value generally corresponds to increasing the returned data size.

NOTE If a server supports this request and two different clients make identical requests to a same target in the same server having a same quality value, e.g. "`quality=80`", the server should have a consistent implementation policy in returning data-bins.

C.7 Server control request fields

C.7.1 Alignment (align)

```
align = "align" "=" ("yes" | "no")
```

This field specifies whether the server response data shall be aligned on natural boundaries. The default value is “no”. If the value is "yes", any JPT-stream or JPP-stream message delivered in response to this request which crosses any “natural boundary” shall terminate at any subsequent “natural boundary.” The natural boundaries for each data-bin type are listed in Table C.3. A message is said to cross a natural boundary if it includes the last byte prior to the boundary, and the first byte after the boundary. For example, a precinct data-bin crosses a natural boundary if it includes the last byte of one packet and the first byte of the next packet. Note carefully that aligned response messages are not actually required to terminate at a natural boundary unless they cross a boundary. This means, for example, that the response may include partial packets from precincts, which may be necessary if a prevailing byte limit prevents the delivery of complete packets.

Table C.C3 — Alignment boundaries based on bin type

Bin type	Natural boundary
Precinct data-bin	End of a packet (one boundary for each quality layer)
Tile data-bin	End of a tile-part (one boundary for each tile-part)
Tile header data-bin	End of the bin (only one boundary)
Main header data-bin	End of the bin (only one boundary)
Metadata-bin	End of a box at the top level of the data-bin (one boundary for each box)

C.7.2 Wait (wait)

```
wait = "wait" "=" "yes" | "no"
```

If the value of the field is "yes”, the server shall completely respond to the previous request on the same channel resource specified through the channel ID field before starting to respond to this request.

If the value of this field is “no”, the server may gracefully terminate the processing of any previous request on the same channel resource (specified through the Channel ID field) prior to completion and may start to respond to this new request. In this context, “graceful termination” implies that the server shall at least complete the current message.

The default value of this field is “no”.

C.7.3 Image Return Type (type)

```
type = "type" "=" 1#image-return-type

image-return-type = media-type | reserved-image-return-type

media-type = TOKEN "/" TOKEN 0*( ";" parameter )

reserved-image-return-type = TOKEN 0*( ";" parameter )

parameter = attribute "=" value

attribute = TOKEN

value = TOKEN
```

The Image Return Type request field is used to indicate the type (or types) of the requested return data. A server unwilling to provide any of the requested return types shall issue an error response.

The value of the Image Return Type request field shall be either a media type (defined in RFC 2046) or one of the reserved image return types defined in Table C.4.

Table C.C4 — Legal image return types

Type	Interpretation
“jpp-stream”	A JPP-stream as defined in Annex A. “jpp-stream” may optionally be followed by “;ptype=ext”, in which case the requested return type is one in which all precinct data-bin message headers have the extended form. (See A.3.2)
“jpt-stream”	A JPT-stream as defined in Annex A. “jpt-stream” may optionally be followed by “;ttype=ext”, in which case the requested return type is one in which all tile data-bin message headers have the extended form. (See A.3.2)
“raw”	The client is requesting the entire sequence of bytes in the logical target to be delivered unchanged.
Other values	Reserved for ISO use

If the `type` request field is omitted, the return type should be determined by another means.

In a session, i.e., one whose requests involve a Channel ID request field, the value of the return parameter shall be maintained in successive responses for image data or metadata requests which correspond to the same logical target.

C.7.4 Delivery Rate (drate)

```
drate = "drate" "=" rate-factor

rate-factor = UFLOAT
```

If this field is supplied, the server shall deliver data belonging to the various codestreams in the view-window following a temporally sequenced schedule. The codestreams which belong to the view-window are all those identified via the Codestream request field and the Codestream Context request field, possibly sub-sampled in accordance with the Sampling Rate request field.

In order to provide meaning to this request field, timing information shall be associated with the various codestreams in the view-window. If the codestreams belong to an MJ2 file, the timing information is provided by that file. The MJ2 file provides a mapping between each codestream and a nominal playback time, which is identified here as the "source time."

If the codestreams do not have source timing information, but the Sampling Rate request field is present, the server shall assume that codestreams in the view-window have source times which are separated by the reciprocal of the value in the Sampling Rate request field.

If the codestreams do not have source timing information, and the Sampling Rate request field is not present, the server shall assume that the codestreams in the view-window have source times which are separated by exactly one second.

The Delivery Rate request field provides a scaling factor between delivery and source rates. If the rate-factor is given as 1, the server should attempt to deliver codestreams to the client at the rate suggested by their source times, noting that these source times might not necessarily be regular. More generally, if the rate-factor is *F*, the server should attempt to deliver codestreams to the client at a rate which is *F* times faster than that suggested by their source times.

If the server is unable to deliver all relevant data for each codestream at the requested rate (e.g. due to bandwidth constraints), it should deliver only part of the data for each codestream, so as to avoid violating the requested delivery rate. The portion of each codestream's data which is not delivered may depend upon the view-window-pref value supplied in a Client Preferences request field. If the preference is "progressive" or no such preference is identified, the server should attempt to deliver a uniform, maximum image quality over the view window, subject to the delivery rate constraint. If a view-window-pref value of "fullwindow" has been supplied, the server might truncate the representation associated with each codestream in some other way. In any event, the behaviour should be similar to that which would have resulted from the client issuing a succession of requests for each of the relevant codestreams in turn, at the delivery rate.

If the server is able to deliver all relevant data for each codestream, at the requested rate, it should idle the connection as required to ensure that the delivery rate is not exceeded.

If this field is not supplied and if a view-window-pref value of "fullwindow" has not been specified, the server should attempt to sequence the relevant data in such a way as to progressively increment the quality of all codestreams uniformly.

C.8 Cache management request fields

C.8.1 The “model” Request Field

C.8.1.1 General

```
model = "model" "=" 1#model-item

model-item = model-element | codestream-qualifier

model-element = ["-"] bin-descriptor

bin-descriptor = explicit-bin-descriptor | implicit-bin-descriptor

codestream-qualifier = "[" 1$(codestream-range) "]"

codestream-range = first-codestream-id ["-" last-codestream-id]

first-codestream-id = UINT

last-codestream-id = UINT
```

The “model” request field may be used in stateful or stateless requests. A stateful request is any request that includes a Channel-ID field, since channels are associated with a session managed by the server. The “model” field contains one or more bin-descriptors, each of which identifies a data-bin, or a range of data-bins, about which cache information is being signalled. For requests within a session, this cache information serves to update the server’s model of the client’s cache. There is only one cache model for each logical target associated with the session. For a stateless request, the server’s model of the client’s cache is empty at the start of the request, but is updated by the “model” field (if one exists) before the server formulates its response. All cache model information is discarded at the end of a stateless request.

The bin-descriptor values that explicitly refer to data-bins are of the following types: M (metadata-bins), Hm (main header data-bins), H (tile header data-bins), P (precinct data-bins) or T (tile data-bins). The “H*” descriptor therefore includes all tile header data-bins, but does not include the main codestream header data-bin, “Hm”.

If a bin-descriptor is preceded by a “-” symbol, it is said to be subtractive. Otherwise, it is said to be additive. A subtractive bin-descriptor informs the server that the relevant data should be removed from the server’s model of the client cache. Removal of elements from the cache model means that the server shall not assume that the client already has these elements. Bin-descriptor values are processed in order.

An additive bin-descriptor (one which is not preceded by the “-” symbol) informs the server of data which the client already has in its cache. The server may add this information to its cache model and may assume that the client already has the indicated data.

Two forms are provided for bin-descriptor values to facilitate the efficient exchange of cache model information. These are termed the “explicit” and the “implicit” forms and are described in the following sub-sections. Clients may issue requests using either form and may mix the two forms of bin-descriptor within a single “model” request field if desired.

The “model” field may reference data-bins that are not relevant to the view-window of interest identified by other request fields (Frame Size, Region Size, Offset, etc.). Where this happens, the cache model manipulation might not affect the response to the current request, but may nevertheless affect the response to future requests (unless the request is stateless).

Wherever the list of model-items includes a codestream-qualifier, all subsequent model-elements shall be added or subtracted (as appropriate) from all codestreams whose identifiers are listed by the codestream-qualifier. Codestream-qualifiers may be interspersed throughout the list to progressively alter the collection of codestreams that are to be affected by the ensuing model-elements. Any model-element that is not preceded

by a codestream-qualifier applies to the first codestream requested via a Codestream request field. If no Codestream request field is present, model-element values which are not preceded by a codestream-qualifier shall refer to codestream 0, regardless of whether or not a Codestream Context request field is included.

Requests within a session may not include any codestream-qualifier which references more than a single codestream.

NOTE 1 The server should attempt to exploit additive cache model manipulation statements, but is free to disregard some or all of them at the possible expense of transport efficiency. Clients should be aware that servers might be quite likely to disregard additive cache model manipulation statements that refer to data-bins belonging to codestreams that will not be serviced by the current request. To remove such uncertainties where multiple codestreams are involved, the "mset" request field may be used to determine the set of codestreams which are being modelled.

NOTE 2 Manipulation of a stateful server's cache model generally affects the response to both the current request and any future requests. Moreover, all channels within a session that are associated with a single logical target share the same cache model. Thus, "model" fields in requests that arrive using one channel (Channel ID field) may affect the response to requests that arrive using a different channel. It is important to bear in mind that requests which use different JPIP channels (different Channel ID values) may arrive asynchronously at the server, if separate TCP channels are used to transport the request either directly from the client or indirectly at an intermediate proxy. Clients should take whatever action is necessary to ensure that their cache model manipulation instructions remain meaningful in light of these considerations.

C.8.1.2 Explicit Form

```
explicit-bin-descriptor = explicit-bin
                          [":" (number-of-bytes | number-of-layers )]

explicit-bin = codestream-main-header-bin
             | meta-bin
             | tile-bin
             | tile-header-bin
             | precinct-bin

number-of-bytes = UINT

number-of-layers = "L" UINT

codestream-main-header-bin = "Hm"

meta-bin = "M" bin-uid

tile-bin = "T" bin-uid

tile-header-bin = "H" bin-uid

precinct-bin = "P" data-uid

bin-uid = UINT | "*"
```

Explicit bin-descriptors identify the relevant data-bin (or data-bins) within the relevant codestreams, using either a unique integer-valued identifier, or a wildcard character, "*". The only exception to this is the codestream main header data-bin, whose bin-descriptor is "Hm". For all other data-bin classes, the unique identifier is identical to the value communicated by the in-class identifier in JPP-stream and/or JPT-stream message headers. (SeeAnnex A).

The wildcard character, "*", may only be used in stateless requests. Where it is used, the bin-descriptor refers simultaneously to all data-bins of the relevant class (metadata, precinct, tile header or tile).

Each bin-descriptor may be qualified by a number of bytes. An additive bin-descriptor which is qualified by the number of bytes, *B*, indicates only that the client already has the first *B* bytes of the indicated data-bin in its

cache; the server may add the first *B* bytes of the data-bin to its cache model. A subtractive bin-descriptor that is qualified by the number of bytes, *B*, indicates that the client has at most the first B bytes of the indicated data-bin; the server shall remove any bytes following the first *B* bytes of the data-bin from its cache model. A qualified subtractor bin-descriptor such as "-P23:10" means only that the server should remove all but the first 10 bytes of precinct data-bin 23 from its cache model. This does not imply that the client has the first 10 bytes of precinct data-bin 23 in its cache and the server should not add these bytes to its cache model.

Precinct bin-descriptors may alternatively be qualified by a number of layers. An additive bin-descriptor that is qualified by the number of layers, *L*, indicates that the client already has the first *L* layers (first *L* packets) of the indicated precinct; the server may add the bytes corresponding to these layers to its cache model. A subtractive precinct bin-descriptor that is qualified by the number of layers, *L*, indicates that the client has at most the first *L* layers (*L* packets) of the indicated precinct; the server shall remove the bytes corresponding to any subsequent layers of that precinct from its cache-model.

EXAMPLE 1 "model=M0,Hm,H0:20,P0" means that the client has all of metadata-bin 0, all of the main codestream header, the first 20 bytes of tile header 0, and all of precinct 0 in its cache.

EXAMPLE 2 "model=P1:256,P5:L2,-P6:20" means that the client has the first 256 bytes of precinct 1 and the first two layers (packets) of precinct 5, but it does not have anything beyond the 20th byte of precinct 6 (it may not have the first 20 bytes either).

EXAMPLE 3 "model=M*,-M5,-H*,-P*:L3" means that the client has (or is prepared to let the server believe it has) all metadata-bins except metadata-bin 5, no tile header data-bins which are relevant to the view window and at most the first 3 layers of any precinct which is relevant to the view window. Note, that the wildcards used here are permissible only when the "model" statement appears in a stateless request.

EXAMPLE 4 "model=[30-200],Hm,H*,M*,P0,[0-29],-Hm,-H*,-M*,-P*" means that the client has all headers and metadata, plus precinct data-bin 0 from codestreams 30 through 200 inclusive, but that it has removed all header, metadata and precinct data-bins from the first 30 codestreams.

C.8.1.3 Implicit Form

```
implicit-bin-descriptor = 1*implicit-bin [":" number-of-layers]

implicit-bin  = implicit-bin-prefix (data-uid | index-range-spec)

implicit-bin-prefix = "t"    ; tile
                    | "c"    ; component
                    | "r"    ; resolution level
                    | "p"    ; position

index-range-spec = first-index-pos "-" last-index-pos

first-index-pos = UINT

last-index-pos = UINT

data-uid = UINT | "*"
```

Implicit bin-descriptors are used to identify precinct data-bins via the indices of the tile to which the precinct belongs, the image component to which the precinct belongs, the resolution level of the tile-component to which the precinct belongs, and the position of the precinct within its tile-component-resolution. All indices start from 0. A resolution level index of 0 refers to the lowest resolution level (LL subband) of the tile-component. Position indices run from left to right and top to bottom of the tile-component-resolution progression, in scan-line fashion, as described in ITU–T Rec. T.800 | ISO/IEC 15444–1.

In stateless requests, any or all of the tile, component, resolution level or position implicit-bin specifier may be omitted or the index range may be replaced with the wildcard character, "*". In either case the bin-descriptor is expanded to include all values of the relevant index. Neither of these options shall be used for requests within a session.

In stateless requests, any or all of the tile, component, resolution level or position indices may also be replaced with a single range of indices. The first-index-pos value in an index-range-spec gives the first index in a range. The last-index-pos value gives the last index in the range and shall be greater than or equal to the value of the first-index-pos. Both indices specified are inclusive. The last-index-pos may not be omitted. If a range of tile indices ("t") is given, the range refers to a rectangular array of tiles whose upper left-hand corner has the first-index-pos value and whose lower right-hand corner has the last-index-pos value. Similarly, if a range of position indices ("p") is given, the range refers to a rectangular array of precinct positions whose upper left and lower right corners are given by the first-index-pos and last-index-pos values, respectively. As for wildcards, ranges shall not be used in requests within a session.

Implicit precinct bin-descriptors may be qualified by a number of layers, for which the syntax and interpretation are identical to those of layer qualified explicit precinct bin-descriptors, described previously.

EXAMPLE 1 "model=t0c2r3p4:L5" indicates that the client has the first 5 packets of the 5th precinct in sequence, of the fourth resolution level, of the third component, of tile 0.

EXAMPLE 2 "model=t10r0,t*r1:L4" means that the client has all layers of the tile index 10 at resolution level 0, and the first 4 layers of all tiles relevant to view-window at resolution level 1. Note that the wildcard is appropriate only for stateless requests.

EXAMPLE 3 "model=t0-10:L2" indicates that the client has the first 2 layers from tiles 0 to 10. Note that the range is appropriate only for stateless requests.

EXAMPLE 4 "model=t*r0-2:L4" indicates that the client has the first 4 layers from resolution levels 0 to 2 of all the tiles relevant to the view-window.

C.8.2 Summary of Cache descriptor options (Informative)

Table C.5 — Cache descriptor option summary

	Wildcard		**Range**	**number-of-layers** (e.g. ":L3")	**number-of-bytes** (e.g. ":256")
	stateless	**stateful**			
Explicit form	Allowed	Not allowed	Not allowed	Allowed	Allowed
Implicit form	Allowed	Not allowed	Allowed	Allowed	Not allowed

C.8.3 The "tpmodel" Request Field involving JPT-streams

```
tpmodel =  "tpmodel" "=" 1#tpmodel-item

tpmodel-item = tpmodel-element | codestream-qualifier

tpmodel-element = ["-"] tp-descriptor

tp-descriptor = tp-range | tp-number

tp-range = tp-number "-" tp-number

tp-number = tile-number "." part-number

tile-number = UINT

part-number = UINT
```

The “tpmodel” (tile-part model) request field may be used to indicate specific tile-parts that the client would like to add to or subtract from the server’s cache model. Like the “model” field, it may be used in both stateful and stateless requests. In the case of stateless requests, the cache model is empty at the start of the request and does not persist between requests, but it still provides a useful mechanism for identifying the image elements which are already in the client’s cache.

If a tile-part descriptor is preceded by a “-” character, it is said to be subtractive. Otherwise it is additive. An additive tile-part descriptor indicates that the client already has the indicated tile-part or range of tile-parts in its cache; the server may add these elements to its cache-model. A subtractive tile-part descriptor indicates that the client does not have the indicated tile-part or range of tile-parts in its cache; the server shall remove these elements from its cache-model.

The first value in the tile-part number is the tile index (starting from 0); the second value is the part number (starting from 0) within the tile. A tp-range is considered to independently contain tiles from the first tile number to the second tile number and tile-parts from the first tile-part number to the second tile-part number. Thus 4.0-5.1 includes tile-parts 4.0, 4.1, 5.0, and 5.1, but not 4.2.

The “tpmodel” and “model” request fields may both appear within a single request. In this case, however, the the server shall reflect the effects of the “model” field on its cache model before processing the “tpmodel” field.

Codestream-qualifier values may be interspersed amongst the list of tpmodel-elements in order to alter the collection of codestreams to which the ensuing tpmodel-elements apply, following exactly the same principles as for the “model” request field.

Note that unlike the “model” request field, ranges of tile-parts and ranges of codestreams (in codestream-qualifiers) are both permitted within the “tpmodel” request field, regardless of whether is appears within a stateful or a stateless request.

EXAMPEL 1 “tpmodel=4.0,4.1,5.0-6.2” indicates that the client already has the first two tile-parts of tile 4, and the first 3 tile-parts of tiles 5 and 6 in its cache.

EXAMPLE 2 “tpmodel=-4.0-6.254” indicates that the client has no tile-parts from tiles 4, 5 or 6 in its cache.

EXAMPLE 3 “tpmodel=3.0,[131-133],4.0,[100],-0.0-65534.254” indicates that the client has tile-part 0 of tile 3 from all codestreams referenced in the request, plus tile-part 0 of tile 4 from each of codestreams 131 through 133 inclusive, and that it is deleting all tile-parts from its cache of codestream 100.

C.8.4 The “need” Request Field for Stateless Requests

```
need = "need" "=" 1#need-item

need-item = bin-descriptor | codestream-qualifier
```

The “need” request field may appear only in stateless requests, i.e., those which do not include a Channel ID request field. It has the same syntax as the model request field, except that bin-descriptors shall not be preceded by a “-” symbol. The “need” request field shall not appear within the same request as a “model” or “tpmodel” request field.

The “need” request field indicates the set of data-bins (or data-bin suffices) which are of potential interest to the client. The server need not send information that is not of potential interest. Regardless of how large the set of potentially interesting data-bins may be, the server should only send information which is relevant to the view-window request fields or the metadata request field.

The effect of the “need” field on the server’s request may be explained using the concept of a temporary cache model. The temporary cache model is initialized (empty) immediately before the request is processed and discarded after the response is generated. If a “need” field appears in the request, all possible data-bins are added into the cache model, after which all elements referenced by the bin-descriptors in the “need” field are removed from the cache model. The server then processes the requested view-window, using the cache model to determine the elements that need not be sent to the client.

Codestream-qualifiers may be interspersed amongst the list of bin-descriptors in order to alter the collection of codestreams to which the ensuing bin-descriptors apply, following exactly the same principles as for the "model" and "tpmodel" request fields.

EXAMPLE 1 "need=M1,H0:20,P0" means that the client needs all metadata-bin 1, data from the 20th byte of tile header data-bin 0 and all of precinct data-bin 0.

EXAMPLE 2 "need=P1:256,P5:L2" means that the client needs data beyond the 256th byte (or from byte 256) of precinct data-bin 1, and layers beyond the 2nd layer (or from layer index 2) of precinct data-bin 5.

EXAMPLE 3 "need= H*,P*:L3" means that the client needs all tile header data-bins relevant with the view-window and layers beyond the 3rd layer (or from layer index 3) of all precinct data-bins relevant with the view-window.

EXAMPLE 4 "need=t10r0,t*r1:L4" means that the client needs all layers of the tile index 10 at resolution level 0, and layers beyond the 4th layer (or from layer index 4) of all tiles relevant to view-window at resolution level 1.

EXAMPLE 5 "need=t*r0-2:L4" means that the client needs all layers from layer 4 (layer index 4, 5, …) of all the precinct data-bins in resolution levels 0 to 2 (0, 1 and 2) in all the tiles and components relevant to the view-window request.

EXAMPLE 6 "need=[120-131],r0,[140;143-145],r0-1" means that the client needs resolution level 0 of codestreams 120 through 131 inclusive, and resolution levels 0 and 1 of codestreams 140 and 143 through 145 inclusive.

C.8.5 The "tpneed" Request Field for Stateless Requests

```
tpneed = "tpneed" "=" 1#tpneed-item

tpneed-item = tp-descriptor | codestream-qualifier
```

The "tpneed" request field may appear only in stateless requests, i.e., those which do not include a Channel ID request field. It has the same syntax as the tpmodel request field, except that tp-descriptors shall not be preceded by a "-" symbol. The "tpneed" request field shall not appear within the same request as a "model" or "tpmodel" request field.

The "tpneed" request field indicates the set of tile-parts which are of potential interest to the client. The server need not send information that is not of potential interest. Regardless of how large the set of potentially interesting tile-parts may be, the server should only send information which is relevant to the view-window request fields or the metadata request field.

The effect of the "tpneed" field on the server's request may be explained using the concept of a temporary cache model. The temporary cache model is initialized (empty) immediately before the request is processed and discarded after the response is generated. If a "tpneed" field appears in the request, all possible tile-parts and data-bins are added into the cache model, after which all elements referenced by the bin-descriptors in the "need" field and all tile-parts in the "tpneed" field are removed from the cache model. The server then processes the requested view-window, using the cache model to determine the elements that need not be sent to the client.

Codestream-qualifiers may be interspersed amongst the list of tile-parts in order to alter the collection of codestreams to which the ensuing tile-parts apply, following exactly the same principles as for the "model" and "tpmodel" request fields.

C.8.6 The "mset" Request Field for Requests within a session

```
mset =  "mset" "=" 1#sampled-range
```

The "mset" request field serves two purposes. In the first instance it informs the server of the set of codestreams for which the client is prepared to cache data delivered by the server. In the second instance it provides a mechanism for the client to learn about the codestreams for which the server is prepared to model the client's cache. Specifically, if the collection of codestream indices supplied in a "mset" request differs in

any way from the set of codestreams over which the server is currently prepared to offer cache modelling, the server shall provide a “mset” response header, as discussed in D.2.17.

The “mset” request field’s parameter string consists of a comma-separated list of ranges of codestream indices, possibly sub-sampled, following the conventions outlined in connection with the Codestream request field in C.4.6.

In addition to codestreams mentioned in the “mset” request, the server may also provide a cache model for all codestreams associated with its response to the current request. This is the collection of codestreams identified by the client’s request (see the Codestream and Codestream Context request fields C.4.6), unless the server indicates a reduced set of codestreams via a Codestream response header (see D.2.8). If no “mset” request field is provided, the client should not assume that the server is providing a cache model for any codestreams other than those associated with its response; however, it may model other codestreams. If an “mset” request field is given, the server shall discard any cache model information it has for all codestreams other than those mentioned either in the “mset” request, or in the set of codestreams associated with its response data. Moreover, the effects of any cache model manipulation via “model” or “tpmodel” request fields shall be restricted to just these codestreams.

The server may, at its discretion, reduce the number of codestreams in the “mset”, in which case it shall supply a “mset” response header identifying the actual set of codestreams which are being modelled; moreover, this set of modelled codestreams shall at least include all codestreams associated with the server’s response data (those requested by the client’s request, or identified by the server’s Codestream response header, if any). In this case, these statements apply to those codestreams contained in “mset” identified by the server. The server may not identify a larger set of codestreams than those mentioned in the client’s “mset” request, combined with those codestreams which are associated with the server’s response data.

Note that the server may change its “mset” from request to request, so clients which need to keep track of and/or tightly constrain the server’s “mset” might choose to include an “mset” request field with every request.

C.9 Upload request parameters

C.9.1 Upload (upload)

```
upload = "upload" "=" upload-type

upload-type = image-return-type                    ; Annex C.7.3
```

This field specifies that the client is uploading new image or metadata to the server. The value of `upload-type` can be any of the valid `image-return-type` values that could be used with the type request field. See Annex E for information on uploading data.

C.10 Client capability and preference request fields

C.10.1 Client Capability (cap)

```
cap = "cap" "=" 1#(capability-group)

capability-group = processing-capability |
                   depth-capability |
                   config-capability

processing-capability = compatibility-capability |
                        vendor-capability

compatibility-capability = "cc." compatibility-code  ; 7.3
```

```
vendor–capability = "vc." UUID

depth-capability = "depth:" UINT

config-capability = "config:" UINT
```

The Client Capability field specifies the software capabilities of the client. For session based requests (those which include a Channel ID request field), any capability fields transmitted by the client shall affect only the channel associated with the request, and shall be considered persistent. Capabilities need not be retransmitted by the client for subsequent requests on the same channel.

When a new channel is created from an existing channel, its client capabilities are inherited. For stateless requests, and for requests issued within a channel whose capabilities have never been specified or inherited, the client capabilities may be determined or anticipated by other means. The capabilities associated with a channel may be changed by including a Client Capabilities request field within any request.

If the Client Capabilities request field identifies one or more of the `processing-capability` options, the server shall assume that the client does not have any of the other `processing-capability` options which could have been mentioned. If no `processing-capability` options are supplied in the Client Capabilities request field, the server shall continue to use whatever previous knowledge it had concerning processing capabilities. The `processing-capability` options defined by this international standard are described in Table C.5.

Table C.C6 — Legal capabilities in the Capability request field

Capability	Meaning
`"cc." compatibility-code`	The client supports all files that contain `compatibility-code` in the compatibility list in the File Type box. For example, to indicate that the client supports all JP2 files, the client would transmit the “c.jp2_” in a Capability request field. A compatibility-code value of "jp2c" shall be used to indicate support for raw JPEG 2000 codestreams.
`"vc." vc`	The client supports the vendor capability defined by `vc`. `vc` shall be a string specifying the reverse domain name of the vendor that defined the feature, followed by the vendor feature name. For example, if example.com defined a feature called “distance”, then the value of `vc` for this feature shall be “com.example.distance”. An optional value parameter may be specified, as defined by the particular vendor feature.

If a `depth-capability` parameter is supplied, it indicates the maximum sample bit-depth (precision) at which the client is able to exploit decompressed imagery. If the client supports different bit-depths for different image components, this field shall specify the bit depth of the highest capability component. Note that Clients having the capability to handle only *N* bits per sample will still generally be able to handle codestreams whose SIZ marker indicates a bit depth much larger than *N*. However, this flag may be used by the server to determine an appropriate manner in which to deliver the requested image data.

If a `config-capability` parameter is supplied, it shall be in the range 0 to 255, representing an 8-bit word whose individual bits are interpreted as configuration flags. The interpretation of the configuration flags is provided in Table C.6.

Table 1 — Legal values of the config-capability pa

Value	Meaning
1xxx yyyy	The client is capable of processing colour image data.
0xxx yyyy	The client is not capable of processing colour image data and desires the server to transmit any requested image regions as greyscale.

x1xx yyyy	The client has a pointing device for end-user interaction
x0xx yyyy	The client does not have a pointing device for end-user interaction
xx1x yyyy	The client has a keyboard for end-user interaction
xx0x yyyy	The client does not have a keyboard for end-user interaction
xxx1 yyyy	The client has sound output capabilities
xxx0 yyyy	The client does not have sound output capabilities
Other values	Reserved for ISO use

A bit value of “x” in Table C.7 indicates that the specified value includes cases where that bit is set to either “1” or “0”. Bits indicated as “y” are unused by this standard and shall be set to 0 by clients and ignored by servers.

C.10.2 Client Preferences (pref)

C.10.2.1 General

```
pref = "pref" "=" 1#(related-pref-set ["/r"])

related-pref-set = view-window-pref            ; Annex C.10.2.2
                 | colour-meth-pref            ; Annex C.10.2.3
                 | max-bandwidth               ; Annex C.10.2.4
                 | bandwidth-slice             ; Annex C.10.2.5
                 | placeholder-pref            ; Annex C.10.2.6
                 | codestream-seq-pref         ; Annex C.10.2.7
                 | other

other = TOKEN
```

This field specifies the client preferences for server behaviour. For session based requests (those which include a Channel ID request field), any preference fields transmitted by the client shall affect only the channel associated with the request, and shall be considered persistent. Preferences need not be retransmitted by the client for subsequent requests on the same channel. Each preference shall occur no more than once in a single preference request field.

When a new channel is created from an existing channel, its preferences are inherited. For stateless requests, and for requests issued within a channel whose preferences have never been specified or inherited, the client preferences may be determined or anticipated by other means. If the client desires to change its preferences, it shall send the entire affected `related-pref-set` again.

Unless otherwise stated, each `related-pref-set` specifies an ordered list of individual preference tokens, from most preferred to least preferred. Where possible, the server shall respect the client preferences identified by this request field. If a `related-pref-set` is followed by the “/r” modifier, the server shall either support one of the preferences listed in the `related-pref-set`, or else it shall respond with an error. In the latter case, the server shall return an Unavailable preference response header which identifies any `related-pref-set` which had the “/r” modifier but could not be supported. See Annex D.2.19for more on the Unavailable preference response header.

For example, consider the following Client Preferences request:

```
pref=fullwindow/r,color-ricc:2;color-icc
```

This preference request requires that the server return the complete view-window requested, regardless of how large that view-window may be (see Annex C.10.2.2for a discussion of the “fullwindow” preference). Since the “/r” modifier has been used, the server should return an error response unless it is able to support

this preference. In addition, the client prefers to use Restricted ICC profiles rather than arbitrary ICC profiles, provided the Restricted ICC profile is at least of "exceptional quality." (See Annex C.10.2.3 for a discussion of colour space preferences).

A server shall ignore any value for `related-pref-set` that it does not understand and is not immediately followed by “/r”. If the not-understood value is followed by “/r”, then the server shall return the Client Preferences return header, indicating the preference that it is not able to perform.

Values of the token `other` are reserved for ISO use.

C.10.2.2 View window handling preferences

```
view-window-pref = "fullwindow" | "progressive"
```

This standard defines two options to specify the behaviour of the server in the event that the request cannot be serviced exactly as stated, following a quality-progressive ordering of the return data. These two options are specified in Table C.7:

Table C.C8 — View window handling preferences

Option	Meaning
“fullwindow”	The server shall honour the view-window request parameters but is allowed to return the data in a non-quality-progressive order.
“progressive”	The server may modify the view-window request parameters in order to retain quality-progressive properties of the return data. In the event that the server does modify view-window request parameters, the modified view-window should be a subset of the originally requested view-window.

If neither "fullwindow" nor "progressive" is specified in the Client Preferences request field, the server shall assume that the client’s preference is "progressive".

Note that the interpretation of “progressive” delivery may be affected by the presence of a Delivery Rate request field, as explained in C.7.4.

C.10.2.3 Colour space method preference

```
color-meth-pref = 1$(color-meth [":" meth-limit])

color-meth = "color-enum" | "color-ricc" | "color-icc" | "color-vend"

meth-limit = UINT
```

This standard defines four options that specify what forms of colour space specification data should be returned by the server. A single JPEG 2000 file may contain multiple specifications of the colour space for a single codestream or compositing layer. This allows a file writer to provide the optimal colour space specification while still providing interoperable solutions.

However, not all readers will support all colour space methods, and the data provided for some colour space methods may be of significant size. In those cases, the server should only send the colour space specification data that is desired by the client.

If the Client Preferences request field does not contain any colour space method preferences, then the supported colour space methods are defined according to the information contained within the Capability field, and no preference is defined.

Each colour space method preference consists of two parts: the particular colour space method, and an optional limit on that preference. Legal values of the colour space method are specified in Table C.8.

Table C.C9 — Colour space method client preferences

Method	Meaning
“color-enum”	The client prefers colour space specifications that use the Enumerated Method
“color-ricc”	The client prefers colour space specifications that use the Restricted ICC Method
“color-icc”	The client prefers colour space specifications that use the Any ICC Method
“color-vend”	The client prefers colour space specifications that use the Vendor Method

The optional `meth-limit` value specifies a limit on the APPROX value for that particular colour space method. When using these preferences to select a colour space specification, the server shall consider a colour space method specification with an APPROX value of `meth-limit` or less as if the actual APPROX value was 1 (exact). This allows clients to specify the point at which colour fidelity is not important for a particular colour space method, for the current application. For example, a page-layout application that is only concerned with aligning the image data with other elements on the page may not care at all about colour fidelity and set `meth-limit` to 4, meaning that the accuracy of the colour space methods is unimportant. Another application that displays images on a low-quality screen may set `meth-limit` to 2, to indicate that as long as the colour accuracy is reasonable, it would be satisfied. The characters of the field shall be interpreted as an unsigned decimal integer. Legal values are defined by the definition of the APPROX field in Table M–24 of ITU–T Rec. T.801 | ISO/IEC 15444–2, and by extensions and amendments to that standard.

When selecting which Colourspace Specification box to transmit to the client, the server shall use the following algorithm, as shown in Figure C.4.

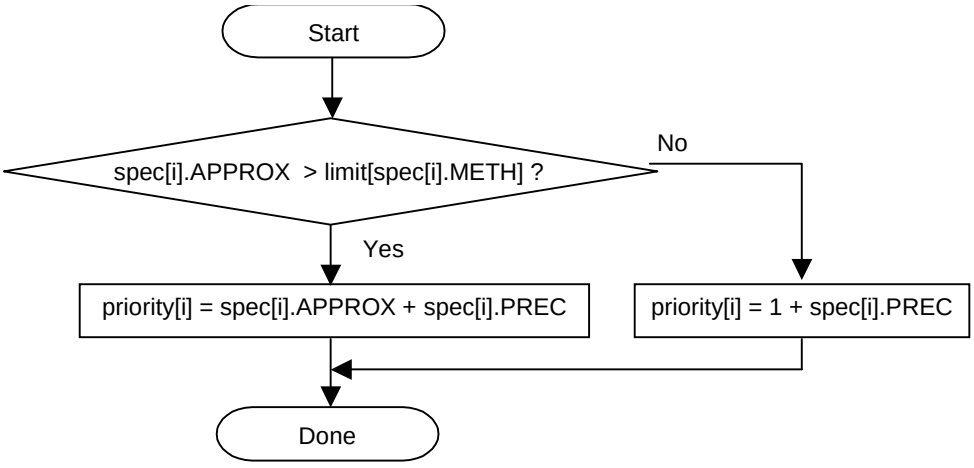

Figure C.C4 — Colourspace specification box selection procedure

For each Colourspace Specification box which uses a method that is supported by the client, where:

— spec[] is an array containing all of the Colourspace Specification boxes from the given logical target.

— spec[i].APPROX is the value of the APPROX field for the ith Colourspace Specification box as it appears in the logical target

— spec[i].METH is the value of the METH field for the ith Colourspace Specification box as it appears in the logical target

— spec[i].PREC is the value of the PREC field for the ith Colourspace Specification box as it appears in the logical target

— limit[] is an array containing the `meth-limit` values specified in the request field, indexed by the legal values of the METH field in the Colourspace Specification box.

— priority[] is an array of calculated priority values for each Colourspace Specification box in the given logical target. priority[i] corresponds to spec[i].

If the server knows that the client does not support a particular Colourspace Specification box, then the server shall ignore that box for purposes of selecting the preferred Colourspace Specification box. Once the priority[] values have been calculated for each supported Colourspace Specification box, the server shall select the box with the lowest priority value. In the event that multiple boxes have a priority value equal to the minimum value for this logical target, the server shall select the colour space method using the following preference order:

1. Enumerated method
2. Vendor method
3. Restricted ICC method
4. Any ICC method.

Regardless of the client preferences for Colourspace Specification boxes, the server may return more Colourspace Specification boxes than the single colour box specified by this algorithm, depending upon the division of a file into metadata-bins.

C.10.2.4 Max bandwidth

```
max-bandwidth = "mbw:" mbw

mbw = UINT [ "K" | "M" | "G" | "T"]
```

This preference signals the maximum rate at which the client would like to be sent data over the network. If the `mbw` value ends in "K" the value is in kilobits/second, where 1 kilobit = 1024 bits. If the `mbw` value ends in "M" the value is in megabits/second, where 1 megabit = 1024×1024 bits. If the `mbw` value ends in “G”, the value is in gigabits/second, where 1 gigabit = 1024×1024×1024 bits. If the `mbw` value ends in “T”, the value is in terabits, where 1 terabit = 1024×1024×1024×1024 bits. Otherwise, the value is in bits/second.

C.10.2.5 Bandwidth slice

```
bandwidth-slice = "slice:" slice

slice = UNIT
```

This preference may be used to identify the fraction of the available bandwidth that should be allocated to this channel. The value of `slice` shall be strictly greater than 0. The bandwidth fraction is obtained by dividing each channel’s slice value by the sum of all channel slice values. If not specified, the channel’s slice value defaults to 1.

As an example, a low `slice` value could be used for requesting a “background” view-window, while a higher `slice` might be used for a “foreground” view-window. If the session contains channels that are associated with different logical targets, slice values affect the proportion of the available bandwidth which is assigned to these different targets (images).

C.10.2.6 Placeholder preference

```
placeholder-pref = "meta:" placeholder-branch

placeholder-branch = "incr" | "equiv" | "orig"
```

This preference may be used to indicate the preferred treatment of Placeholder boxes. Where Placeholder boxes appear within the metadata in a JPP-stream or JPT-stream, there may be as many as three different representations of the content of a box: the original box; a streaming equivalent box; and an incremental codestream (signalled via the index). These possibilities are explained in Annex A.3.6 and Annex A.4. As explained in Annex A.4, the recommended default assumption is that the client would prefer to receive the incremental codestream, if available, failing which it would prefer to receive the streaming equivalent box, if available. The client may signal an alternate preference using the mechanism described here.

A value of “orig” means that the client would prefer to receive the original box, if available. Failing that, the client would prefer to receive a streaming equivalent box, if available.

A value of “equiv” means that the client would prefer to receive a streaming equivalent box, if available. Failing that, the client would prefer to receive the original box, if available.

A value of “incr” means that the client would prefer to receive the incremental codestream data-bins, if available. Failing that, the client would prefer to receive the streaming equivalent box, if available. This is the same as the recommended default policy.

It is not legal to provide more than one value for the placeholder preference.

C.10.2.7 Codestream sequencing

```
codestream-seq-pref = "codeseq:" codestream-seq-option

codestream-seq-option = "sequential"
                      | "interleaved"
```

This preference may be used to indicate how the client desires that the server deliver multiple codestreams that have been requested within a single request.

A value of “sequential” means that the client would prefer to receive the multiple codestreams in a frame sequential order (i.e. multiple frames in a Motion JPEG 2000 file).

A value of “interleaved” means that the client would prefer to receive the multiple codestreams in an interleaved manner (i.e. multiple compositing layers in a JPX file).

It is not legal to provide more than one value for the codestream sequencing preference.

C.10.3 Constrast sensitivity (csf)

```
csf = "csf" "=" 1#(csf-sample-line)

csf-sample-line = csf-density [";" csf-angle] ";" 1$sensitivity

csf-density = "density" ":" UFLOAT

csf-angle = "angle" ":" UFLOAT

sensitivity = UFLOAT
```

This request field may be used to supply information concerning contrast sensitivity. While this information may represent the effects of both visual sensitivity and the modulation transfer function of a display device, it is most easily described in terms of an assumed hypothetical modulation transfer function. When reproduced at the frame size identified by the Frame Size request field, the imagery is assumed to be passed through a device whose modulation transfer function (MTF) is $m(\omega_1,\omega_2)$, after which it is viewed by a subject whose human visual system has a perfectly uniform contrast sensitivity function. The MTF $m(\omega_1,\omega_2)$ is described through a collection of samples. The samples are logarithmically spaced in the radial direction, along one or more oriented axes. The server may interpolate these samples using any method it sees fit, in order to

recover the MTF, which in turn may be used to adjust the order in which byte ranges of data-bins are communicated to the client through JPP-stream or JPT-stream messages.

Each csf-sample-line represents MTF samples $m(\omega_1, \omega_2)\big|_{\omega_1=\pi d^n \cos\psi, \omega_2=\pi d^n \sin\psi}$, where *n* is the sample index, starting from *n*=0 for the first csf-density sample in the csf-sample-line, ψ is the orientation of the CSF sample line, expressed in degrees (defaults to 0 if there is no csf-angle value), and *d* is the sampling density; it shall be no larger than 1.0. The ω_1 value describes the horizontal frequency in radians, where $\omega_1 = \pi$ is the horizontal Nyquist frequency. The ω_2 value describes the vertical frequency in radians, where $\omega_2 = \pi$ is the vertical Nyquist frequency.

The MTF sample values have meaning only in relation to each other; there is no particular interpretation for their absolute values.

Annex D
(normative)

Server response signalling

D.1 Reply syntax

D.1.1 Reply structure

The JPIP response consists of the following elements:

— status-code

— reason-phrase

— jpip-response-header

— return data

The elements in the response should comply with the selected transport protocol. As an example, in HTTP, the status code and the reason phrase appear in the status line, the JPIP response headers appear in the HTTP response headers and the return data (if any) appears in the HTTP entity-body.

```
Status-Code = 3DIGIT;

Reason-Phrase = 0*<TEXT, excluding CR and LF>

jpip-response-header =
                      | JPIP-tid                    ; Annex D.2.2
                      | JPIP-cnew                   ; Annex D.2.3
                      | JPIP-fsiz                   ; Annex D.2.4
                      | JPIP-rsiz                   ; Annex D.2.5
                      | JPIP-roff                   ; Annex D.2.6
                      | JPIP-comps                  ; Annex D.2.7
                      | JPIP-stream                 ; Annex D.2.8
                      | JPIP-context                ; Annex D.2.9
                      | JPIP-roi                    ; Annex D.2.10
                      | JPIP-layers                 ; Annex D.2.11
                      | JPIP-srate                  ; Annex D.2.12
                      | JPIP-metareq                ; Annex D.2.13
                      | JPIP-len                    ; Annex D.2.14
                      | JPIP-quality                ; Annex D.2.15
                      | JPIP-type                   ; Annex D.2.16
                      | JPIP-mset                   ; Annex D.2.17
                      | JPIP-cap                    ; Annex D.2.18
                      | JPIP-pref                   ; Annex D.2.19
```

The reason-phrase string should ideally impart a textual explanation of the status code. The following status codes may be sufficient for JPIP applications.

D.1.2 Status codes and reason phrases

D.1.2.1 General

The `status code` is a 3-digit integer result code of the attempt to understand and satisfy the request. A subset of the status codes and reason phrases from HTTP/1.1 are used. JPIP clients should expect the following codes. JPIP clients operating over HTTP may see other status codes as well.

D.1.2.2 200 (OK)

The server should use this status code if it accepts the view-window request for processing, possibly with some modifications to the requested view-window, as indicated by additional headers included in the reply.

D.1.2.3 202 (Accepted)

Servers should issue this status code if the view-window request was acceptable, but a subsequent view-window request was found in the queue, which rendered the current request irrelevant. This is a common occurrence in practice, since an interactive user may change his/her region of interest multiple times before the server finishes responding to an earlier request, or before the server is prepared to interrupt ongoing processing.

D.1.2.4 400 (Bad Request)

Servers should issue this status code if the request is incorrectly formatted, or contains an unrecognized field in the query string.

D.1.2.5 404 (Not Found)

This status code should be issued if the server cannot reconcile the requested resource with an issued Target ID. This may result from unauthorized access attempts or, more likely, from a time limit expiring. If the client misses this time window, due to a poor connection, it may find that the Target ID is no longer active.

D.1.2.6 415 (Unsupported media type)

This status code may be used if the single image type specified in the Image Return Type request field cannot be serviced.

D.1.2.7 501 (Not implemented)

This status code may be used if an optional portion of this International Standard that is required by the request cannot be serviced.

D.1.2.8 503 (Service unavailable)

This status code should be used if a channel id specified in the Channel ID request field is invalid.

D.2 JPIP Response headers

D.2.1 Introduction to JPIP Response Headers

In responding to a client request, the server may be permitted to modify some aspects of the request. Wherever this happens, the modified parameters shall be identified via response headers. The name of each response header is derived from the name of the request field whose parameters are being modified, by prefixing the name of the request field with “JPIP-“. As a general rule, if the parameters identified in the response header had been used in the client’s request, the server would have responded in the same way

without issuing the response header. Parameters to the derived response header indicated by the same BNF element as parameters in the original request field have the same meaning and formatting as the parameters to the original request field.

The only exceptions to this general rule are found in connection with the New Channel and Quality response headers.

D.2.2 Target ID (JPIP-tid)

```
JPIP-tid = "JPIP-tid" ":" LWS target-id
```

The server should send this response header if the server's unique target identifier differs in any way from the identifier supplied with a Target ID request field. The `target-id` is an arbitrary, server-assigned string, not exceeding 255 characters in length. If the Target ID request field specifies a value of "0", the server is obliged to include a Target ID response header, indicating the actual target-id. If the Target ID response header specifies 0, the server is unable to assign unique identifiers to the requested logical target and hence cannot guaranty its integrity between multiple requests or sessions. If the server supplies a `target-id` which is different from that specified in the request, it should disregard all `model`, `tpmodel`, and `need` request fields when responding this request.

D.2.3 New Channel (JPIP-cnew)

```
JPIP-cnew = "JPIP-cnew" ":" LWS "cid" "=" channel-id
            ["," 1#(transport-param "=" TOKEN)]

transport-param = TOKEN
```

The server should send this response header if and only if it assigns a new channel in response to a New Channel request field. The value string consists of a comma-separated list of name=value pairs, the first of which identifies the new channel's channel-id token.

The following `transport-param` tokens are defined by this International Standard (Table D.1).

Table D.D1 — Legal values of transport-param

Value	Meaning
"transport"	This parameter shall be assigned one of the values in the list of acceptable transport names supplied in the New Channel request field. If multiple transport names were supplied in the request field, the response header shall identify the actual transport that will be used with the channel.
"host"	This parameter identifies the name or IP address of the host for the JPIP server that is managing the new channel. The parameter need not be returned unless the host differs from that to which the request was actually sent.
"path"	This parameter identifies the path component of the URL to be used in constructing future requests with this channel. The parameter need not be returned unless the path name differs from that used in the request which was actually sent.
"port"	This parameter identifies the numerical port number (decimal) at which the JPIP server that is managing the new channel is listening for requests. The parameter need not be returned if the host and port number are identical to those to which the original request was sent. The parameter also need not be returned if the host differs from that to which the request was sent and the default port number associated with the relevant transport is to be used.

Table D.D1 — Legal values of transport-param

Value	Meaning
“auxport”	This parameter is used with transports requiring a second physical channel. If the “http-tcp” transport is used, the auxiliary port is used to connect the auxiliary TCP channel. For further details, see Annex G. The parameter need not be returned if the original request involved a channel that also employed an auxiliary channel, having the same auxiliary port number. Otherwise, the parameter need be returned only if the auxiliary port number differs from the default value associated with the selected transport.

D.2.4 Frame Size (JPIP-fsiz)

```
JPIP-fsiz = "JPIP-fsiz" ":" LWS fx "," fy
```

The server should send this response header if the frame size for which response data will be served differs from that requested via the Frame size request field.

D.2.5 Region Size (JPIP-rsiz)

```
JPIP-rsiz = "JPIP-rsiz" ":" LWS sx "," sy
```

The server should send this response header if the size of the region for which response data will be served differs from that requested.

D.2.6 Offset (JPIP-roff)

```
JPIP-roff = "JPIP-roff" ":" LWS ox "," oy
```

The server should send this response header if the offset of the region for which response data will be served differs from that requested.

D.2.7 Components (JPIP-comps)

```
JPIP-comps = "JPIP-comps" ":" LWS 1#(UINT-RANGE)
```

The server should send this response header if the components for which it will serve data differ from those requested via the Components request field. It is not obliged to send this request field if requested image components do not exist within any of the requested codestreams.

D.2.8 Codestream (JPIP-stream)

```
JPIP-stream = "JPIP-stream" ":" 1#(sampled-range)
```

The server should send this response header to inform the client of the codestream or codestreams for which it will serve data, unless these codestreams are identical to those specified explicitly via the Codestream request field and implicitly via the Codestream Context request field. A codestream identifier range shall be open ended (no upper bound supplied after the “-“ character of a sampled-range) if and only if there is no way for the server to know how many codestreams will be in the range. This might happen, for example, while sending live video.

D.2.9 Codestream Context (JPIP-context)

```
JPIP-context = "JPIP-context" "=" 1$(context-range "=" 1#sampled-range)
```

The server should send this response header if it is able to process any of the context-range values supplied via a Codestream Context request field. The header describes each context-range which is being processed, along with the indices of all codestreams which are associated with that context-range. The server may omit some context-range values which were originally provided in the Codestream Context request field, if they are not being processed. The server may also modify context-range values originally provided in the Codestream Context request field. Two types of modification are allowed: 1) the server may restrict the collection of image elements (e.g., compositing layers) which were originally requested; 2) the server may drop geometric transformation modifiers which it is not able to support (e.g., a “track” or “movie” modifier within an mj2t-context string).

D.2.10 ROI (JPIP-roi)

```
JPIP-roi = "JPIP-roi" ":" LWS
           "roi" "=" region-name ";"
           "fsiz" "=" UINT "," UINT ";"
           "rsiz" "=" UINT "," UINT ";"
           "roff" "=" UINT "," UINT ";"

region-name = 1*(DIGIT | ALPHA | "_")
```

In response to a client request for an ROI, a server shall specify through the ROI response header the extent of the ROI actually being served. If the server is unable to fulfil the ROI request, it shall reply with the ROI response header simply set to: "JPIP-roi: roi=no-roi". In addition to the ROI, the server also specifies through the Frame size, Region size and Offset response headers the region of the image that it is serving as a fallback.

If the server is able to serve the ROI, but for some reason needs to resize the portion of the returned image, it shall send the ROI response header describing the ROI and the Frame size, Region size and Offset response headers describing the part of the ROI being returned.

D.2.11 Layers (JPIP-layers)

```
JPIP-layers = "JPIP-layers" ":" LWS UINT
```

The server should send this response header if the number of layers for which it will serve is smaller than the value specified by the layers request field. Since the view-window is typically served in quality progressive fashion, the server is not obliged (and indeed may not be able) to determine the number of layers which are spanned by the response data it delivers. However, if the requested number of layers exceeds the number of layers available from any codestreams in the view-window, the server should at least identify the maximum number of available layers. Any server that accepts an aligned request (C.7.1) shall provide a JPIP-layers response if the number of layers for which it will serve is smaller than the value specified by the layers request field.

D.2.12 Sampling Rate (JPIP-srate)

```
JPIP-srate = "JPIP-srate" ":" LWS UFLOAT
```

The server should send this response header if the average sampling rate of the codestreams which it will send to the client is expected to differ from that requested via a Sampling Rate request field and the sampling rate is known. If the source codestreams have no timing information, this response header should not be sent.

D.2.13 Meta request (JPIP-metareq)

```
JPIP-metareq = "JPIP-metareq" ":" LWS
               1#( "[" 1$(req-box-prop) "]" [root-bin] [max-depth] )
               [metadata-only]

req-box-prop = box-type [limit] [metareq-qualifier] [priority]
```

The server should send this response header if it is modifying the `max-depth`, `limit`, `metareq-qualifier` or `priority` value provided in a Metadata Request request field.

D.2.14 Maximum Response Length (JPIP-len)

```
JPIP-len = "JPIP-len" ":" LWS UINT
```

The server should send this response header if the byte limit specified in a Maximum Response Length request field was too small to allow a non-empty response unless the byte limit was equal to zero.

D.2.15 Quality (JPIP-quality)

```
JPIP-quality = "JPIP-quality" ":" LWS (1*2DIGIT | "100" | "-1")
```

The server may send this response header to inform the client of the quality value that will be associated with the image data returned once this request has been completed. If the request is interrupted by another request (not having “wait=yes”), this quality value may not be accurate. The quality value refers only to the view-window requested, and has the same interpretation as the quality request field. If the server ignored the client’s request, a value “-1” shall be returned.

D.2.16 Image Response Type (JPIP-type)

```
JPIP-type = "JPIP-type" ":" LWS image-return-type
```

The server should include this response header unless another mechanism identifies the MIME subtype of the return image data. Examples of other mechanisms include:

— an HTTP “Content-Type:” header,

— Responses to requests that are associated with a session whose return image type has already been signalled.

D.2.17 Model Set (JPIP-mset)

```
JPIP-mset = "JPIP-mset" ":" LWS 1#(sampled-range)
```

The server should include this response header if the client’s request contains a “mset” field, and the collection of codestreams identified by the client’s “mset” request field differ in any way from the collection of codestreams for which the server is actually prepared to maintain cache model information. The set of codestreams for which the server maintains cache model information should include all codestreams which are associated with the server’s response data (either those identified in the client’s request, or those identified by the server’s Codestream response header, if any). Apart from those codestreams, the server’s “mset” may be no larger than that identified by the client’s “mset” request field.

D.2.18 Needed Capability (JPIP-cap)

```
JPIP-cap = "JPIP-cap" ":" LWS 1#(capability-class)
```

This response header specifies that the client shall support a particular feature in order to interpret the logical target in a conformant manner. Legal capabilities are the same as those defined for the Capability request field in Table C. 5 in Annex C.10.1

D.2.19 Unavailable Preference (JPIP-pref)

```
JPIP-pref = "JPIP-pref" ":" LWS 1#(related-pref-set)
```

This response header should be provided if and only if a Client Preferences request field contained a `related-pref-set` with the "/r" modifier (required), which the server was unwilling to support. In this case, an error value should also be returned for the response status code. The value string consists of one or more of the `related-pref-sets` that could not be supported, repeated in exactly the same form as they appeared in the Client Preferences request.

Although desirable, it is not necessary for this response header to list all of the required `related-pref-sets` that cannot be supported. Thus, it is permissible for a server to walk into the Client Preferences request field only until it encounters a `related-pref-set` which specifies "/r" and cannot be supported. See C.10.2.1 for more information on when this response header is to be used.

D.3 Response data

For anything other than the JPP- or JPT-stream image return types, the return data should consist of the requested entity in full. For JPP- or JPT-stream image return types, the return data consist of a sequence of messages as defined in Annex A, terminated by a single EOR (End Of Response) message. The EOR message is not defined in Annex A and is not formally part of the JPP or JPT-stream media types.

An EOR message consists a header and a body. The EOR message header consists of the single byte identifier, 0x00, followed by a single byte reason code, R, and then a single VBAS byte count, indicating the number of bytes in the body of the EOR message. This International Standard provides no normative interpretation for the contents of the EOR message body.

Note that the EOR message body does not contribute to the byte count restriction associated with the max length request field as defined in Annex C.

Note that the EOR means that the server has delivered all the pertinent contents of the relevant data-bins for a client request. This is not necessarily the entire contents of those data-bins. The response is terminated when a client specified limit has been reached. If no limit was specified, then the EOR would mean that all the contents of the relevant data-bins have been served.

The following reason codes are currently defined (Table D.2):

Table D.D2 — Defined reason codes

R	Reason	Explanation
R=1	Image Done	The server has transferred all available image information (not just information relevant to the requested view-window) to the client.
R=2	Window Done	The server has transferred all available information that is relevant to the requested view-window.
R=3	Window Change	The server is terminating its response in order to service a new request which does not specify Wait=yes.
R=4	Byte Limit Reached	The server is terminating its response because the byte limit specified in a byte limit specified in a max length request field has been reached.
R=5	Quality Limit Reached	The server is terminating its response because the quality limit specified in a quality request field has been reached.
R=6	Session Limit Reached	The server is terminating its response because some limit on the session resources, e.g. a time limit, has been reached. No further request should be issued using a channel ID assigned in that session.
R=7	Response Limit Reached	The server is terminating its response because some limit, e.g., a time limit, has been reached. If the request is issued in a session, further requests can still be issued using a channel ID assigned in that session.
R=0xFF	Non Specified Reason	

Annex E
(normative)

Uploading images to the server

E.1 Introduction

It is anticipated that images will be placed on a server in a variety of ways outside of the scope of this standard. The purpose of this annex is to describe a mechanism that allows portions of an image to be uploaded to a server.

E.2 Upload request

E.2.1 Request structure

An upload request consists of one or more request fields defined in Annex C, and a request body.

E.2.2 Upload request fields

The request fields for an upload shall contain an upload request field. The Target ID, target and sub-target request field may also be used. For an upload of a complete image media type, the Frame Size, region Size and Offset fields are used to indicate the position of the uploaded portion within the entire image. For uploads of JPT-stream and JPP-stream, the number of the data-bin (and hence the tile or precinct number) along with the main header indicate the location of the coded data and the view-window request fields are unnecessary.

E.2.3 Upload request body

E.2.3.1 General

The body of an upload request consists of one of the supported image types: JPP-stream, JPT-stream, or a complete image media type. The body contains the data that the client is requesting to have handled by the server. This International Standard does not support uploading raw image data.

E.2.3.2 JPT-stream

The body of the request contains all data-bins the client wants the server to replace (header data-bins, metadata-bins, and tile data-bins). If the client does not upload a main header data-bin the tile data-bins shall be encoded in a compatible manner with the current main header.

E.2.3.3 JPP-stream

The body of the request contains all data-bins the client wants the server to replace (header data-bins, tile-header data-bins, metadata-bins, and precinct data-bins). If the client does not upload a main header data-bin or tile header data-bin the precincts shall be encoded in a compatible manner with the current main and tile-headers.

E.2.3.4 Complete image upload

The body of the request contains a complete image media type representing those samples the client wishes to modify.

In the case of a complete image upload, the request may include Frame Size, Region Size and Offset request fields. The Frame Size field shall be the size of the reference grid of the image. In the case of a complete image upload, the compression need not be done in a compatible way with the logical target on the server. If the size of the uploaded image exceeds the extent in the Region Size field, the server should limit modifications to the extent specified in the Region Size field.

E.3 Server response

E.3.1 General

The server shall respond to an upload request with a status code and reason phrase from Annex D. Useful return codes and reason phrases for image upload include:

E.3.2 201 (Created)

The server should use this status code if upon receiving an upload request, a new resource has been defined on the server. The server shall have completed the creation before returning this request. If there will be a delay, the server should return 202 (Accepted) instead of 201 (Created).

The server should include a header with the response with a new target ID field for the updated resource.

No body need be returned.

E.3.3 202 (Accepted)

The server should use this status code if an upload creates a new resource but the server is not yet prepared to serve it. The server may also use this status code for an update of a current resource.

E.3.4 400 (Bad Request)

Servers should issue this status code if the request is incorrectly formatted, or if the query contains request fields that are incompatible with uploading or contains an unrecognized field in the query string.

E.3.5 404 (Not Found)

This status code should be issued if the server cannot reconcile the requested resource with an issued target ID.

E.3.6 415 (Unsupported media type)

This status code may be issued to indicate that while uploads are supported, uploads of the particular type (e.g. complete image, JPT-stream, or JPP-stream) included with the request are not supported.

E.3.7 501 (Not implemented)

This status might be used if the server does not support upload or does not support a particular option with upload.

E.4 Merging data on the server

E.4.1.1 Updating the image

After receiving the uploaded data, the server may create a new version of the logical target and provide the new version to clients accessing a new or the old URL. However, the server shall not use the old target-id request field to provide access to any merged or updated data.

If the client includes a Target ID request field in the upload request and that target ID does not match the server's current target ID for the resource, the server should not update the image. This mismatch may indicate the client has edited a previous version of the image that has already been modified. Servers may refuse to accept uploads which do not contain a Target ID field. This is one way to prevent multiple simultaneous edits of a target by different clients. Servers providing editing capabilities may take care of such issues as target locking by some other means.

A JPIP client may upload part of a new image by specifying a target ID of 0, or using a new URL, or target which the server doesn't have. The server should issue a target ID for the upload. A client may continue to upload additional portions of the new image by using the target ID returned by the server with the previous upload.

E.4.1.2 JPT-stream

A server accepting tile data-bin data shall first remove all the old tile data-bin data for those tiles being uploaded, and then include the new tile data-bin data into the codestream. An update cannot be made that results in a change to the number or dimension or location of tiles - the structure of the image cannot be changed by an upload. In particular, a server should not accept tile data-bin uploads for a codestream containing a PPM marker segment in the main header, unless the client provides a new main header with the upload. Any PLM or TLM marker segments shall be deleted or updated. A JPT-stream main header data-bin shall be uploaded for new images.

How the codestream tile-parts from a tile data-bin are formed is not specified. The client need not necessarily provide all tile-parts of a tile, nor need the last tile-part be completed. The server shall update the main header and any portions of the file format affected (for example length of the codestream box).

[Editor's Note: The number or size of tiles, and thus the size of the image, shall not be changed by an update/upload. Should not change the meaning of whatever is not replaced/updated.

E.4.1.3 JPP-stream

A server accepting precinct data-bin messages shall first remove the corresponding old precinct data-bins for those precincts being uploaded, and then include the new precinct data-bin data. A change cannot be made to a header that results in a change to the number of precincts, or the meaning of the precinct identifier, or the location or size of each precinct within its tile-component-resolution. JPP-stream tile header data-bins and main header data-bins shall be uploaded for new images.

[Editor's note: Should not change the meaning of whatever is not replaced/updated. Typically the header is not able to be updated for most elements. A comment could be updated. Others?

How the precinct packets from a precinct data-bin are formed is not specified. The client need not necessarily provide all packets of a precinct, or even complete the last provided packet.

E.4.1.4 JPP-stream and JPT-stream metadata-bins

Metadata-bin boxes can be uploaded, replacing existing metadata-bin boxes. Since the server has control of the division of allocating metadata into metadata-bins, the client shall follow the server's metadata-bin structure. The client shall not change placeholders in a metadata-bin, except to completely remove a placeholder. When uploading an entire metadata-bin, clients can add new metadata by appending to the end of the old metadata-bin, or by inserting new metadata between boxes in the old metadata-bin. The server shall

manage the placeholders and the metadata-bin structure. This includes updating all placeholders pointing to any decedent metadata boxes that have been changed or affected by the change. The server shall delete any metadata boxes that were pointed to by a placeholder that the client has removed. The server may re-structure the metadata after an upload is accepted, but before the new resource is created. If unused sections are left in the file after uploading, Free boxes shall be used to fill those sections.

E.4.1.5 Complete image upload

In the case of an acceptable complete image upload, the server should uncompress (if required) the uploaded sub-image, uncompress some portion of the full image on the server, replace those pixels in the (uncompressed) spatial domain and recompress all tiles or precincts affected by the update operation**.**

NOTE This technique requires more computation on the server; however, it removes the possibility that the client will use compressed image data in an incompatible way (e.g. the wrong number of levels of wavelet transform).

Annex F
(normative)

Using JPIP over HTTP

F.1 Introduction

This annex defines the method to use JPIP with the HTTP for both requests and responses. The JPIP request parameters from Annex C are encapsulated in legal HTTP request structures. The server responses (including status codes, headers, messages, and response codes) from Annex D are encapsulated in legal HTTP responses.

Note the text and examples in this annex describe use of JPIP over HTTP. It is expected that the same binding can be used for secure HTTP.

F.2 Requests

F.2.1 Requests Introduction

Annex C defines request fields. When transported via HTTP, the JPIP request can appear as a query string for an HTTP GET request or as the body of an HTTP POST request. Because some HTTP systems limit the length of the query string provided in a GET request, the POST request is preferred for long JPIP requests.

Note that HTTP defines a `Request` as:

```
Request = Request-Line              ; HTTP Section 5.1
          0*(( general-header       ; HTTP Section 4.5
            | request-header        ; HTTP Section 5.3
            | entity-header ) CRLF) ; HTTP Section 7.1
           CRLF
           [ message-body ]         ; HTTP Section 4.3
```

Note that HTTP defines `Request-Line` and `Request-URI` as:

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
Request-URI = "*" | absoluteURI | abs_path | authority
```

Note that RFC2396 defines:

```
absoluteURI   = scheme ":" ( hier_part | opaque_part )

hier_part     = ( net_path | abs_path ) [ "?" query ]

abs_path      = "/"  path_segments
```

F.2.2 GET requests

A JPIP request can be provided to a server as a HTTP request. For a “GET” request the HTTP request is restricted in the following manner:

— The “Method” shall be “GET”

— The “query” shall be zero or more `jpip-request-field` separated by ‘&’.

An example of a JPIP request encapsulated in an HTTP GET request is:

```
GET /images/kids.jp2?rsiz=640,480&roff=320,240&fsiz=1280,1024 HTTP/1.1
Host: get.jpeg.org
CRLF
```

An equivalent example using an `absoluteURI` instead of an `abs_path` is:

```
GET http://get.jpeg.org/images/kids.jp2?rsiz=640,480&roff=320,240
&fsiz=1280,1024 HTTP/1.1
CRLF
```

Note this standard imposes no restriction on the scheme component of the absoluteURI.

F.2.3 POST requests

A JPIP request can be provided to a server encapsulated in an HTTP POST request. For a “POST” request the HTTP request is restricted in the following manner:

— The “Method” shall be “POST”.

— The “entity-body” shall be zero or more jpip-request-field separated by ‘&’.

— The “Content-type:” header line should be included as an “entity-header” and contain the value “application/x-www-form-urlencoded”.

An example of a JPIP request encapsulated in an HTTP POST request is:

```
POST /cgi-bin/j2k_server.cgi HTTP/1.1
Host: post.jpeg.org
Content-type: application/x-www-form-urlencoded
Content-length: 62
CRLF
target=/images/kids.jp2&rsiz=640,480&roff=320,240&fsiz=1280,1024
```

F.2.4 Upload requests

An upload request is a legal HTTP request restricted as follows:

— The “Method” shall be “POST”.

— The URL shall contain the upload query-field.

— The Content-type shall be the image type of the body: image/jpt-stream, image/jpp-stream, or a complete image media type.

F.3 Session Establishment

A stateful HTTP session is established by using the New Channel request field with a value of “http”, i.e. “cnew=http” as part of a request. This request is typically delivered by HTTP. The request may contain a view-window request that becomes the first request in the new channel. The response to this request is returned on the same connection as the request was made.

A client may open an HTTP connection and issue a request which includes the HTTP header “Connection: keep-alive.” This is useful for efficient sessions, but it is neither necessary nor sufficient to have a session. A

single HTTP connection may be used for traffic for different targets, different channels, or even non-JPIP traffic, e.g. requests for html files. A JPIP request that is part of a session may arrive on HTTP connections other than the HTTP connection used to request and issue the new channel, although this is discouraged.

F.4 Responses

F.4.1 Introduction

Each component of a response from Annex D may be encapsulated as a portion of a legal HTTP response.

Note HTTP responses are defined in section 6 of RFC2616 as:

```
Response = Status-Line                    ; HTTP Section 6.1
           0*(( general-header            ; HTTP Section 4.5
             | response-header            ; HTTP Section 6.2
             | entity-header ) CRLF)      ; HTTP Section 7.1
            CRLF
            [ message-body ]              ; HTTP Section 7.2
```

JPIP responses transported over HTTP shall be legal HTTP responses, with further limitations on some of the parts of the HTTP response as described in the following sections.

F.4.2 Status code and Reason phrase

All of the status codes listed in D.1.2 may be used directly as HTTP status codes. In addition a server providing JPIP over HTTP may use any HTTP status code deemed useful, e.g. 402.

All values for Reason-Phrase provided in D.1.2 may be used directly as HTTP Reason-Phrase. The Reason-Phrase shall be appropriate for the status code. A server providing JPIP over HTTP may use any HTTP Reason-Phrase deemed useful, e.g. Payment required.

F.4.3 Header info

F.4.3.1 JPIP Headers

The header lines from D.2 shall be included as the “entity-header” in the HTTP response without modification.

F.4.3.2 Use of HTTP Accept header

A server providing JPIP over HTTP may use an HTTP “Accept:” header line found in a request to determine the type of JPIP response. If the request contains a “type=” query parameter, the return type shall be one of the types listed in the type parameter. If the request contains both a “type=” query parameter and an “Accept:” header line, the server may use the priorities specified in the “Accept:” line to select between the types specified in the “type=” query parameter. If no “type=” query parameter is present in the request, the server may select a return type supported by the underlying JPIP server from the list of types in the “Accept:”.

F.4.3.3 Use of Cache-control header

Note that the caches in HTTP proxies are different from the caches and cache models in JPIP.

Any JPIP request with a New Channel request field is part of a session and such responses cannot generally be cached by HTTP proxy servers. Similarly, any response which includes a New Channel response header is also part of a session. In both cases, the server’s response should include an HTTP “Cache-Control:” header line with the value “no-cache”.

F.4.3.4 Use of Content-type header

A server providing JPIP over http should include a “Content-type:” header line, indicating the type of data in the body, most commonly this is image/jpp-stream or image/jpt-stream.

F.4.3.5 Use of Redirect header

The http Redirect header may be useful to inform a client that the resources has moved or should be accessed from a different host.

Note that the JPIP response defines a way to do a redirect as well. The JPIP response should be preferred within a session.

F.4.4 Body

The messages from Annex D shall be included as the body of the HTTP response. Note that a HTTP response shall have a mechanism to determine the length of the response. If the server does not plan to interrupt a response, it may provide this information with a “Content-Length” HTTP header line. The preferred method of providing the length is to use the HTTP header line “Transfer-Encoding: chunked” and then to provide the body in chunks of a size determined by the server and specified before each chunk. Indicating the end of a response by closing the HTTP connection is discouraged.

F.5 Additional HTTP features

F.5.1 Use of HTTP HEAD method

JPIP clients and servers are not required to use or support the HTTP HEAD method. A server choosing to implement the HEAD method shall do so as specified in Section 9.4 of RFC2616. In particular, “The HEAD method is identical to GET except that the server shall not return a message-body in the response.”

Clients may find it useful to issue head requests as a means to determine if the server will modify any of the request parameters as specified in Annex D. Clients should not issue a HTTP HEAD request with cache model query fields as this may cause the server to update its cache model.

Note a client wishing to update the server cache model without receiving a response may use the length request field.

Servers may refuse any or all HEAD requests. Unlike typical HTTP HEAD requests that require relatively little effort for a server to fulfil, some JPIP server implementations might have to obtain data from several locations in a logical target, compute the nature of the response, and then discard the body of the response in order to respond to a HEAD request.

F.5.2 Use of HTTP Options method

JPIP clients and servers are not required to use or support the HTTP OPTIONS method.

F.5.3 Etag usage

Note that HTTP defines the entity tag (ETag) mechanism that is similar to the JPIP Target ID request field in that it is used to denote changes in a resource. If both an entity tag and a target ID are associated with a resource, it is recommended that the ETag defined by HTTP be changed whenever the target-id is changed.

F.5.4 Use of chunked transfer encoding

Because responses containing compressed data can be very large and thus take a long time to transmit it is important to be able to stop in the midst of transmission. Unless “Transfer-Encoding: chunked” is specified,

HTTP requests shall specify the full length of the body in a “Content-Length:” header or indicate the end of data by closing the connection. Neither of these is desirable in an interactive protocol, since it may be necessary to stop the current response and send more data on the same connection for a new response.

Note section 19.4.6 of the HTTP RFC provides an algorithm for removing the chunked transfer encoding.

Note chunked transfer encoding may be useful with JPIP when delivered over protocols other than HTTP.

Annex G
(normative)

Using JPIP with HTTP requests and TCP returns

G.1 Introduction

The JPIP protocol itself is neutral with respect to underlying transport mechanisms for the client requests and server responses, except in regard to channel requests represented by the New Channel request field and the New Channel ("JPIP-cnew") response header, where transport specific details shall be communicated. This International Standard defines two specific transports, which are identified by the strings “http” and “http-tcp” in the value string associated with New Channel requests. This annex provides details of the second transport, which shall be identified in this text as HTTP-TCP. The first transport is identified in this text as HTTP and is described in Annex F.

The HTTP-TCP transport uses exactly the same mechanisms as the HTTP transport to send client requests to the server and receive the server’s response headers and status codes. However, the server’s response data (not the response headers) is delivered over an auxiliary TCP connection. The information transported on this auxiliary TCP connection is identical to that which would have been transported as the entity body of a pure HTTP response, except that it is framed into chunks, each of which has a chunk sequence number.

The client explicitly acknowledges the arrival of each chunk by sending its sequence number back to the server on the auxiliary TCP connection’s return path. One of the principle benefits of the HTTP-TCP transport is that the server receives incremental notification of the arrival of its response data chunks via this client acknowledgement mechanism. This allows the server to manage the flow of data in such a way as to maintain responsiveness and network efficiency.

G.2 Client Requests

Requests are delivered on the primary channel exactly as HTTP requests. They have exactly the same form as requests issued over a channel that uses the HTTP transport described in Annex F. In particular, HTTP GET and POST requests may both be used.

G.3 Session Establishment

G.3.1 Channel establishment

A new channel may be established to a JPIP server by issuing a request that includes the New Channel (“cnew”) request field. As an example, such a request might be issued using HTTP, although it might also be issued to a JPIP-specific server using any suitable transport mechanism. If the server’s response (through the New Channel response header) indicates that a new channel has been created to work with the HTTP-TCP transport, the client shall establish the auxiliary TCP connection using the auxiliary port number returned via the New Channel response header. Furthermore, the request which included the New Channel request field is then treated as though it had been issued within the newly created HTTP-TCP transported channel, meaning that the response data generated by that request shall be returned via the auxiliary TCP connection, as soon as it has been connected.

To establish the auxiliary TCP connection, the client issues a TCP connection request to the server host identified via the New Channel response header, on the port identified by the New Channel response header. The client then immediately sends a single line of ASCII text, consisting of the new channel-id string, followed

by two consecutive CR-LF pairs. This is the only text-oriented communication delivered over the auxiliary TCP connection.

The client then waits to receive the server's response data over the auxiliary TCP connection. This response data cannot be empty, since every request issued within an HTTP-TCP transported channel shall have a response data stream that consists of at least the EOR (End of Response) message. See G.4 for more on this.

G.3.2 Server framing of response data

All response data sent by the server via the auxiliary TCP connection shall be framed into chunks. Each chunk consists of an 8-byte chunk header, followed by the chunk body that holds the server's return data, as shown in Figure G.1. The first 2-byte word of the chunk header holds an unsigned big-endian integer representing the total length of the chunk, including the length word itself. The contents of the remaining 6 bytes of the chunk header are not defined by this International Standard. They may be used for additional server-specific signalling. The client will return the entire 8-byte chunk header in its chunk acknowledgement messages.

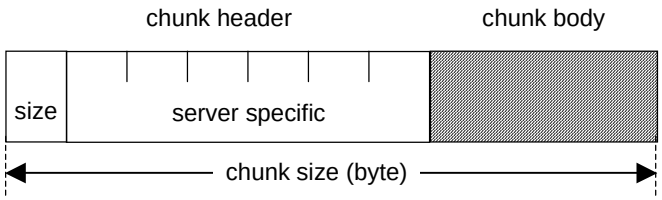

Figure G.G1 — Response data structure on http-tcp connection

G.3.3 Client acknowledgement of server response chunks

Upon receipt of a server response data chunk on the auxiliary TCP connection, the client shall send the 8-byte chunk header back to the server as an unframed stream of data, using the TCP connection's return path. Each received chunk is to be acknowledged in sequence.

G.4 Server Responses

In response to each client request, the server sends an HTTP reply paragraph back to the client over the primary channel. The reply paragraph contains the status code, reason phrase and all relevant JPIP response headers and any appropriate HTTP response headers. However, no response data is returned via the primary channel. For this reason, there shall be no HTTP entity body in an HTTP-TCP response. Neither shall the "Content-length:" or the "Transfer-encoding:" HTTP response headers be used.

The response data itself is delivered over the auxiliary TCP channel, framed into chunks in the manner described in G.3.2. Since the HTTP-TCP transport may be used only with sessions and hence only with JPP-stream and JPT-stream image return types, the response data invariably consists of a sequence of JPP-stream or JPT-stream messages.

The response data resulting from each request shall consist of a whole number of chunks, meaning that no chunk may contain response data generated in response to two different requests.

The response to each and every request shall be terminated with an EOR message, even if the response data would otherwise have been empty. The EOR message is considered as part of the response data and is framed into chunks along with the actual JPP-stream and JPT-stream messages.

This means that every request issued on an HTTP-TCP transported JPIP channel results in the generation of at least one non-empty response chunk from the server and that the last chunk generated in response to each request terminates with the EOR message.

Note that there is no actual requirement for HTTP-TCP transported response chunks to be aligned on message boundaries.

G.5 TCP and length request field (Informative)

There may be little or no reason for using the length field with a TCP return channel, where the server is able to carefully regulate the flow of response data to the client so as to maintain responsiveness. With a HTTP return channel, the server does not receive continuous feedback from the client and may easily push a great deal of data into the pipe, which shall be fully received before any data for a new window can be processed. To maintain responsiveness, clients should use this Bytes field to regulate the flow of traffic and hence maintain responsiveness. Clients will generally need to implement their own flow control algorithms to adjust the request length to changing network conditions.

Annex H
(normative)

Using JPIP with alternate transports

H.1 Introduction (Informative)

The purpose of this annex is to provide guidelines on the deployment of JPIP over unreliable transports. This international standard does not define any specific transport protocol other than the “http” transport described in Annex F and the “http-tcp” transport described in Annex G. Rather, this annex provides a generic approach which may be applied to a wide variety of transports.

In developing the general approach, it is helpful to divide aspects of the communication into two logical transport connections, termed the “request connection” and the “data connection”. Each logical connection is understood to provide both a forward communication path and a reverse communication path. The roles played by these paths are as follows:

— The forward request connection path is used to deliver JPIP requests from the client to the server.

— The reverse request connection path is used by the server to acknowledge the receipt of requests and return response headers to the client.

— The forward data connection path is used to deliver JPIP stream messages from the server to the client.

— The reverse data connection path is used by the client to acknowledge receipt of JPIP stream messages from the server.

The reader will observe that these roles are consistent with those served by the forward and reverse communication paths of the two TCP channels used by the “http-tcp” transport described in Annex G. Indeed the material in this annex may be interpreted as an extension of the “http-tcp” transport to unreliable transports. Note, however, that although this annex is described in terms of two different logical connections, there is no reason why the communication cannot be carried over a single transport connection.

Finally, it is assumed that each logical connection provides one of the following two types of services:

1. A reliable stream-oriented service, such as that offered by TCP.

2. An unreliable packet-oriented service, such as that offered by UDP. In this case, packets may arrive out of order or not at all, and acknowledgement handshaking shall be implemented explicitly so as to determine whether or not a packet has arrived successfully.

Two scenarios are considered in this annex. In the first case, the request connection path is assumed to offer a reliable stream-oriented service, but the data connection path is unreliable. In the second case, both the request and data connection paths are unreliable. It is helpful to treat these two scenarios in order.

H.2 Reliable Requests with Unreliable Data (Informative)

In this clause, the request connection is reliable, meaning that requests arrive at the server in order without loss, and server responses are received by the client in order and again without loss. In this case, the request fields and response headers may be communicated exactly as in the “http-tcp” protocol, and indeed HTTP is recommended for the transport of requests and response headers. A transport protocol of this flavour might, for example, be named “http-udp”, but such specifics are beyond the scope of this annex.

The JPIP stream messages, including the EOR message, shall be partitioned into packets and delivered over the unreliable data connection (e.g., over UDP). The client shall acknowledge receipt of each such packet by sending the packet's header back to the server. This enables the server to estimate network conditions, and determine whether or not packet retransmission is justified. In the event that the client's view window has changed, the server might decide not to retransmit an unacknowledged packet.

The following general guidelines should be observed when constructing transport protocols of this flavour:

1. Each request should include a Request ID (qid) request field (see Annex C.3.4).

2. For each request, there shall be a corresponding EOR message, even if no JPIP stream messages are sent in response to the request. This requirement also applies in the case of the "http-tcp" transport.

3. Each data connection packet constructed by the server shall consist of a whole number of JPIP stream messages and/or EOR messages. Moreover, the first JPIP stream message in each packet shall contain a complete header, not relying upon repetition of the codestream identifier or class code components of a previous message.

4. All JPIP stream messages (not necessarily EOR messages) found in a data connection packet shall belong to the response from a single request, and the corresponding Request ID (qid) shall be encoded in the packet's header.

5. EOR messages may be found either at the end of a packet bearing the same Request ID value as the request whose response is being ended, or in a block of one or more consecutive EOR messages found at the start of the first packet following the last packet bearing that Request ID. This policy allows EOR messages corresponding to one or more consecutive empty responses (e.g. due to pre-empted requests) to be bundled into the first packet of the subsequent non-empty response.

6. In addition to the Request ID value, each packet header should include a packet sequence number. The packet sequence counter is set to 0 for the first packet associated with any particular Request ID value. Subsequent packets with the same Request ID value have consecutive sequence numbers. This policy allows a client to identify any EOR messages which might not have been received due to packet loss. It is important that a client be able to associate requests with response data, so as to synchronize the effects of cache model manipulation statements at the server with the state of their own cache.

7. Clients shall acknowledge the receipt of each packet by sending acknowledgement messages to the server on the return data connection path. Each acknowledgement message should contain a replica of the received packet's header, but might conceivably contain additional information. The client may, at its discretion, aggregate acknowledgement messages to several packets when constructing acknowledgement packets. However, excessive aggregation may affect the reliability with which servers can estimate network statistics.

8. The server is not obliged to retransmit any unacknowledged packet and clients should not expect retransmission of missing packets. An intelligent server might, for example, choose to retransmit unacknowledged packets depending upon their relevance to the current view window.

H.3 Unreliable Requests with Unreliable Data (Informative)

This clause is concerned with transports in with both the request and data connections are unreliable. Guidelines for the data connection are exactly as described in Annex H.2 for the case in which requests are delivered reliably. In this case, however, it is possible that one or more requests might be lost or arrive out of order at the server. JPIP is well adapted to handling this situation, since servers have the freedom to pre-empt previous requests when a new request arrives.

The following general guidelines should be observed when handling unreliable requests, in addition to those listed in Annex H.2.

1. Each request packet should include a header, identifying the value of the Request ID.

2. Each request packet should also include a sequence number, carrying sufficient information to determine whether or not all packets associated with a request have been received.

3. In many cases, servers can simply ignore missing request packets when a new request arrives. To do this, the server has only to send EOR messages on the data connection, indicating that the missing request was pre-empted immediately. There is no fundamental need for acknowledgement messages to be sent in response to request packets or for any response headers to be sent in response to requests which are being immediately pre-empted because some or all of the request packets were lost.

4. For each request which arrives in full at the server, the server should send one or more response packets which identify the Request ID and include any response headers. This is true even if the request arrives after the response was issued to a subsequent request (e.g., because some packets of a request were unduly delayed). This provides the client with a mechanism for determining whether or not an important request was received by the server.

5. Certain types of requests shall be processed by the server to avoid loss of synchronization with the client. The most important of these are requests which include subtractive cache model manipulation fields. To enable the server to detect such requests, without having to fully serialize the request stream, request packet headers should include the following two fields:

    a. A flag indicating whether or not the packet belongs to a request which shall be processed before processing subsequent requests.

    b. The Request ID associated with the most recent request for which the flag mentioned in 5a was set.

    If the server does not receive one or more packets of a request which shall be processed, it shall idle until the packets are retransmitted by the client.

H.4 Request and Response Syntax (Informative)

The request and response syntax described in Annex C and Annex D should be followed when designing new transports for the JPIP protocol. However, it is permissible to develop equivalent binary representations of various request fields and response headers.

H.5 Session Establishment (Informative)

The New Channel request field and corresponding response header may be used to create channels associated with transport protocols other than the “http” and “http-tcp” transports described normatively in this international standard. For this purpose, new transport protocol names may be registered with the registration board defined in Annex H. The procedure for creating channels for new transports should follow the same general conventions outlined for “http-tcp”. In particular, the response headers for the request which creates the new channel should be returned on the transport which was used to create the channel, while response data should be delivered using the new channel’s transport.

Annex I
(normative)

Indexing JPEG 2000 files for JPIP

I.1 Introduction (informative)

ITU–T Rec. T.800 | ISO/IEC 15444–1:2002 and other standards define a family of JPEG 2000 file formats. The family utilises a common syntax, whose basic element is the container called a box. This annex defines new file format boxes containing indexing information, the inclusion of which in JPEG 2000 family files may facilitate the deployment of those files in a JPIP system, by enabling file readers to locate within the files the elements that are required to construct images incrementally.

In particular, these boxes may be useful

— to a server-side implementation of the JPIP protocol

— to a client, accessing an image remotely, using a simpler protocol but one that allows access to specified byte-ranges of the file

This annex defines index boxes corresponding to both file-level information and codestream information. The boxes may be categorised as follows:

— The Codestream Index (cidx) superbox indexes codestream information. It contains a Codestream Finder (cptr) box pointing to the indexed codestream, a Manifest (manf) box summarising the rest of the contents, and index table boxes, which are the Main Header Index Table (mhix) box, the Tile-part Index Table (tpix) superbox, the Tile Header Index Table (thix) superbox, the Precinct Packet Index Table (ppix) superbox and the Packet Header Index Table (phix) superbox. The index table boxes correspond to the different types of codestream data represented by data-bin classes in the JPP-stream and the JPT-stream defined in Annex A. The index table boxes which are superboxes contain Fragment Array Index (faix) boxes listing the actual codestream elements. The Precinct Packet and Packet Header index table superboxes also each contain a Manifest box.

— The File Index (fidx) superbox indexes file-level information. Unless it indexes the top level of the file, in which case it is called a root File Index box, it contains a File Finder (fptr) box pointing to the indexed superbox. It may contain Proxy (prxy) boxes representing the contents of the indexed file or superbox. The File Index box corresponds to the metadata-bin class of the JPP-stream and JPT-stream.

— The Index Finder (iptr) box points to a root File Index, enabling its location to be discovered.

The figure below (Figure I.1) illustrates an example JPEG 2000 file containing JPIP index boxes:

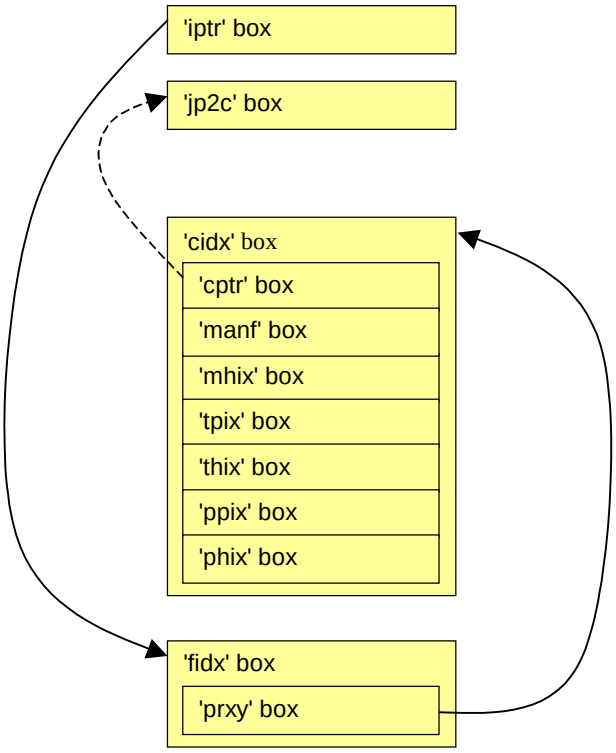

Figure I.I1 — Part of an example JPEG 2000 file containing JPIP index boxes

I.2 Identifying the use of JPIP index boxes in the JPEG 2000 file format compatibility list

Files that contain one or more of the index boxes defined in this International Standard may contain a CL^{i} field in the File Type box (as defined in Annex I of ITU–T Rec. T.800 | ISO/IEC 15444–1) with the value ‘jpip’ (0x6a70 6970).

I.3 Key to graphical descriptions (informative)

The description of each box is followed by a figure that shows the order and relationship of the parameters in the box. Figure I.2 shows an example of this type of figure. A rectangle is used to indicate the parameters in the box. The width of the rectangle is proportional to the number of bytes in the parameter. A shaded rectangle (diagonal stripes) indicates that the parameter is of varying size. Two parameters with superscripts and a gray area between indicate a run of several of these parameters. A sequence of two groups of multiple parameters with superscripts separated by a gray area indicates a run of that group of parameters (one set of each parameter in the group, followed by the next set of each parameter in the group). Optional parameters or boxes will be shown with a dashed rectangle.

The figure is followed by a list that describes the meaning of each parameter in the box. If parameters are repeated, the length and nature of the run of parameters is defined. As an example, in Figure I.2, parameters C, D, E and F are 8, 16, 32 bit and variable length respectively. The notation G^{0} and $G^{N–1}$ implies that there are N different parameters, G^{i}, in a row. The group of parameters H^{0} and $H^{M–1}$, and J^{0} and $J^{M–1}$ specify that the box will contain H^{0}, followed by J^{0}, followed by H^{1} and J^{1}, continuing to H^{M-1} and J^{M-1} (M instances of each parameter in total). Also, the field E is optional and may not be found in this box.

In addition, in a figure describing the contents of a superbox, an ellipsis (…) will be used to indicate that contents of the file between two boxes is not specifically defined. Any box (or sequence of boxes), unless otherwise specified by the definition of that box, may be found in place of the ellipsis.

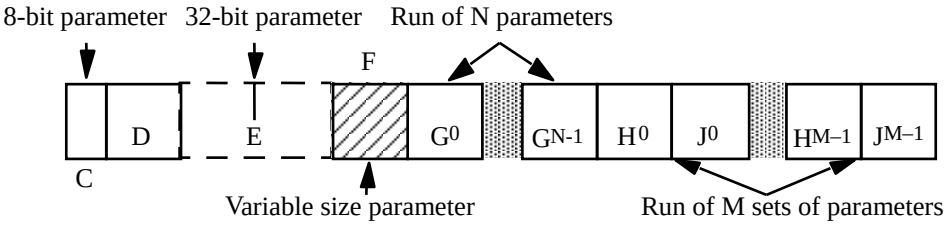

Figure I.I2 — Example of the box description figures

For example, the superbox shown in Figure I.3 must contain an AA box and a BB box, and the BB box must follow the AA box. However, there may be other boxes found between boxes AA and BB. Dealing with unknown boxes is discussed in Annex I.8 of ITU–T Rec. T.800 | ISO/IEC 15444–1:2002.

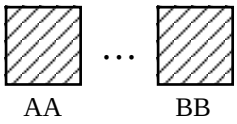

Figure I.I3 — Example of the superbox description figures

I.4 Defined boxes

I.4.1 General

Table I.1 lists all boxes defined as part of this International Standard. For the placement of and restrictions on each box, see the relevant section defining that box.

[Editor's note: should this table be informative, to avoid double definitions?]

Table I.I1 — Defined boxes

Box name	Type	Superbox	Comments
Codestream index box (I.3.2)	'cidx' (0x6369 6478)	Yes	This box contains indexing information about a JPEG 2000 codestream.
Codestream Finder box (I.3.2.2)	'cptr' (0x6370 7472)	No	This box points to a JPEG 2000 codestream.
Header Index Table box (I.3.2.4.3)	'mhix' (0x6D68 6978)	No	This box specifies an index of the marker segments in the main header of a codestream or the tile-part headers of a tile.
Tile-part Index Table box (I.3.2.4.4)	'tpix' (0x7470 6978)	Yes	This box specifies the locations and lengths of each tile-part in the codestream.
Tile Header Index Table box (I.3.2.4.5)	'thix' (0x7468 6978)	Yes	This box specifies the locations and lengths of each part of the codestream necessary to construct tile headers for each tile for the correct decoding of precinct packet data.
Precinct Packet Index Table box (I.3.2.4.6)	'ppix' (0x7070 6978)	Yes	This box specifies the locations and lengths of packets within the codestream.

Table I.I1 — Defined boxes

Box name	Type	Superbox	Comments
Packet Header Index Table box (I.3.2.4.7)	‘phix’ (0x7068 6978)	Yes	This box specifies the locations and lengths of packet headers within the codestream.
Manifest box (I.3.2.3)	‘manf’ (0x6D61 6E66)	No	This box summarizes the boxes that immediately and contiguously follow it, within its containing box or file at the same level as the Manifest box.
Fragment Array Index box (I.3.2.4.2)	‘faix’ (0x6661 6978)	No	This box specifies the locations and lengths of the elements of a codestream.
File Index box (I.3.3)	‘fidx’ (0x6669 6478)	Yes	This box can be used to find other indexes and arbitrary data within the file.
File Finder box (I.3.3.2)	‘fptr’ (0x6670 7472)	No	This box points to an indexed box.
Proxy box (I.3.3.3)	‘prxy’ (0x7072 7879)	No	This box represents in a File Index box a box elsewhere in the file.
Index Finder box (I.3.4)	‘iptr’ (0x6970 7472)	No	This box points to the root File Index box of a file.

I.4.2 Codestream Index box (superbox)

I.4.2.1 General

The Codestream Index box contains indexing information about a JPEG 2000 codestream. The type of a Codestream Index box shall be ‘cidx’ (0x6369 6478). The contents of a Codestream Index box shall be as follows (Figure I.2):

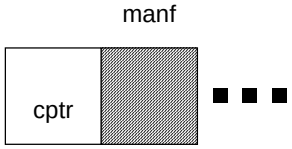

Figure I.I2 — Organization of the contents of a Codestream Index box

cptr: Codestream Finder box. This box points to the codestream indexed by the Codestream Index box. Its structure is specified in I.3.2.2.

manf: Manifest box. This box summarises the index tables following it inside the Codestream Index box. Its structure is specified in I.3.2.3.

I.4.2.2 Codestream Finder box

The Codestream Finder box points to a JPEG 2000 codestream. The type of a Codestream Finder box shall be ‘cptr’ (0x6370 7472). The contents of a Codestream Finder box shall be as follows (Figure I.3):

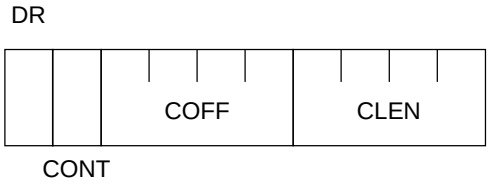

Figure I.I3 — Organization of the contents of a Codestream Finder box

DR: Data Reference. This field specifies the location of the codestream, or of the Fragment Table box standing for it. If 0, the codestream or its Fragment Table box exists in the current file. Otherwise, the quantity identifies an entry in the Data Reference box in the current file. In this case, the Data Reference entry identified by DR indicates the resource that contains the codestream or Fragment Table box. This field is stored as a 2-byte big endian unsigned integer.

CONT: Container Type. If the value of this field is 0, the entire codestream appears as a contiguous range of bytes within its file or resource. In this case, the offset and length values given here refer to the codestream itself. Note that the codestream may well be within a Contiguous Codestream box, but the offset and length values refer to the codestream itself, starting at the SOC marker and ending immediately after the EOC marker. If CONT is 1, the codestream is fragmented and the location and length values refer to the Fragment List box (including its box header) describing the locations and lengths of each of the fragments that represent the codestream. Note that all subsequent locations and lengths are expressed relative to the start of the codestream, as it would appear after reconstituting all of the fragments identified in the Fragment List box. All other values of CONT are reserved. This field is stored as a 2-byte big endian unsigned integer.

COFF: Codestream Offset. This field specifies the location of the codestream or Fragment List box, as appropriate, relative to the start of the file or resource identified by DR. This field is stored as an 8-byte big endian unsigned integer.

CLEN: Codestream Length. This field specifies the length of the codestream or Fragment List box, as appropriate. This field is stored as an 8-byte big endian unsigned integer.

I.4.2.3 Manifest box

The Manifest box summarises the boxes that immediately and contiguously follow it, within its containing box or file at the same level as the Manifest box.

NOTE The Manifest box may be used to facilitate random access into these following boxes, such as the index boxes following it inside a Codestream Index box.

The type of a Manifest box shall be ‘manf’ (0x6D61 6E66). The contents of the Manifest box shall be as follows (Figure I.4):

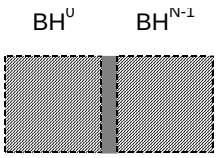

Figure I.I4 — Organization of the contents of a Manifest box

BH^{i}: Box Header. This field contains the complete box header of the i^{th} box immediately following the Manifest box. The length of this field is 16 bytes if the value of the LBox field contained within that box header is 1, or 8 bytes otherwise.

The number of boxes, *N*, whose headers are contained within the Manifest box, is determined by the length of the Manifest box. When used inside a Precinct Packet Index Table box or a Packet Header Index Table box, *N* is the number of codestream components.

Inside a Codestream Index box, a Precinct Packet Index Table box or a Packet Header Index Table box, a Manifest box shall include all of the boxes that follow it, up to the end of the containing box.

I.4.2.4 Index tables

I.4.2.4.1 General

The Codestream Index box may contain an index table for each of the following kinds of codestream data: main header, tile-parts, tile headers, (precinct) packets and packet headers. Each index table is a different type of box. There shall be no more than one of each kind of table in a Codestream Index box.

The Tile-part Index Table, Tile Header Index Table, Precinct Packet Index Table and Packet Header Index Table boxes are superboxes containing Fragment Array Index boxes. Below we define first the Fragment Array Index box and then the Index Table superboxes.

I.4.2.4.2 Fragment Array Index box

The Fragment Array Index box contains locations and lengths of the elements of a codestream. It is used within the Tile-part Index Table, Tile Header Index Table, Precinct Packet Index Table and Packet Header Index Table superboxes.

The type of a Fragment Array Index box shall be ‘faix’ (0x6661 6978). The contents of the Fragment Array Index box shall be as follows (Figure I.5):

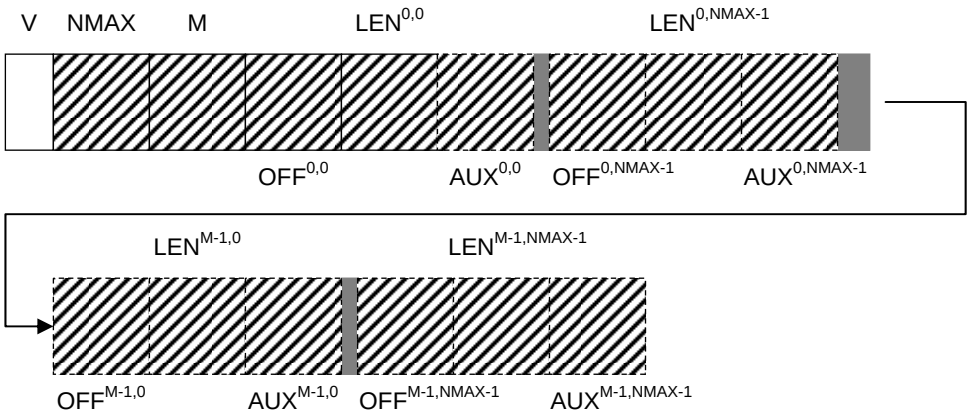

Figure I.I5 — Organization of the contents of a Fragment Array Index box

V: Version. If 0, all subsequent $OFF^{i,j}$ and $LEN^{i,j}$ fields are encoded as 4-byte big endian unsigned integers and $AUX^{i,j}$ fields are not present. If 1, all subsequent $OFF^{i,j}$ and $LEN^{i,j}$ fields are encoded as 8-byte big endian unsigned integers and $AUX^{i,j}$ fields are not present. If 2, all subsequent fields are encoded as 4-byte big endian unsigned integers. If 3, all subsequent $OFF^{i,j}$ and $LEN^{i,j}$ fields are encoded as 8-byte big endian unsigned integers and all subsequent $AUX^{i,j}$ fields are encoded as 4-byte big endian unsigned integers. All other values of V are reserved. This field is encoded as a 1-byte unsigned integer.

NMAX: Maximum number of valid elements in any row of the array. When used inside a codestream index table, NMAX is the maximum number of elements that will be specified for any tiles.

M: Number of rows of the array. When used inside a codestream index table, M is the number of tiles.

$OFF^{i,j}$: Offset. This field specifies the offset in bytes (relative to the start of the codestream) of the j^{th} element in row *i* of the array.

$LEN^{i,j}$: Length. This field specifies the length in bytes of the j^{th} element in row *i* of the array.

$AUX^{i,j}$: Auxiliary. This field specifies auxiliary information about the j^{th} element in row *i* of the array. The value of this field shall be zero unless otherwise permitted by the superbox containing this box. All nonzero values of this field are reserved.

While all rows of the array specified in the Fragment Array Index box shall be stored with NMAX number of elements, the object being described by that row may have a smaller number of elements to specify. In this case, where for any row *i* containing V valid elements where V is less than NMAX, the values of $OFF^{i,V}$ to $OFF^{i,NMAX–1}$ and $LEN^{i,V}$ to $LEN^{i,NMAX–1}$ shall be set to zero.

I.4.2.4.3 Header Index Table box

The Header Index Table box indexes the main header of a codestream or the tile-part headers of a tile, indicating the total main header length or first tile-part length and the locations and lengths of marker segments in the header. All marker segments shall be included, except that the SOT marker segment may be omitted for tile-part headers that consist of only SOT and SOD. Marker segments need not be listed in the order in which they occur in the codestream. The Header Index Table box may occur only inside a Codestream Index box. At the top level, it indexes a codestream and shall occur no more than once. Inside a Tile Header Index Table box, it indexes tile-part headers.

NOTE The intent is to provide an efficient means for skipping over pointer information in the header, which is not required for efficiently browsing the file but may unnecessarily bulk out the header. Listing multiple marker segments with the same marker code contiguously in the Header Index Table box will allow readers to skip over groups of marker segments in which they are not interested.

The type of a Header Index Table box shall be ‘mhix’ (0x6D68 6978). The contents of the Header Index Table box shall be as follows (Figure I.6):

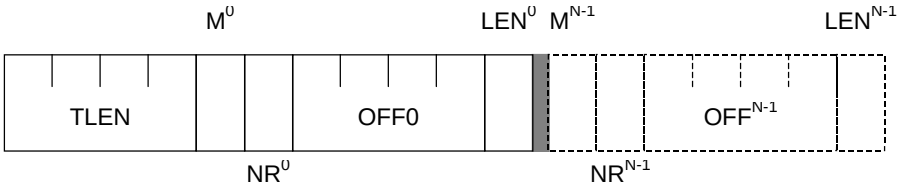

Figure I.I6 — Organization of the contents of a Header Index Table box

TLEN: Length. When the Header Index Table box indexes a main header, this field specifies the total length of the main header. When the Header Index Table box indexes tile-part headers, this field specifies the total length of the first tile-part header. The value of this field is encoded as an 8-byte big endian unsigned integer.

M^i: Marker code. This field specifies the marker code beginning the i^{th} marker segment listed in this box. The value of this field is encoded as a 2-byte big endian unsigned integer.

NR^i: Number remaining. This field indicates that (at least) NR^i marker segments with the same marker code M^i are listed immediately and contiguously following the i^{th} marker segment in this list. The value of this field is encoded as a 2-byte big endian unsigned integer.

OFF^i: Offset. This field specifies the offset in bytes, relative to the start of the codestream, of the marker segment parameters (including the length parameter but not the marker itself) for the i^{th} marker segment in this list. The value of this field is encoded as an 8-byte big endian unsigned integer.

LEN^i: Length. This field specifies the length in bytes of the marker segment parameters (including the two bytes of the length parameter but not the two bytes of the marker itself) for the i^{th} marker segment in this list. The value of this field is encoded as a 2-byte big endian

unsigned integer, and is the same as the value of the length parameter in the marker segment itself.

The number of marker segments, *N*, listed in the Header Index Table box, is determined by the length of the Header Index Table box.

I.4.2.4.4 Tile-part Index Table box (superbox)

The Tile-part Index Table box indexes the locations and lengths of each tile-part in the codestream, where each tile-part commences with its SOT marker and finishes with the last packet of the tile-part.

The type of a Tile-part Index Table box shall be ‘tpix’ (0x7470 6978). The contents of the Tile-part Index Table box shall be as follows (Figure I.7):

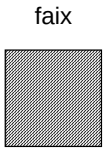

Figure I.I7 — Organization of the contents of a Tile-part Index Table box

faix: Fragment Array Index box. This box lists the locations and lengths of all the tile-parts in the codestream. Its structure is specified in I.3.2.4.2. The m^{th} row in this table corresponds to the m^{th} tile in the codestream. The entries on this row hold the locations and lengths of all tile-parts in the corresponding tile, in codestream order. If the Fragment Array Index box contains Auxiliary fields, these fields specify for each tile-part the number of complete resolutions starting from the lowest resolution which would be available for all components in the tile if this tile-part were combined with all previous tile-parts of the same tile. Thus the value 1 is used when the lowest resolution is complete, and the value is the number of wavelet transform levels plus one when all resolutions have been delivered. Because resolutions do not necessarily appear in order in a tile some resolutions above the value given in the Auxiliary field may have been completed, but this cannot be determined from the message header.

I.4.2.4.5 Tile Header Index Table box (superbox)

The Tile Header Index Table box indexes the tile headers of each tile, for the correct decoding of precinct packet data.

The type of a Tile Header Index Table box shall be ‘thix’ (0x7468 6978). The contents of the Tile Header Index Table box shall be as follows (Figure I.8):

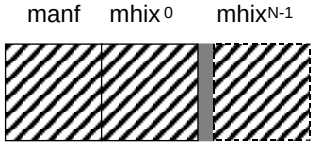

Figure I.I8 — Organization of the contents of a Tile Header Index Table box

The number of Header Index Table boxes, N, is the number of tiles.

manf: Manifest box. This box summarises the boxes specified by $mhix^i$ inside this Tile Header Index Table box. Its structure is specified in I.4.2.3.

$mhix^i$: Header Index Table box. This box indexes the tile-part headers for the i^{th} tile. Its structure is specified in **Error! Reference source not found.**.

I.4.2.4.6 Precinct Packet Index Table box (superbox)

The Precinct Packet Index Table box indexes the packets within the codestream. The type of a Precinct Packet Index Table box shall be ‘ppix’ (0x7070 6978). The contents of the Precinct Packet Index Table box shall be as follows (Figure I.9):

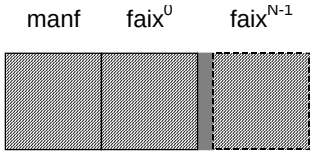

Figure I.I9 — Organization of the contents of a Precinct Packet Index Table box

The number of Fragment Array Index boxes, N, is the number of codestream components.

manf: Manifest box. This box summarises the boxes specified by $faix^i$ inside this Precinct Packet Index Table box. Its structure is specified in I.4.2.3.

$faix^i$: The i^{th} Fragment Array Index box corresponds to the i^{th} image component in the codestream. The m^{th} row in this table corresponds to the m^{th} tile in the codestream. The entries on this row hold the locations and lengths of all packets in the corresponding tile-component. Packets appear contiguously within their respective precincts, and precincts appear in the order associated with the sequence number *s*, defined in A.3.2. However, the fixed order of the packets is not necessarily the same as that specified in any COD/POC marker segments within the codestream.

If packet headers are packed into PPM or PPT marker segments, the corresponding entries in the fragment array refer to the location and length of the packet body only, as it appears inside its tile-part body. Entries that refer to non-existent packets (either because the relevant tile-component contains fewer packets than another tile-component in the same array, or because the codestream has been truncated prior to the point at which that packet would have existed) should have their location field set to zero. Entries that refer to packets whose body is empty and whose header consists of exactly one byte, 0x80, may be identified using a length value of zero. Such packets occur frequently in JPEG 2000 codestreams; applications may avoid the overhead of explicitly fetching such packets whose content is predictable. If the relevant COD marker segment specifies that EPH markers are to appear after each packet header in some tile, the special length value of zero shall be interpreted in that tile as meaning that the packet consists of the 0x80 byte followed by the EPH marker.

I.4.2.4.7 Packet Header Index Table box (superbox)

The Packet Header Index Table box indexes the packet headers within the codestream. The type of a Packet Header Index Table box shall be ‘phix’ (0x7068 6978). The contents of the Packet Header Index Table box shall be as follows (Figure I.10):

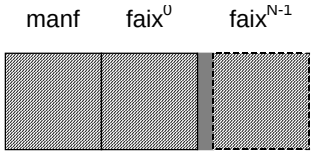

Figure I.I10 — Organization of the contents of a Packet Header Index Table box

The number of Fragment Array Index boxes, N, is the number of codestream components.

manf: Manifest box. This box summarises the boxes specified by $faix^i$ inside this Packet Header Index Table box. Its structure is specified in I.3.2.3.

faix^i: The i^{th} Fragment Array Index box corresponds to the i^{th} image component in the codestream. The m^{th} row in this table corresponds to the m^{th} tile in the codestream. The entries on this row hold the locations and lengths of all packet headers in the corresponding tile-component. Packet headers appear contiguously within their respective precincts, and precincts appear in the order associated with the sequence number *s*, defined in A.3.2. However, the fixed order of the packet headers is not necessarily the same as that specified in any COD/POC marker segments within the codestream.

Entries that refer to non-existent packet headers (either because the relevant tile-component contains fewer packets than another tile-component in the same array, or because the codestream has been truncated prior to the point at which that packet header would have existed) should have their location field set to zero. Entries that refer to packets whose body is empty and whose header consists of exactly one byte, 0x80, may be identified using a length value of zero. Such packets occur frequently in JPEG 2000 codestreams; applications may avoid the overhead of explicitly fetching such packets whose content is predictable. If the relevant COD marker segment specifies that EPH markers are to appear after each packet header in some tile, the special length value of 0 shall be interpreted in that tile as meaning that the packet consists of the 0x80 byte followed by the EPH marker.

I.4.3 File Index box (superbox)

I.4.3.1 General

The File Index box can be used to find other indexes (in particular, the codestream index corresponding to a codestream) and arbitrary data within the file.

A root File Index box indexes the top level of the file. Any other File Index box indexes a superbox within the file. There shall be at most one File Index box with a given scope (top level or a particular superbox) within a given file.

The type of a File Index box shall be ‘fidx’ (0x6669 6478). The contents of the File index box shall be as follows (Figure I.11):

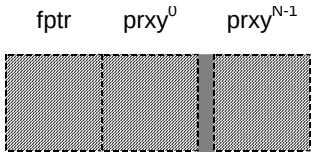

Figure I.I11 — Organization of the contents of a File Index box

fptr: File Finder box. A root File Index box shall not include this box. Any other File Index box shall include this box, which shall point to the superbox indexed by the File Index box. The structure of the File Finder box is defined in I.3.3.2.

prxy^i: Proxy box. This box represents a box in the portion of the file indexed by the File Index box. A root File Index box shall include proxies only for boxes at the top-level of the file. Any other File Index box shall include proxies only for boxes at the top level of the superbox indexed by the File Index box. The proxies shall occur in the same order as the boxes, but not all boxes need be proxied. The structure of the Proxy box is defined in I.3.3.3.

NOTE Because in some cases the presence, absence, or ordering of boxes in the file is significant, it may be helpful to applications if, preceding any such proxied boxes, no boxes within the scope of the index are omitted from the index.

I.4.3.2 File Finder box

The File Finder box points to a box. The type of a File Finder box shall be ‘fptr’ (0x6670 7472). The contents of a File Finder box shall be as follows (Figure I.12):

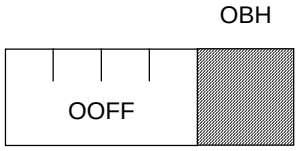

Figure I.I12 — Organization of the contents of a File Finder box

OOFF: Original Offset. This field specifies the offset in bytes (relative to the start of the file) of the box pointed to by this File Finder box. The value of this field is encoded as an 8-byte big endian unsigned integer.

OBH: Original Box Header. This field contains the complete box header of the box pointed to by this File Finder box. The length of this field is 16 bytes if the value of the LBox field contained within that box header is 1, or 8 bytes otherwise.

I.4.3.3 Proxy box

The Proxy box represents in a File Index box a box elsewhere in the file, indicating its location and length, the location and length of any index to the box, and a prefix of the contents of the box.

The type of a Proxy box shall be ‘prxy’ (0x7072 7879). The contents of the Proxy box shall be as follows (Figure I.13):

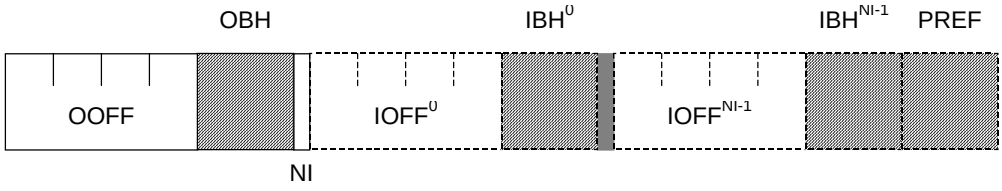

Figure I.I13 — Organization of the contents of a Proxy box

OOFF: Original Offset. This field specifies the offset in bytes (relative to the start of the file) of the box represented by this Proxy box. The value of this field is encoded as an 8-byte big endian unsigned integer.

OBH: Original Box Header. This field contains the complete box header of the box represented by this Proxy box. The length of this field is 16 bytes if the value of the LBox field contained within that box header is 1, or 8 bytes otherwise.

NI: Number of Indexes. This field indicates the number of index pointers included in this Proxy box. Each set of subsequent $IOFF^i$, and IBH^i fields points to either a File Index or a Codestream Index box that indexes the box represented by this Proxy box. All other values are reserved. The value of this field is encoded as a 1-byte unsigned integer.

$IOFF^i$: Index Offset. This field contains the offset in bytes (relative to the start of the file) of the ith index box. The value of this field is encoded as an 8-byte big endian unsigned integer.

IBH^i: Index Box Header. This field contains the complete box header of the i^{th} index box. The length of this field is 16 bytes if the value of the LBox field contained within that box header is 1, or 8 bytes otherwise.

PREF: Prefix. This field contains an arbitrary prefix of the data in the box represented by this Proxy box. It may have any length from zero up to the length of the content of the original box.

I.4.4 Index Finder box

The Index Finder box points to the root File Index box of a file. It shall occur only if the file contains a root File Index box. The type of an Index Finder box shall be ‘iptr’ (0x6970 7472). The contents of an Index Finder box shall be as follows (Figure I.14):

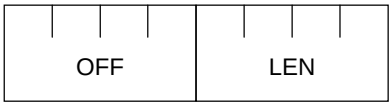

Figure I.I14 — Organization of the contents of an Index Finder box

OFF: Offset. This field specifies the location of the root File Index box relative to the start of the file. This field is stored as an 8-byte big endian unsigned integer.

LEN: Length. This field specifies the size of the root File Index box. This field is stored as an 8-byte big endian unsigned integer.

I.5 Association of codestream indexes with codestreams

In a JP2, JPX or JPM file, the Codestream Index box shall occur at the top level of the file and the i^{th} Codestream Index box shall correspond to the i^{th} codestream, also at the top level of the file. The Codestream Finder box within the Codestream Index box also indicates the codestream that is indexed by the Codestream Index box.

I.6 Placement restrictions (informative)

Few placement restrictions have been imposed on the boxes defined in this annex. They may be placed at the end of the file if desired; this is likely to be convenient when a non-indexed file is subsequently indexed. However, it may be helpful to place the Index Finder box near the beginning of the file, preferably immediately after any boxes that are required to be in a contiguous group at the beginning of the file (such as after the File Type box in a JP2 file or after the Reader Requirements box in a JPX file), where it may easily be found by file readers. To minimise the movement of file boxes, on the addition of this box and optionally the addition of a ‘jpip’ code to the compatibility list in the File Type box, a Free box (defined in ISO/IEC 15444–2 Annex M.11.20) could be used as a placeholder for it in a yet-to-be-indexed file.

Annex J
(normative)

Registration of extensions to this standard

J.1 Introduction to registration

Registration is the process of adding extensions to the capabilities to this standard after the standard has been published. In this standard, many capabilities may be extended through registration. This clause identifies those items which may be extended by registration, the process by which capabilities may be registered, and the process by which the Registration Authority will publish the those extensions. Only items that are specified in this clause may be extended by registration.

J.2 Registration Elements

The registration process is composed of the following elements.

— **Registration Authority:** The organizational entity responsible for reviewing, maintaining, distributing, and acting as a point of contact for all activities related to the registration. The registration Authority shall be ISO/IEC JTC 1/SC 29/WG 1.

— **Submitter:** The submitter is the organization or person who requests that the item be registered.

— **Review board:** The review board is the organizational entity that approves the registration of a proposed item. It is composed of an ad hoc committee appointed by the Review Board Chair. The review board shall be the WG1 JPIP committee.

— **Review board chair:** The review board chair is responsible for seeing that each candidate item is considered. He communicates with the submitter through the Registration Authority. The review board chair shall be the chair of the WG1 JPIP committee.

— **Test:** Rational that the Review Board should use to determine if submission/item should be registered.

— **Submission/Item:** This is the proposal for registration. Each proposal shall include the name of the item to be extended, the proposed tag/identity for the extension, and a rational/purpose for the extension.

J.3 Items which can be extended by registration

J.3.1 Extended boxes inside a Placeholder box

New box types for boxes that will be used within the ExtendedBoxList field in the Placeholder box (A.3.6.3) shall be registerable. A proposal to register a new box type shall contain a complete definition of that box (box type and contents of the box), instructions on when a server may write this box inside a Placeholder box, and instructions on what a client may do when it encounters a Placeholder box containing this box.

The Review Board shall evaluate all submissions. The Review Board shall evaluate proposed boxes based on the following criteria:

— Does it meet a need not already met by other defined boxes?

— Is the binary format of the contents of the box sufficiently defined?

— Does the box meet a general need (i.e. streaming video applications in general) or a vendor specific need (i.e. a particular vendor's implementation of streaming video).

J.3.2 Codestream scope

New scope values for requesting specific codestreams using the Codestream request field (`scope` in C.4.6) shall be registerable. A proposal to register a new scope shall contain a complete definition of the format of the associated `TOKEN` value, instructions on how the server shall map that `TOKEN` value into the available codestreams in the logical target, and instructions on how the server shall respond in the Codestream response header.

The Review Board shall evaluate proposed codestream scopes based on the following criteria:

— Does it meet a need not already met by other defined scopes?

— Is the structure of TOKEN sufficiently defined?

— Does the scope meet a general need (i.e. streaming video applications in general) or a vendor specific need (i.e. a particular vendor's implementation of streaming video).

J.3.3 Channel transport

New channel transports (Annex H) shall be registerable. A proposal to register a new channel transport shall contain a complete definition of the transport, including the identifier to be used for that channel transport.

The Review Board shall evaluate proposed channel transports based on the following criteria:

— Does it meet a need not already met by other defined transports?

— Does the transport meet a general need (i.e. streaming video applications in general) or a vendor specific need (i.e. a particular vendor's implementation of streaming video).

— Is the mapping of JPIP request and response components to the elements of the transport protocol sufficiently defined?

J.3.4 Preferences

New client preferences shall be registerable. This includes new preference sets (new values of related-pref-set as specified in C.10.2.1, or new options for existing or registered preference set. A proposal to register a new preference option or preference set shall contain a complete definition of the syntax, meaning of new options, and instructions on how the server shall respond when acting according to that preference.

The Review Board shall evaluate proposed preferences based on the following criteria:

— Does it meet a need not already met by other defined preference or capabilities?

— Does the preference meet a general need (i.e. streaming video applications in general) or a vendor specific need (i.e. a particular vendor's implementation of streaming video).

J.4 Registration Process

The following is the registration process.

1) A submitter creates a candidate item for registration.

2) The candidate item is submitted to the Registration Authority.

3) The Registration Authority passes the candidate item to the Review Board Chair.

4) The Review Board Chair distributes the candidate item to the Review Board and schedules meetings, phone calls, etc. as appropriate for consideration of the item.

5) The Review Board shall evaluate all submissions. If the text of the submission does not meet the requirements, then it shall be return it to the submitter for clarification. Favour will be given to solutions that are more general, and proposed solutions that are highly vendor specific may be returned to the submitter to be made more general and more applicable to the industry at large.

6) If approved the Chair passes the approval to the Registration Authority who notifies ISO and the submitter, and makes the registered or published item available.

7) If declined, the Chair prepares a response document indicating why the item was declined and passes this to the Registration Authority who notifies the submitter.

J.5 Timeframes for the registration process

The Review Board shall respond to all requests for registration within seven months from the date of submission. Within that time period, the Review Board will meet at an official meeting of ISO/IEC JTC1/SC29/WG1 to evaluate the proposal, make a decision, and draft the response.

Annex K
(informative)

Application Examples

K.1 Introduction

K.2 Use of JPIP with codestreams in other file formats

JPIP may be used to access JPEG 2000 codestreams stored in file formats other than JPEG 2000 family files. For example, DICOM and PDF files both have the capability to contain JPEG 2000 codestreams. In a client server environment, some procedure not specified in this Standard may be used to locate the JPEG 2000 codestream. JPIP requests and responses may be used on the object once the codestream is located. The Subtarget request field is intended for just such a situation. Alternatively, a server could provide access to the codestreams via a different URL.

K.3 Tile-part Implementation Techniques

K.3.1 Server determination of relevant tile-parts for a view-window request

For communication via tile-part the mapping of a view-window to a set of tiles is easy. The desired region of the image is converted to “reference grid units.” The XTsiz and YTsiz portions of the SIZ marker segment are used to determine which tiles intersect with the region of interest. The resolution level and quality are used to determine the tile-parts needed.

From the codestream, the SOT marker gives the details of tile indices and the number of bytes required for each tile. From the codestream, the appropriate bytes, corresponding to the tile parts that need to be sent, are transmitted to the client side. In case the view-window changes, and the corresponding relevant tiles also change, then only the tiles that have not been sent earlier are sent to update the display image.

K.3.2 Decoding an image from returned tile-part messages

JPIP specifies mechanisms to communicate compress image data and metadata between a client and a server. The mechanisms for the client to display the returned data are not specified, and indeed will vary widely with the application. This clause provides information on obtaining component samples from returned data.

A client application, that has received all of the main header data (indicated by the completed header data-bin appearing in a response message for header-data-bin 0), may concatenate that data-bin with the tile-part data-bins to form a legal JPEG 2000 codestream. This codestream may be provided to a conformant JPEG 2000 decoder and the result displayed. Of course for efficiency purposes a client may wish to provide view-window parameters to an intelligent decoder along with the codestream so only portions needed for the current view-window will be displayed.

K.4 JPIP protocol transcripts

In the following example transcripts, the text following the symbols “>>” at the beginning of a line is sent from the client to the server, the text following the symbols “<<” at the beginning of a line is sent from the server to the client, and the text following the symbols “--" is a comment and is not actually transmitted. The comments may indicate that some of the data transmitted is not shown.

K.4.1 Using HTTP

The following transcript shows five requests sent from the client to the server and the response of the server.

The first request asks for the JP2 file called phoenix.jp2, the first codestream in the file is requested, a maximum length is put on the response, a target id is requested, the data is requested to be returned as a JPP-stream, and establishment of a session over HTTP is requested. No window, and hence no image data is requested.

The server replies providing a target ID for the image, and an ID for >the newly established channel. The header line starting "JPIP-cnew" indicates a new path that can be used to access the image file. The value for the path "jpip" may be a path to a CGI program on the server designed to deal with all JPIP interactive commands. Some data from the file is returned in the body these will be file format boxes, and perhaps the main header of the first codestream.

The client's 2nd request uses the new path, "jpip", and the channel ID to identify the desired image (no image name or target ID is necessary). This request also specifies a particular window of interest.

The response to the 2nd request indicates that the view-window has been changed and a smaller window centered in the requested view window is being returned. The server starts returning the data for this window.

Before receiving the complete response to the 2nd request, the client issues a 3rd request. The client has adjusted it's view window to the size specified by server.

The server continues to respond to the 2nd request for a while, then starts a response to the 3rd request. During this response, the client issues a 4th request with a slightly different region. The server continues to respond to the 3rd request for a while then starts responding to the 4th request.

The client waits until the 4th response has completed, then issues a request to terminate both the session and the HTTP connection. There is no response data shown in this case as the connection closes.

```
<< GET /phoenix.jp2?stream=0&len=2000&tid=0&type=jpp-stream&cnew=http HTTP/1.1
<< Host: dst-m
<<
    >> HTTP/1.1 200 OK
    >> JPIP-tid: 281B6E135135BBC0BC588452AC9B73C5
    >> JPIP-cnew: cid=JPH_033C38BE48115AC9,path=jpip.cgi,transport=http
    >> Cache-Control: no-cache
    >> Transfer-Encoding: chunked
    >> Content-Type: image/jpp-stream
    >>
    >> 102
       -- 258 bytes of binary data
    >> 0
    >>
<< GET /jpip.cgi?fsiz=834,834&roff=0,0&rsiz=834,790&comps=0-
2&stream=0&len=2000&cid=JPH_033C38BE48115AC9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> HTTP/1.1 200 OK, with modifications
    >> JPIP-roff: 120,114
    >> JPIP-rsiz: 593,561
    >> Cache-Control: no-cache
    >> Transfer-Encoding: chunked
    >> Content-Type: image/jpp-stream
    >>
    >> 393
       -- 915 bytes of binary data
<< GET /jpip.cgi?fsiz=834,834&roff=120,114&rsiz=593,561&comps=0-
```

```
2&stream=0&len=2000&cid=JPH_033C38BE48115AC9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> 3f9
       -- 1017 bytes of binary data
    >> 0
    >>
    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >> Transfer-Encoding: chunked
    >> Content-Type: image/jpp-stream
    >>
    >> 359
       -- 857 bytes of binary data
<< GET /jpip.cgi?fsiz=834,834&roff=309,297&rsiz=121,86&comps=0-
2&stream=0&len=3906&cid=JPH_033C38BE48115AC9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> 234
       -- 564 bytes of binary data
    >> 3d0
       -- 976 bytes of binary data
    >> 24f
       -- 591 bytes of binary data
    >> 0
    >>
    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >> Transfer-Encoding: chunked
    >> Content-Type: image/jpp-stream
    >>
    >> 3b2
       -- 946 bytes of binary data
    >> 400
       -- 1024 bytes of binary data
    >> 263
       -- 611 bytes of binary data
    >> 356
       -- 854 bytes of binary data
    >> 209
       -- 521 bytes of binary data
    >> 0

<< GET /jpip.cgi?cclose=JPH_033C38BE48115AC9,len=0 HTTP/1.1
<< Host: dst-m
<< Connection: close
<< Cache-Control: no-cache
<<
```

K.4.2 Using HTTP with TCP return

```
<< GET /phoenix.jp2?stream=0&len=2000&tid=0&type=jpp-stream&cnew=http-tcp,http
HTTP/1.1
<< Host: dst-m
<<
    >> HTTP/1.1 200 OK
    >> JPIP-tid: 281B6E135135BBC0BC588452AC9B73C5
    >> JPIP-cnew: cid=JPHT033C38BE481154F9,path=jpip,transport=http-
```

```
tcp,auxport=80
    >> Cache-Control: no-cache
    >>
        << JPHT033C38BE481154F9 – [Note: This is the TCP channel connection
message]
        <<

<< GET /jpip?fsiz=834,834&roff=0,0&rsiz=834,790&comps=0-
2&stream=0&cid=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> HTTP/1.1 200 OK, with modifications
    >> JPIP-roff: 120,114
    >> JPIP-rsiz: 593,561
    >> Cache-Control: no-cache
    >>

<< GET /jpip?fsiz=834,834&roff=229,254&rsiz=155,113&comps=0-
2&stream=0&cid=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >>

<< GET /jpip?fsiz=1667,1667&roff=457,507&rsiz=310,226&comps=0-
2&stream=0&cid=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >>

<< GET /jpip?fsiz=3334,3334&roff=914,1014&rsiz=620,452&comps=0-
2&stream=0&cid=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >>

<< GET /jpip?cclose=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
```

K.5 Using JPIP with HTML

A JPIP system can be used with HTML pages in a variety of ways. If a JPIP server includes the ability to transcode portions of an image to JPEG or other complete image media types, then HTML can be used to access portions of a JPEG 2000 image without any changes to current browsers.

Consider a web page containing the following html:

```
<img
```

```
src=http://jpip.jpeg.org/name.jp2?fsiz=128,128&rsiz=128,128&type=image/jpeg
width="128" height="115">
```

Any web browser wishing to display this web page with images will issue a request to obtain the image. This request will start to:

```
GET /name.jp2?fsiz=128,128&rsize=128,128&type=image/jpeg
Host: jpip.jpeg.org
```

and will include many other HTTP header lines, typically identifing the browser, and the types of things the browser accepts. This HTTP request is a legal JPIP request and a JPIP server which receives this request shall either return an error message or determine the relavent portion of the JP2 file to access and translate it to a JPEG file. The returned message could look like:

```
HTTP/1.1 200 OK
Content-type: image/jpeg
Content-length: 20387
CRLF
JPEG-Compressed-Image-Data
```

Which is a legal JPIP response, and is also a legal HTTP response that all image browsers know how to display. Note that it is preferred but not required for the server to use the chunked transfer coding so that this request could be interrupted.

It is also possible to write web pages which will use JPEG when only JPEG is available, use JPEG 2000 when available, and JPT-stream or JPP-stream when available in the client's browser. Consider the HTML:

```
<img src=http://jpip.jpeg.org/name.jp2?rsiz=128,128 width="128" height="115">
```

In this case there is no explict type requested. A JPIP server using HTTP should therefore examine the "Accept:" line of the HTTP request issued by the server. Depending on the presence of image/jp2 or image/jpt-stream or image/jpp-stream or image/jpeg, the server can determine a compatible format to return.

Annex L
(informative)

APIs

[Editor's note: Text to be added.]

Bibliography

[1] D. Taubman, “Remote Browsing of JPEG 2000 Images”, Proc. Int. Conf. on Image Processing, vol. 1, pp. 229—232, Sep. 2002.

[2] J. Li, H. Sun, H. Li, Q. Zhang, X. Lin, “Vfile – A Virtual File Media Access Mechanism and its Application in JPEG2000 Images for Browsing over Internet,” ISO/IEC JTC1/SC29/WG1 N1473, Nov. 1999.

[3] M. Boliek, G. K. Wu, and M. J. Gormish, “JPEG 2000 for Efficient Imaging in a Client / Server Environment”, Proc. SPIE Conf. on Applications of Digital Image Processing, vol. 4472, pp. 212—223, Dec. 2001.

[4] S. Deshpande and W. Zeng, “Scalable Streaming of JPEG2000 Images Using Hypertext Transfer Protocol”, Proc. ACM Conf. on Multimedia, pp. 372—381, Oct. 2001.

[5] A. Wright, R. Clark, and G. Colyer, “An Implementation of JPIP Based on HTTP”, ISO/IEC JTC1/SC29 WG1 N2426, Feb. 2002.

[6] M. Gormish, and S. Banerjee, "Tile-Based Transport of JPEG 2000", N. Garcia, J.M. Martinez, L. Salgado (Eds.): VLVB03, LNCS 2849, pp. 217-224, 2003.

[7] D. Taubman, and R. Rosenbaum, "Rate-Distortion Optimized Interactive Browsing of JPEG2000 Images," Proc. Int. Conf. on Image Processing, Sep. 2003.

[8] D. Taubman and R. Prandolini, "Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000," presented at Visual Communications and Image Processing, Lugano, Switzerland, 2003.