ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation

ITU-T T.808 (V2) (12/2022)

SERIES T: Terminals for telematic services

Still-image compression – JPEG 2000

Information technology – JPEG 2000 image coding system: Interactivity tools, APIs and protocols

ITU-T T-SERIES RECOMMENDATIONS

TERMINALS FOR TELEMATIC SERVICES

Facsimile – Framework	T.0–T.19
Still-image compression – Test charts	T.20–T.29
Facsimile – Group 3 protocols	T.30–T.39
Colour representation	T.40–T.49
Character coding	T.50–T.59
Facsimile – Group 4 protocols	T.60–T.69
Telematic services – Framework	T.70–T.79
Still-image compression – JPEG-1, Bi-level and JBIG	T.80–T.89
Telematic services – ISDN Terminals and protocols	T.90–T.99
Videotext – Framework	T.100–T.109
Data protocols for multimedia conferencing	T.120–T.149
Telewriting	T.150–T.159
Multimedia and hypermedia framework	T.170–T.189
Cooperative document handling	T.190–T.199
Telematic services – Interworking	T.300–T.399
Open document architecture	T.400–T.429
Document transfer and manipulation	T.430–T.449
Document application profile	T.500–T.509
Communication application profile	T.510–T.559
Telematic services – Equipment characteristics	T.560–T.619
General multimedia application frameworks	T.620–T.649
User interfaces – Accessibility and human factors	T.700–T.799
Still-image compression – JPEG 2000	**T.800–T.829**
Still-image compression | JPEG XR	T.830–T.849
Still-image compression – JPEG-1 extensions	T.850–T.899

For further details, please refer to the list of ITU-T Recommendations.

INTERNATIONAL STANDARD ISO/IEC 15444-9
RECOMMENDATION ITU-T T.808

Information technology – JPEG 2000 image coding system: Interactivity tools, APIs and protocols

Summary

Rec. ITU-T T.808 | ISO/IEC 15444-9 provides a network protocol that allows for the interactive and progressive transmission of JPEG 2000 coded data and files from a server to a client. The first edition of this Recommendation | International Standard dates to 2005. It has since then been supplemented by amendments and corrigenda. Additionally, other members of the JPEG 2000 family of Recommendations | International Standards, that are capable of being used with the network protocol described in this Recommendation | International Standard have since been introduced. This second edition incorporates the changes associated with these developments, without modifying the original scope.

This Recommendation was developed jointly with ISO/IEC JTC 1/SC 29/WG 1 (JPEG), and is common text with ISO/IEC 15444-9.

This second edition cancels and replaces the first edition, which has been technically revised.

The main changes compared to the previous edition are as follows:

1. consolidates all outstanding amendments and corrigenda published since the first edition;
2. extends support for the file format specified in Rec. ITU-T T.815 | ISO/IEC 15444-16;
3. clarifies normative server responsibilities in response to certain request fields documented in Annex C;
4. removes the registration authority (Annex L); and
5. adds media type registration information (Annex O).

This Recommendation contains an electronic attachment that is available from the ITU website at: https://handle.itu.int/11.1002/2000/7460, and from the ISO website at: https://standards.iso.org/iso-iec/15444/-9\ed-2/en.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T T.808	2005-01-08	16	11.1002/1000/7460
1.1	ITU-T T.808 (2005) Amd. 1	2006-05-29	16	11.1002/1000/8815
1.2	ITU-T T.808 (2005) Cor. 1	2007-01-13	16	11.1002/1000/9049
1.3	ITU-T T.808 (2005) Amd. 2	2007-08-29	16	11.1002/1000/9232
1.4	ITU-T T.808 (2005) Cor. 2	2008-06-13	16	11.1002/1000/9517
1.5	ITU-T T.808 (2005) Amd. 3	2008-06-13	16	11.1002/1000/9516
1.6	ITU-T T.808 (2005) Amd. 4	2010-05-22	16	11.1002/1000/10646
1.7	ITU-T T.808 (2005) Cor. 3	2011-05-14	16	11.1002/1000/11315
1.8	ITU-T T.808 (2005) Amd. 5	2013-03-16	16	11.1002/1000/11884
2.0	ITU-T T.808 (V2)	2022-12-14	16	11.1002/1000/15209

Keywords

API, application programme interface, image coding, interactivity, JPEG 2000, protocols.

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

CONTENTS

Figures

Page

Tables

Introduction

Rec. ITU-T 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 Recommendation | 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.

The substantive updates in this edition, compared to Edition 1, are:

1. consolidates all outstanding amendments and corrigenda published since the first edition;
2. extends support the file format specified in Rec. ITU-T T.815 | ISO/IEC 15444-16;
3. clarifies normative server responsibilities in response to certain request fields documented in Annex C;
4. removes the registration authority (Annex L); and
5. adds media type registration information (Annex O).

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – JPEG 2000 image coding system: Interactivity tools, APIs and protocols

1 Scope

This Recommendation | International Standard[1] defines, in an extensible manner, syntaxes and methods for the remote interrogation and optional modification of JPEG 2000 codestreams and files in accordance with their definition in Rec. ITU-T T.800 | ISO/IEC 15444-1 and other members of the Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | Standards.

In this Recommendation | 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 can be exchanged in a manner that avoids or minimises the communication of information not required by client. This Recommendation | International Standard includes definitions of the semantics and values to be exchanged, and suggests how these can be passed using a variety of existing network transports.

With JPIP, the following tasks can 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 files, JPEG 2000 codestreams and other container files:
  - selective data segments;
  - selective and defined structures;
  - parts of an image or its related metadata.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions 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.

- Recommendation ITU-T T.800 | ISO/IEC 15444-1, *Information technology – JPEG 2000 image coding system: Core coding system.*
- Recommendation ITU-T T.801 | ISO/IEC 15444-2, *Information technology – JPEG 2000 image coding system: Extensions.*
- Recommendation ITU-T T.802 | ISO/IEC 15444-3, *Information technology – JPEG 2000 image coding system: Motion JPEG 2000.*
- Recommendation ITU-T T.805 | ISO/IEC 15444-6, *Information technology – JPEG 2000 image coding system: Compound image file format.*
- Recommendation ITU-T T.809 | ISO/IEC 15444-10, *Information technology – JPEG 2000 image coding system: Extensions for three-dimensional data.*
- Recommendation ITU-T T.814 | ISO/IEC 15444-15, *Information technology – High-Throughput JPEG 2000*.
- Recommendation ITU-T T.815 | ISO/IEC 15444-16, *Information technology – Encapsulation of JPEG 2000 images into ISO/IEC 23008-12.*

[1] This Recommendation | International Standard contains an electronic attachment that is available from the ITU website at: https://handle.itu.int/11.1002/2000/7460, and from the ISO website at: https://standards.iso.org/iso-iec/15444/-9\ed-2/en.

- IETF RFC 768 (1980), *User Datagram Protocol*. Available from World Wide Web: http://www.ietf.org/rfc/rfc0768.txt.
- IETF RFC 2046 (1996), *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. Available from World Wide Web: http://www.ietf.org/rfc/rfc2046.txt.
- IETF RFC 2616 (1999), *Hypertext Transfer Protocol – HTTP/1.1*. Available from World Wide Web: http://www.ietf.org/rfc/rfc2616.txt.
- IETF RFC 3986 (2005), *Uniform Resource Identifiers (URI): Generic Syntax*. Available from World Wide Web: https://datatracker.ietf.org/doc/html/rfc3986.
- IETF RFC 5234 (2008), *Augmented BNF for Syntax Specifications: ABNF*. Available from World Wide Web: https://datatracker.ietf.org/doc/html/rfc5234.
- IETF RFC 9293 (2022), *Transmission Control Protocol*. Available from World Wide Web: https://datatracker.ietf.org/doc/html/rfc9293.

3 Definitions

For the purposes of this Recommendation | International Standard, the terms and definitions given in Rec. ITU-T T.800 | ISO/IEC 15444-1 and Rec. ITU-T T.801 | ISO/IEC 15444-2 apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at https://www.iso.org/obp
- IEC Electropedia: available at https://www.electropedia.org

3.1 JPEG 2000 definitions

The following definitions are used within this Recommendation | International Standard. In some cases, these definitions differ from those used in other standards and/or Recommendations.

3.1.1 **J2KI**: A file that conforms to the `'j2ki'` brand specified in Rec. ITU-T T.815 | ISO/IEC 15444-16.

3.1.2 **J2KS**: A file that conforms to the `'j2ks'` brand specified in Rec. ITU-T T.815 | ISO/IEC 15444-16.

3.1.3 **JPH**: The file format specified in Rec. ITU-T T.814 | ISO/IEC 15444-15: High Throughput JPEG 2000.

3.1.4 **JPEG 2000 codestream**: Codestream conforming to the specification in Rec. ITU-T. T.800 | ISO/IEC 15444-1, possibly including capabilities specified elsewhere.

3.1.5 **JPEG 2000 family file**: File conforming to one of the file formats defined in the Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | Standards.

3.1.6 **JPM**: The file format specified in Rec. ITU-T T.805 | ISO/IEC 15444-6.

3.1.7 **metadata**: Any collection of "boxes" from a JPEG 2000 family file.

3.1.8 **MJ2**: The file format specified in Rec. ITU-T T.802 | ISO/IEC 15444-3.

3.2 HTTP definitions

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

3.2.1 **connection**: Transport layer virtual circuit established between two programs for the purpose of communication.

3.2.2 **entity**: The information transferred as the payload of a request or response.

Note on entry: An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body.

3.2.3 **proxy**: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

Note on entry: Requests are serviced internally or by passing them on, with possible translation, to other servers.

3.3 JPIP definitions

The following definitions are used within this Recommendation | International Standard. In some cases, these definitions differ from those used in other standards and/or Recommendations.

3.3.1 cache (client-side): Cache managed by the Client for storing JPIP data-bins.

Note on entry: The Client might have a limited cache and might have to purge cached JPIP data-bins from time to time.

3.3.2 cacheable response: Response that may be stored within a cache for use in answering subsequent requests.

Note on entry: Even if a resource is cacheable, there might be additional constraints on whether a cache can use the cached copy for a particular request.

3.3.3 cache-model (server-side): Server-side estimation of the data-bins or portions of data-bins that are available in the client's cache.

Note on entry: The server can add items to its estimation of the client's cache because it assumes successfully delivery, or because it has received acknowledgements of transmitted data, or because of cache-model update statements.

3.3.4 channel: Mechanism for grouping requests and responses such that only one request/response is active at a time within the group.

Note on entry: Multiple channels can be used to issue multiple requests and receive multiple responses concurrently.

3.3.5 client: Program that establishes connections for the purpose of sending requests.

3.3.6 codestream image region: Intersection between the image and the region defined by the Offset and Region Size.

Note on entry: The codestream image region can be empty (no area).

3.3.7 data-bin: Set of bytes of the same type of data which can be partially delivered.

3.3.8 incremental-codestream: Representation of the codestream as a collection of data-bins (main header, tile header, precinct or tile data-bins) having the same codestream identifier.

3.3.9 JPIP index table: File format box which provides information about the location of portions of a file or codestream.

3.3.10 JPEG 2000 family target: Target that corresponds to a JPEG 2000 family file.

3.3.11 logical target: Specific representation of specific original named resource, or a byte range from that specific original named resource, to which the JPIP request is directed.

Note on entry: The specific representation might be transcoded from the original named resource.

3.3.12 message: Set of bytes from a single data-bin and the header identifying those bytes and the data-bin.

3.3.13 raw codestream: Representation of the codestream as a single metadata-bin.

3.3.14 request: Group of fields and values sent from the client to the server to obtain portions of an image or metadata.

3.3.15 resource: Network data object or service that can be identified by a URI.

3.3.16 response: Bytes sent from the server to the client after receiving a request.

3.3.17 server: Application program that accepts connections in order to service requests by sending back responses.

Note on entry: Any given program might 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.

3.3.18 session: Collection of requests and responses applying to the same resource for which the server maintains a cache model.

3.3.19 session-based: Where the server maintains a cache model.

3.3.20 stateless: Single request where the server does not make use of a cache-model in determining the response.

3.3.21 target: Logical identification of JPIP data.

Note 1 on entry: This is the name of the main target and is often the name of a file on the server.

Note 2 on entry: JPEG 2000 files or codestreams might be available in multiple representations (e.g., return type, precinct size) or vary in other ways, each identified by a unique logical target.

3.3.22 tile header: All tile-part headers for a specific tile.

3.3.23 view-window: The portion of the image data that the client desires, as expressed by the combination of the following fields that appear in the request: Region Size, Offset, Frame Size, Codestream, Codestream Context, Sampling Rate, ROI and Layers.

Note on entry: The view-window is often smaller than the whole image data.

3.3.24 slice: Subset of voxels in a volumetric image with the same Z coordinate.

3.3.25 profile: Set of request fields that a server is expected to support and implement and a client communicating with a server in this profile may issue, with the expectation that the server supports them.

Note 1 on entry: Conformance is structured according to profiles.

Note 2 on entry: See Annex J.

3.3.26 variant: Collection of operating modes and features of the JPIP standard that a client and a server use to exchange requests and data.

Note on entry: Clients and servers interoperate by employing a common subset of all variants.

3.4 Symbols

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

c	An index (starting from 0) of the image component to which the precinct belongs
fx	x-axis frame size for client request view-window
fy	y-axis frame size for client request view-window
fz	z-axis frame size for client request view-window
fx'	x-axis frame size for codestream resolution selected by the server based on the client request
fy'	y-axis frame size for codestream resolution selected by the server based on the client request
fz'	z-axis frame size for codestream resolution selected by the server based on the client request
fx"	Modified jpx x-axis frame size for suitable resolution
fy"	Modified jpx y-axis frame size for suitable resolution
fz"	Modified jpx z-axis frame size for suitable resolution
H_{cod}	The codestream height as recorded in the Image Header (ihdr) box (see Annex I.5.3.1 of Rec. ITU-T T.800 | ISO/IEC 15444-1)
H_{comp}	The height of the composited result, supplied in the JPX composition options box (see Annex M.11.10.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
H_{reg}	The height of the compositing layer, as it appears on the compositing layer registration grid
Hs_{inst}	The cropped height
Ht_{inst}	The composited height
I	The unique identifier of a precinct within its codestream
N_L	Is the number of decomposition levels
num_components	The number of components encoded
num_tiles	The number of tiles in the codestream
ox	x-axis offset for client request view-window
oy	y-axis offset for client request view-window
oz	z-axis offset for client request view-window
ox'	x-axis offset for suitable codestream region
oy'	y-axis offset for suitable codestream region
oz'	z-axis offset for suitable codestream region
ox"	Modified jpx x-axis offset for suitable region
oy"	Modified jpx y-axis offset for suitable region
oz"	Modified jpx z-axis offset for suitable region

r	Resolution level
s	A sequence number which identifies the precinct within its tile-component
sx	x-axis size of client request view-window
sy	y-axis size of client request view-window
sz	z-axis size of client request view-window
sx'	x-axis size for suitable codestream region
sy'	y-axis size for suitable codestream region
sz'	z-axis size for suitable codestream region
sx"	Modified jpx x-axis size for suitable region
sy"	Modified jpx y-axis size for suitable region
sz"	Modified jpx z-axis size for suitable region
t	An index (starting from 0) of the tile to which the precinct belongs
W_{cod}	The codestream width as recorded in the Image Header (ihdr) box (see Annex I.5.3.1 of Rec. ITU-T T.800 | ISO/IEC 15444-1)
W_{comp}	The width of the composited result, supplied in the JPX composition options box (see Annex M.11.10.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
W_{reg}	The width of the compositing layer, as it appears on the compositing layer registration grid
Ws_{inst}	The cropped width
Wt_{inst}	The composited width
XC_{inst}	The x-axis cropping offset supplied via the relevant instruction (see Annex M.11.10.2.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
XO_{inst}	The x-axis compositing offset, described via the relevant compositing instruction (see Annex M.11.10.2.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
XO_{reg}	The x-axis codestream registration offset
XOsiz	The horizontal offset from the origin of the reference grid of the relevant codestream's SIZ marker segment
XR_{reg}	The x-axis codestream registration sampling factor, described at the beginning of any codestream registration box (see Annex M.11.7.7 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
Xsiz	The width of the reference grid of the relevant codestream's SIZ marker segment
XS_{reg}	The x-axis registration precision described at the beginning of any codestream registration box (see Annex M.11.7.7 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
YC_{inst}	The y-axis cropping offset supplied via the relevant instruction (see Annex M.11.10.2.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
YO_{inst}	The y-axis compositing offset, described via the relevant compositing instruction (see Annex M.11.10.2.1 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
YO_{reg}	The y-axis codestream registration offset
YOsiz	The vertical offset from the origin of the reference grid of the relevant codestream's SIZ marker segment
YR_{reg}	The y-axis codestream registration sampling factor, described at the beginning of any codestream registration box (see Annex M.11.7.7 of Rec. ITU-T T.801 | ISO/IEC 15444-2)
Ysiz	The height of the reference grid of the relevant codestream's SIZ marker segment

YS_{reg}	The y-axis registration precision described at the beginning of any codestream registration box (see Annex M.11.7.7 of Rec. ITU-T T.801 | ISO/IEC 15444-2)

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply.

ABNF	Augmented Backus-Naur Form
DICOM	Digital Imaging and Communications in Medicine
DWT	Discrete Wavelet Transformation
EOR	End of Response
HTML	Hypertext Markup Language
IP	Internet Protocol
JPIP	JPEG 2000 Interactive Protocol
JPP	JPIP Precinct
JPT	JPIP Tile-part
MTF	Modulation Transfer Function
PDF	Portable Document Format
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VBAS	Variable-length Byte Aligned Segment
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

5 Conventions

5.1 ABNF rules

This Recommendation | International Standard uses the ABNF notation defined in RFC 5234, including the core ABNF syntax rules: ALPHA (letters), CR (carriage return), CRLF (Internet standard newline), CTL (control characters), DIGIT (decimal digits), HEXDIG (hexadecimal digits), LF (line feed), LWSP (linear white space) and SP (space). For the purposes of this Recommendation | International Standard, the following ABNF rules also apply.

```
NZDIGIT = %x31-39          ; 1-9
UPPER = %x41-5A            ; A-Z
LOWER = %x61-7A            ; a-z
UINT = 1*DIGIT
NONZERO = *"0" NZDIGIT *DIGIT
UINT-RANGE = UINT ["-" [UINT]]
UFLOAT = 1*DIGIT ["." 1*DIGIT]
OCTAL-ENCODED-CHAR = "\" QUADDIG OCTDIG OCTDIG
QUADDIG = "0" / "1" / "2" / "3"
OCTDIG = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7"
UUID = 16(HEXDIG)
TOKEN = 1*(ALPHA / DIGIT / "." / "_" / "-")
TEXT-LABEL = DQUOTE TOKEN DQUOTE
IDTOKEN = 1*(TOKEN / ";")
```

This Recommendation | International Standard also defines `PATH`, representing a file or pathname. In the general case, `PATH` values may contain any character, although for a given server architecture, the server shall reject any characters that are not valid on that particular server. In addition, `PATH` shall be properly encoded as specified by the transport technology.

`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 an ABNF 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 "`1$`" refers to one or more repetitions of the parameter that follows, each occurrence of which is separated by a semicolon.

5.2 File format ABNF rules

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

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

box-type-list = "*" / 1#(box-type)
```

`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`).

`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 (\040), then that character shall be encoded as the underscore character ("`_`") or by octal encoding. For any other character, a 4-character string is written in its place, consisting of a backslash character ("`\`") followed by three octal digits representing the value of the character from the box type in octal. The `compatibility-code` is encoded the same way that a `box-type` is encoded.

5.3 Key to graphical descriptions of boxes (informative)

The description of each box is followed by a figure that shows the order and relationship of the parameters in the box. Figure 1 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 grey area between indicate a run of several of these parameters. A sequence of two groups of multiple parameters with superscripts separated by a grey 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 1, parameters A, B, C and D are 8, 16, 32 bit and variable length respectively. The notation E^0 and E^{N-1} implies that there are N different parameters, E^i, in a row. The group of parameters F^0 and F^{M-1}, and G^0 and G^{M-1} specify that the box will contain F^0, followed by G^0, followed by F^1 and G^1, continuing to F^{M-1} and G^{M-1} (M instances of each parameter in total). Also, the field D is optional and might 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 the 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, can be found in place of the ellipsis.

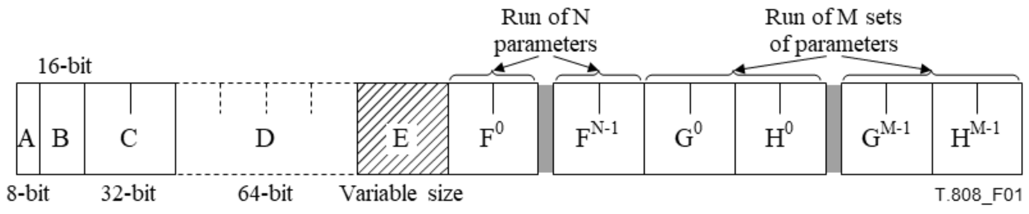

Figure 1 – Example of the box description figures

For example, the superbox shown in Figure 2 contain an AA box and a BB box, and the BB box follows the AA box. However, there can be other boxes found between boxes AA and BB. Dealing with unknown boxes is discussed in Rec. ITU-T T.800 | ISO/IEC 15444-1.

AA
•••
BB
T.808_F02

Figure 2 – Example of the superbox description figures

6 General description

6.1 JPIP protocol

This Recommendation | 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 Recommendation | International Standard enables the flexibility and functionality intended in Rec. ITU-T T.800 | ISO/IEC 15444-1 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 3. This Recommendation | International Standard defines the JPIP client requests and the JPIP server responses. HTTP/1.1 (RFC 2616), TCP (RFC 9293) and UDP (RFC 768) are shown as examples of possible transports for JPIP. 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 responds by delivering 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 can request information about an image as defined in JPIP 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.

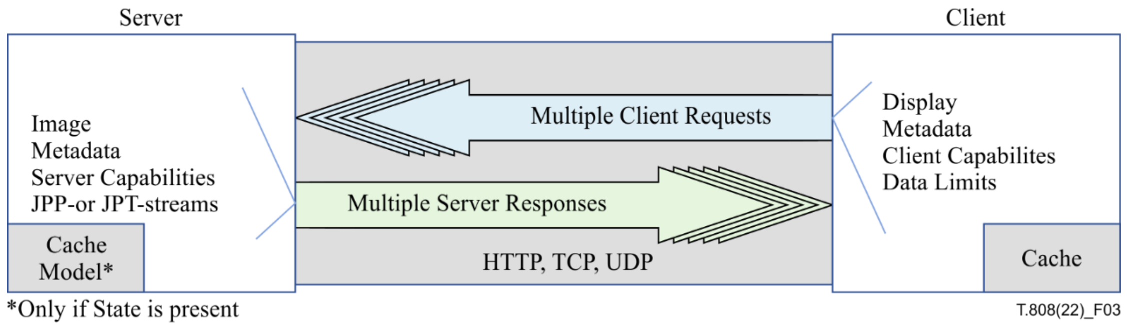

Figure 3 – JPIP protocol overview

This protocol can be used over several different transports as shown in Figure 4. This Recommendation | International Standard includes informative annexes on the use of the JPIP protocol over HTTP, TCP and UDP, and provides suggestions for other example implementations. 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 ("cnew") request field (see C.3.3) and the New Channel ("JPIP-cnew") response header (see D.2.3), where transport-specific details shall be communicated. This Recommendation | International Standard defines four specific transports, which are identified by the strings "http", "https", "http-tcp" and "http-udp" in the value string associated with New Channel requests.

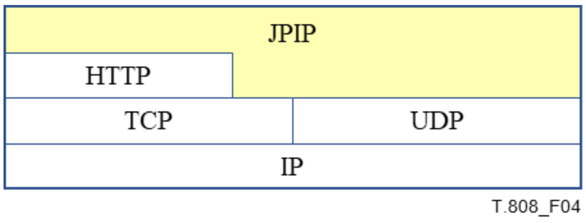

Figure 4 – JPIP protocol stack

The JPIP protocol described in this Recommendation | International Standard provides support for still images, volumetric imagery, motion imagery and animated content, along with a multitude of file formats, as specified by other members of the Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | International Standards.

6.2 Purpose

This Recommendation | International Standard defines the syntax and methods required for both the client and server. Annexes define components that are required to achieve interoperability and functionality between the client and server over several transports. These annexes 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 requests and the server shall be able to parse, interpret and respond to all compliant requests.

– Annex D defines the server response syntax. The server shall produce compliant responses 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.

– Annexes F, G and 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.

– Server and client conformance is further structured into profiles and variants. Profiles define which fields servers are required to support and implement beyond simply parsing and interpreting the fields. Variants define the operating modes and features of the JPIP standard a client and server use to transmit data. Clients and servers need to provide a common subset of variants in order to interoperate. See Annex J for details about conformance and testing for conformance.

– Annex K describes several examples of using this Recommendation | International Standard for several different applications.

Additional annexes found in this Recommendation | International Standard are as follows.

– Annex L is provided to support extensions to this Recommendation | International Standard.

– Annex M provides example JPIP applications and protocol transcripts.

– Annex N collects the ABNF specifications for client requests and server responses, as found in Annex C and Annex D, respectively.

– Annex O provides media type specifications and registration information.

7 Conformance

Conformance with this Recommendation | International Standard by a client means that the client's JPIP requests are well structured, valid and conformant to the JPIP client requests as defined by this Recommendation | International Standard, and that it is able to parse the JPIP responses defined by this Recommendation | International Standard.

Conformance with this Recommendation | International Standard by a server means that the server's JPIP responses are well structured, valid and conformant to the JPIP server response signalling as defined by this Recommendation | International Standard, and is able to parse the JPIP requests defined by this Recommendation | International Standard. Servers shall parse and interpret all normative request types and shall respond to all compliant requests. Compliance to a profile requires servers furthermore to support and implement all mandatory fields within that profile to the extent defined in Annex J.

Conformance, profiles and conformance testing methodologies of this Recommendation | International Standard are defined in Annex J.

It is expected that server applications might send additional data not explicitly requested by a client, or redundant data that has previously been sent to a client, depending on the network quality-of-service. Such implementation decisions are application specific and provide the JPIP system with high utility.

Annex A

The JPP-stream and JPT-stream media types

(This annex forms an integral part of this Recommendation | International Standard.)

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 can be terminated at any point and messages can 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.

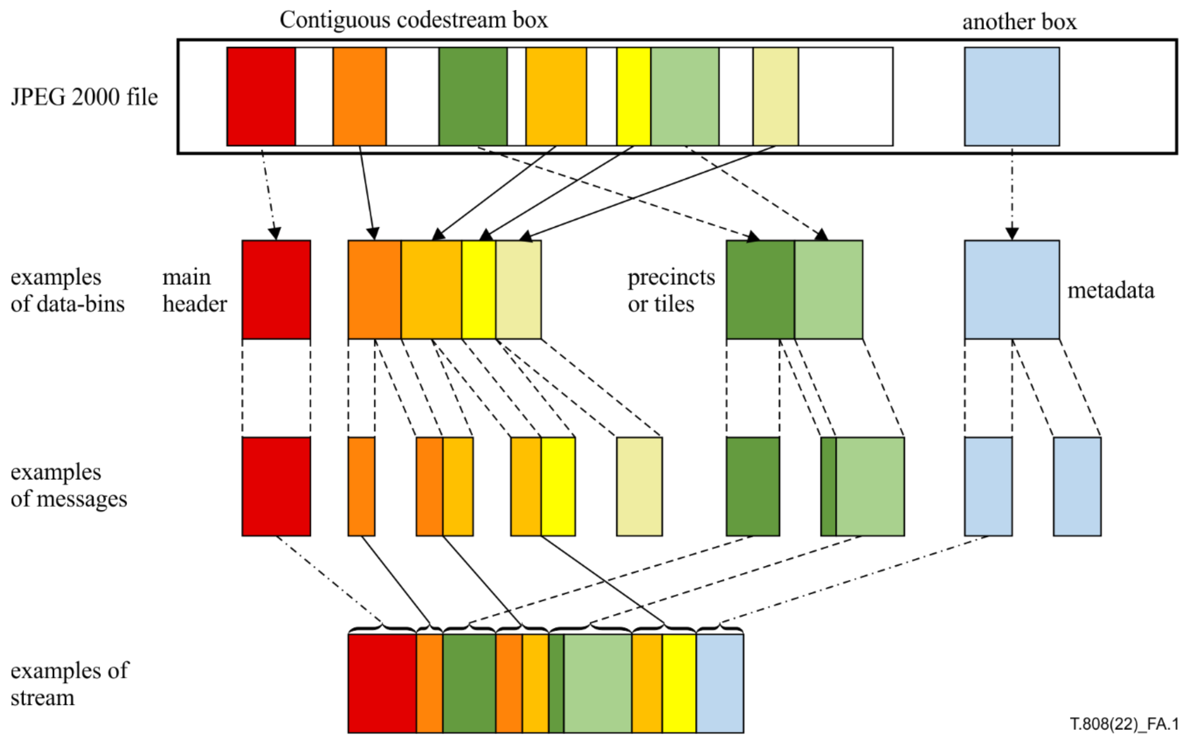

Figure A.1 – Examples of a JPEG 2000 file, JPIP data-bins and JPIP-stream relationships (after G.J. Colyer and R.A. Clark, IEEE Trans. Consumer Electronics, 49 (2003), pp. 850–854)

Figure A.1 is an illustrative example of the relationship between the bit-streams from a JPEG 2000 file, JPIP data-bins, and a JPIP stream. The figure shows the main header colour coded red, 2 precincts with packets coded in shades of orange-yellow and green, and a meta-data box coded blue. Self-describing JPIP messages are formed from these data-bins and concatenated to form a JPIP stream.

A JPIP stream consists of one or more concatenated JPIP messages. Each JPIP message consists of a header and a body. The header provides descriptive information to identify the relevant data-bin. The body is data from that data-bin. Unless further signalling is provided, the message is the concatenation of the header with the body.

NOTE – In this Recommendation | International Standard, all examples provided form binary messages by the concatenation of header and body. It is implementation-specific to the transport and application if other signalling for header and body is provided. For example, auxiliary signalling with variable error protection might be implemented for wireless-based applications.

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.2. The least significant 7 bits of each byte in the VBAS are concatenated to form a bit stream which is used in different ways for different VBASs.

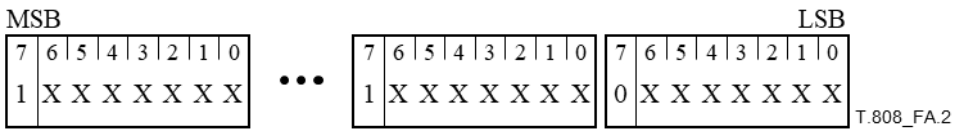

Figure A.2 – VBAS structure

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 dependent on the previous message. Applications can choose to use the long form message headers; these messages can be rearranged in any arbitrary order. Alternatively, applications can 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.

The Bin-ID VBAS serves several roles. Bits 6 and 5 of the first byte of the Bin-ID VBAS, labelled 'b' in Figure A.3, indicate whether the Class and CSn VBASs are present in the message header. Table A.1 defines the bit values and its meaning.

Bit 4 of the first byte of the Bin-ID VBAS, labelled 'c' in Figure A.3, 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.3) form an "in-class identifier", which is used to uniquely identify the data-bin within its class, in the manner described in A.2.3.

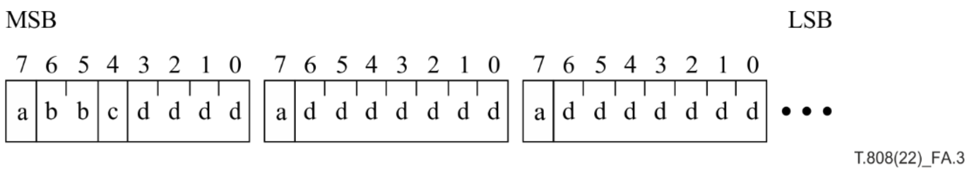

Figure A.3 – Bin-ID VBAS structure

Table A.1 – 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 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 non-negative 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 Aux VBAS can be present. Its presence, and meaning if present, is determined by the message class identifier found within the Bin-ID VBAS, as explained in A.2.2. If present, the Aux VBAS represents a non-negative integer value, formed by concatenating the least significant 7 bits of each byte in the VBAS in big-endian order. The information in the Aux VBAS cannot affect the length of the message body.

A.2.2 Message class identifiers

The message class identifiers defined by this Recommendation | International Standard are the non-negative integers shown in Table A.2. The interpretation of the data-bin classes to which they refer is described in 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 Aux 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.

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 Aux 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 Aux VBAS indicates the total number of quality layers associated with the precinct in the original codestream. Otherwise, the Aux VBAS indicates the quality layer to which the byte immediately following the last byte in the message belongs. The information in the Aux VBAS can be useful to certain clients.

Table A.2 – 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 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 Aux VBAS which identifies the smallest n such that, in all components for which $(N_L - n)$ is non-negative, resolution level $(N_L - n)$ and all lower resolution levels have been completed when the bytes in this message are combined with all preceding bytes of the same tile, where N_L is the number of decomposition levels, which can vary by component. If no resolution levels of any component have been completed, the value of the Aux VBAS is one plus the maximum value of N_L across all components. The value zero is reached when all resolutions in all components have been completed. Because resolutions do not necessarily appear in order in a tile, some resolution levels above the value signalled by the VBAS might have been completed, but this cannot be determined from the message header. The information in the Aux VBAS can 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. 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 JPEG 2000 file or codestream data. These can be based on imagery elements, such as precinct-based data, tile-based data, and headers. They can 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 is defined by Equation A-1.

$$I = t + (c + s \times \text{num\_components}) \times \text{num\_tiles} \qquad \text{(A-1)}$$

where:

I	is the unique identifier of the precinct within its codestream;
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;
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 sub-band 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 sub-band 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).

NOTE – For the sake of efficiency when serving an image containing PPM markers, a server can transcode the packed packet headers in the main header into the tile headers (PPT markers). Otherwise, a client would require tile-part length markers (TLM) to be sent. The server can alternatively transcode the image (transparently to the client) in such a way as to avoid the use of packed packet headers altogether.

For volumetric images encoded in JP3D (Rec. ITU-T T.809 | ISO/IEC 15444-10), the sequence number of precincts within a tile-component is computed as follows: All precincts of the lowest resolution level, i.e., those containing only the [L|X][L|X][L|X] samples are sequenced first, starting from zero, following a raster scan order as defined by Rec. ITU-T T.809 | ISO/IEC 15444-10. The precincts from each successive resolution level are sequenced in turn, again following the raster scan order defined by Rec. ITU-T T.809 | ISO/IEC 15444-10. The precinct with sequence number 0 thus refers to the front most, upper left hand precinct of the lowest resolution sub-band of the image component 0 in tile 0.

A.3.2.2 Precinct data-bin example (informative)

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

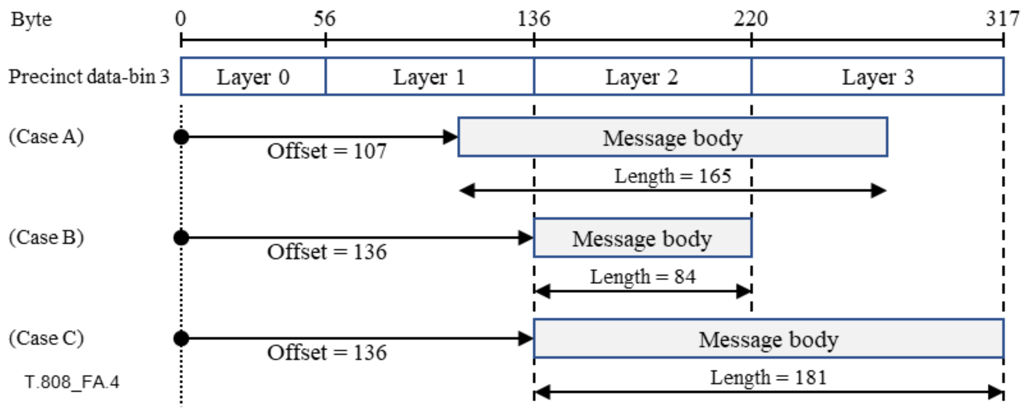

Figure A.4 – Example precinct data-bin

For Cases 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 Aux 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 third byte indicates that both this byte and at least the next byte are part of the Msg-Length VBAS. The 0 bit starting the fourth byte indicates that it is the last byte of the Msg-Length VBAS. Thus, all the low order bits from the third and fourth 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 response 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 can be formed from a valid codestream, by concatenating all marker segments except SOT in all tile-part headers for tile *n*.

NOTE 1 – POC marker segments can also be removed as they are not of interest to a typical JPIP client. However, a server might include the POC markers for the benefit of a client application that wants to output a JPEG 2000 file with the same progression order as the original image available at the server.

A server may send data in any order, but is required to send a tile header data bin for a tile even if the tile header is empty. A client that receives image data for a tile but has not yet received its tile header should not assume that the tile header is empty and attempt to decode the data.

NOTE 2 – It might be beneficial for certain clients to receive the tile header bin in advance of the tile data bin.

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 can be referenced by other data-bins associated with the JPP-stream or JPT-stream. For the purpose of this Recommendation | International Standard, 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 exception to this is that the metadata-bin containing the root of the logical target shall be assigned ID 0.

NOTE 1 – The mechanism for assignment is implementation-dependent; however, it is an informative suggestion that servers assign bin-ID's using consecutive numbers.

A server is required to send at least the metadata bin with bin ID 0, even if no metadata is present. In this case, the metabin #0 will be empty. A client should not assume that no metadata is available if it has not yet received any metadata bin.

NOTE 2 – It might be beneficial for certain clients to receive the metadata bin with bin ID 0 in advance of all other bins.

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 can 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 can 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 can themselves include sub-boxes, any of which might be replaced by further placeholders.

The following colour scheme will be used for metadata-bin example illustrations (Figure A.5):

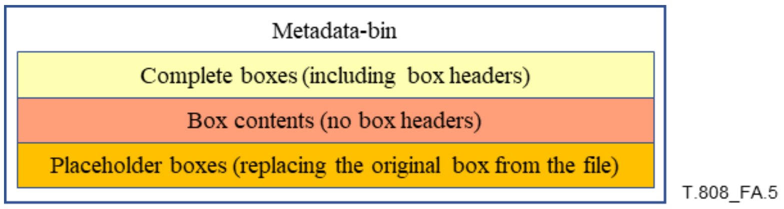
Metadata-bin
Complete boxes (including box headers)
Box contents (no box headers)
Placeholder boxes (replacing the original box from the file)
T.808_FA.5

Figure A.5 – Metadata-bin example colour scheme

As an example, consider a simple JP2 file with the following box structure (Figure A.6):

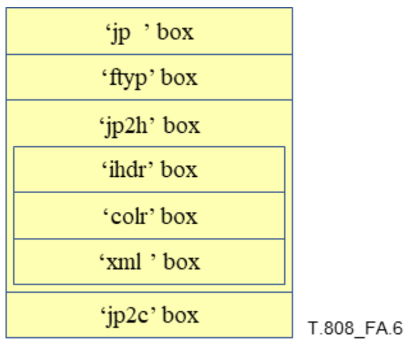

Figure A.6 – A sample JP2 file

This file can 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.7.

While the contents of any metadata-bin shall be the contents of the box or file represented by that bin, the actual data contained in those contents can conceptually vary depending on the type of box. For example, Metadata-bin 1 in Figure A.7 represents the contents of the JP2 Header box. The contents of that box is a sequence of other complete boxes, as the JP2 Header box is a superbox. By 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.7 is that access to codestream data can 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. For convenience of description in this Recommendation | International Standard, this shall be termed the "raw codestream" representation. Raw codestreams are served from metadata-bins.

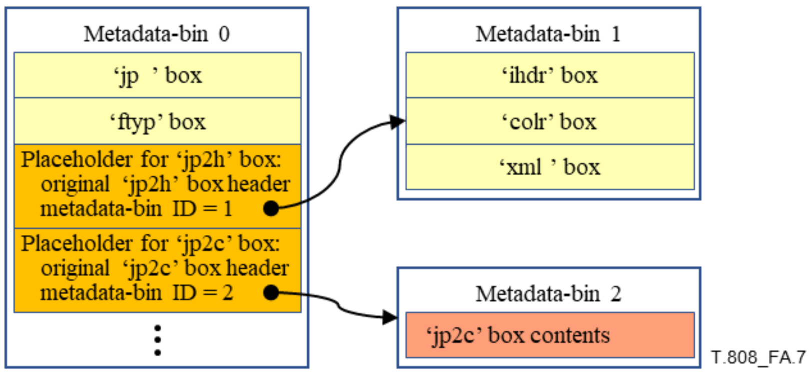

Figure A.7 – A sample JP2 file divided into three metadata-bins

The placeholder can 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 Recommendation | International Standard, this shall be termed the "incremental codestream" representation. Incremental codestreams are served from these data-bins.

In general, placeholders that reference codestream data can 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.3.6.4.

In the simple example of Figure A.7, Placeholder boxes appear only at the top-level of the file, in metadata-bin 0. As already noted, however, placeholders can 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 can 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 can 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.6, the XML data contained within the JP2 Header box can 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.8.

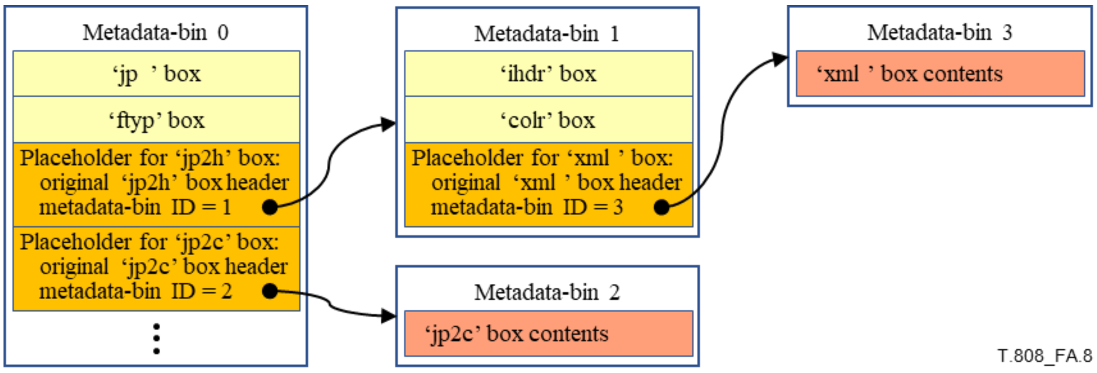
Metadata-bin 0
'jp ' box
'ftyp' box
Placeholder for 'jp2h' box: original 'jp2h' box header metadata-bin ID = 1
Placeholder for 'jp2c' box: original 'jp2c' box header metadata-bin ID = 2
Metadata-bin 1
'ihdr' box
'colr' box
Placeholder for 'xml ' box: original 'xml ' box header metadata-bin ID = 3
Metadata-bin 3
'xml ' box contents
Metadata-bin 2
'jp2c' box contents
T.808_FA.8

Figure A.8 – A superbox with a referenced metadata-bin

It would not be valid, however, for the JP2 Header box to be left in metadata-bin 0, as shown in Figure A.9:

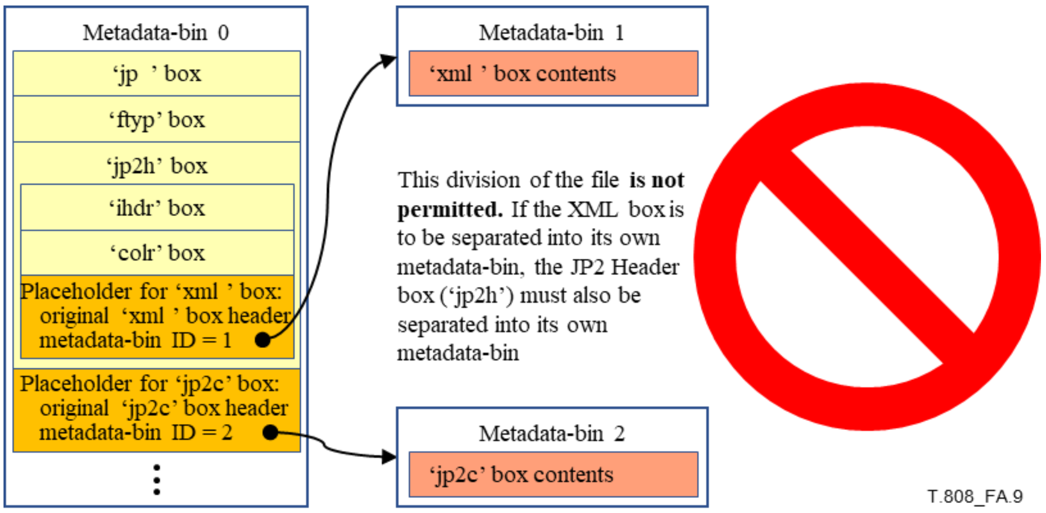
Metadata-bin 0
'jp ' box
'ftyp' box
'jp2h' box
'ihdr' box
'colr' box
Placeholder for 'xml ' box: original 'xml ' box header metadata-bin ID = 1
Placeholder for 'jp2c' box: original 'jp2c' box header metadata-bin ID = 2
Metadata-bin 1
'xml ' box contents
This division of the file is not permitted. If the XML box is to be separated into its own metadata-bin, the JP2 Header box ('jp2h') must also be separated into its own metadata-bin
Metadata-bin 2
'jp2c' box contents
T.808_FA.9

Figure A.9 – An invalid division of the file into metadata-bins

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 reconstructed 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.10) 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 can be created from scratch or they can 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.

Stream equivalents can 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 can 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.

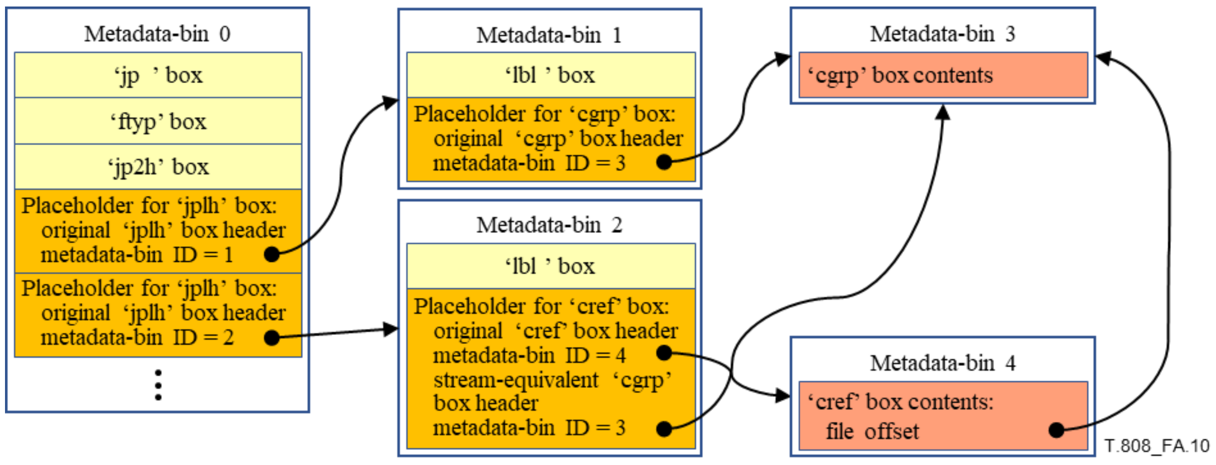
Metadata-bin 0
'jp ' box
'ftyp' box
'jp2h' box
Placeholder for 'jplh' box: original 'jplh' box header metadata-bin ID = 1
Placeholder for 'jplh' box: original 'jplh' box header metadata-bin ID = 2
Metadata-bin 1
'lbl ' box
Placeholder for 'cgrp' box: original 'cgrp' box header metadata-bin ID = 3
Metadata-bin 2
'lbl ' box
Placeholder for 'cref' box: original 'cref' box header metadata-bin ID = 4 stream-equivalent 'cgrp' box header metadata-bin ID = 3
Metadata-bin 3
'cgrp' box contents
Metadata-bin 4
'cref' box contents: file offset
T.808_FA.10

Figure A.10 – Example of the use of stream equivalents

A.3.6.3 Placeholder box format

Figure A.11 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 Recommendation | International 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.

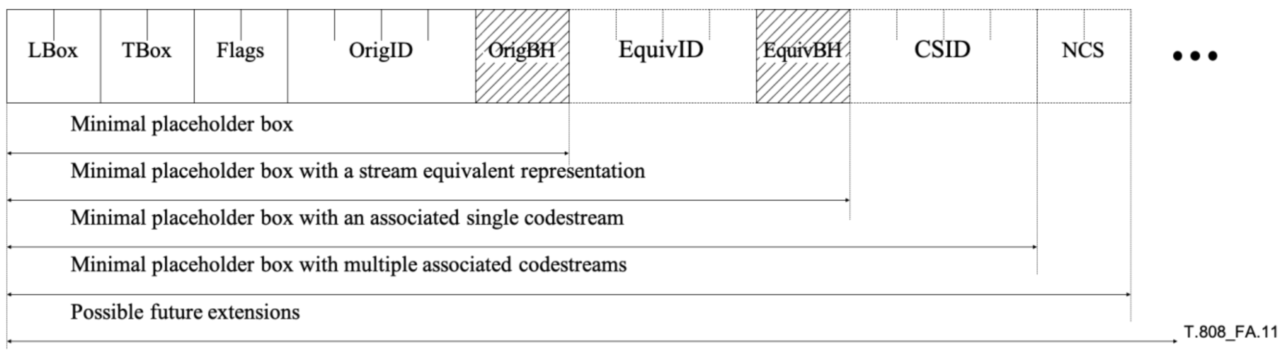
LBox
TBox
Flags
OrigID
OrigBH
EquivID
EquivBH
CSID
NCS
• • •
Minimal placeholder box
Minimal placeholder box with a stream equivalent representation
Minimal placeholder box with an associated single codestream
Minimal placeholder box with multiple associated codestreams
Possible future extensions
T.808_FA.11

Figure A.11 – 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. Valid 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.11. 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 Recommendation | International Standard. Clients shall ignore any box in ExtendedBoxList that is not understood.

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 for a Placeholder box need appear in every Placeholder box. As suggested by the arrows in Figure A.11, 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.

The above definition implies that the placeholder box can be truncated after the last used field, but intermediate fields need to be present, even if unused.

Table A.3 – Valid 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 it was set to "1" regardless of the actual value of that field when the field is present.
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.
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.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 JPEG 2000 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 codestreams that are found in Contiguous Codestream boxes or Fragment Table boxes, either at the top level of the file, or within Multiple Codestream boxes. The first 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 IDs can only be used where the meaning of those codestreams is well defined by the type of the box that is being replaced. For JPX files, Contiguous Codestream boxes, Fragment Table boxes and Multiple Codestream boxes can be replaced by Placeholder Boxes that specify codestream IDs. Placeholders replacing Contiguous Codestream boxes and Fragment Table boxes can only specify a single codestream ID, while a placeholder replacing a Multiple Codestream box can specify multiple codestream IDs, corresponding to the number of codestreams that are found within the box.

A.3.6.5 Using Placeholder boxes with MJ2 and J2KS files

This Recommendation | International Standard defines only two box types suitable for placeholders with Motion JPEG 2000 (MJ2) files or J2KS files. Specifically, either the chunk offset box ('stco') or the chunk large offset box ('co64') can be replaced by a Placeholder box which identifies multiple codestream ID's.

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 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.3.6.6 Using Placeholder boxes with J2KI files

As specified in Rec. ITU-T T.815 | ISO/IEC 15444-16, JPEG 2000 codestreams can be encapsulated as individual images or image collections within ISO/IEC 23008-12 files conforming to the `‘j2ki’` brand.

Individual images are encapsulated as *items*, as specified in Rec. ITU-T T.815 | ISO/IEC 15444-16. Incremental codestreams for all such items can be provided by replacing the Item Location Box with a Placeholder box which identifies one or multiple codestream ID's, as appropriate. If present, the Placeholder box's NCS field shall be equal to the number of items with type `'j2k1'`. If the Placeholder box has no NCS field, then the number of items with type `'j2k1'` shall be exactly 1. Incremental codestream IDs described by the Placeholder box, starting from CSID, shall correspond to the items that have type `'j2k1'`, in increasing order of their Item ID values.

A.4 Conventions for parsing and delivery of JPP-streams and JPT-streams

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 can choose to partition original boxes from a JPEG 2000 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 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, the following server behaviour is recommended:

– Unless a server advertises or has reason to believe otherwise, it should 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 advertises or has reason to believe otherwise, it should 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.

See Annex J for further guidance regarding expected server behaviour.

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 can contain the received JPEG 2000 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 can 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 valid 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 cache for a client. For example, a client might use a database to serve as its implementation of the cache function rather than a file-based cache system.

Annex B

Sessions, channels, cache model and model-sets

(This annex forms an integral part of this Recommendation | International Standard.)

B.1 Requests within a session 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 every request. Even more importantly, the server may keep a log of data it knows the client to have received so that this information need not be re-transmitted in response to future requests. This log is subsequently referred to as the cache model. The cache model would typically be persistent for the duration of a session. Unless explicitly instructed otherwise, the server may assume that the client caches all data it receives 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; 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". However, 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 can be adjusted through appropriate request fields.
- To the extent that the server queues requests, it should provide a separate queue for each JPIP channel.

There is a one-to-one correspondence for the client request and client response on a channel. Different JPIP channels can be on the same transport channel or on different transport channels. Requests that use different JPIP channels can arrive asynchronously at the server if separate transport channels are used to transport the requests. Responses that use different JPIP channels can arrive asynchronously at the client if separate transport channels are used to transport the responses. Servicing of multiple channels is at the discretion of the server; however, the delivery rate request field and the max bandwidth and bandwidth slice preferences should guide the server.

B.3 Cache model management

As already noted, one of the principal functions of a session is that of server-side modelling of the client's cache. Unless explicitly informed otherwise, the server **may** assume that the client has cached all information sent in response to requests within the session: this information **need not** be re-transmitted. However, that the server is not obliged to maintain a complete cache model or indeed any cache model at all: redundant data **may** be transmitted in response to requests.

In addition to the impact of transmitted data, explicit cache model manipulation statements in client requests can 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 not cached or has discarded some data which was previously sent by the server. 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 might 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 one cache model for each logical target in a 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 an "mset" (model-set) 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 can 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.

"mset" 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.8.6.

Annex C

Client request

(This annex forms an integral part of this Recommendation | International Standard.)

C.1 Request syntax

C.1.1 Introduction

This annex describes all possible elements in a JPIP request. Each major subclause 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 valid requests might 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 valid request, a server might not implement all possible request fields or combinations there-of, but it is required to parse and interpret all normative request fields and respond appropriately, even if this response is an error. Details of what servers are expected to implement are defined in Annex J.

NOTE – Which responses or methods for signalling error conditions are appropriate depends on the transport layer used. D.1 provides examples for servers using HTTP as transport protocol.

C.1.2 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 selected 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 can be part of the query field of a GET request, or the body of a POST request. See Annexes F, G and H for details.

The fields in the request shall be sent in compliance with the selected transport protocol. For example, in HTTP, the requests can be part of the query field of a GET request, or the body of a POST request, with individual request fields separated by a "&" character – see Annexes F, G and H for details. In contexts such as these, certain characters found within the BNF syntax or the request parameters might need to be escaped in order to avoid ambiguity. For example, a request field of the form "`target=me&my dog`" should be escaped in an HTTP context, becoming "`target=me%26my%20dog`", so as to avoid confusion with the "&" used to separate request fields. As another example, "`metareq=[roid/w]`" should be escaped in an HTTP context, becoming "`metareq =%5broid/w%5d`" so as to avoid the use of non-URI character – see IETF RFC 3986 for more on reserved characters, disambiguation and escaping via the hex-hex encoding. Parsers of requests found in such contexts should be prepared to perform hex-hex decoding of each request field.

```
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                         ; C.2.2
                   / subtarget                      ; C.2.3
                   / tid                            ; C.2.4
```

```
   channel-field      = cid                          ; C.3.2
                      / cnew                         ; C.3.3
                      / cclose                       ; C.3.4
                      / qid                          ; C.3.5

   view-window-field  = fsiz                         ; C.4.2
                      / roff                         ; C.4.3
                      / rsiz                         ; C.4.4
                      / fvsiz                        ; C.4.5
                      / rvoff                        ; C.4.6
                      / rvsiz                        ; C.4.7
                      / comps                        ; C.4.8
                      / stream                       ; C.4.9
                      / context                      ; C.4.10
                      / srate                        ; C.4.11
                      / roi                          ; C.4.12
                      / layers                       ; C.4.13
                      / mctres                       ; C.4.14

   metadata-field     = metareq                      ; C.5.2

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

   server-control-field = align                      ; C.7.1
                      / wait                         ; C.7.2
                      / type                         ; C.7.3
                      / drate                        ; C.7.4
                      / sendto                       ; C.7.5
                      / abandon                      ; C.7.6
                      / barrier                      ; C.7.7
                      / twait                        ; C.7.8

   cache-management-field = model                    ; C.8.1
                      / tpmodel                      ; C.8.3
                      / need                         ; C.8.4
                      / tpneed                       ; C.8.5
                      / mset                         ; C.8.6

   upload-field       = upload                       ; C.9.1

   client-cap-pref-field = cap                       ; C.10.1
                      / pref                         ; C.10.2
                      / csf                          ; C.10.3
                      / handled                      ; C.10.4
```

C.1.3 Restrictions on combining request fields

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

In general, requests for image data (view-window requests) can be combined with requests for additional metadata. However, there are restrictions 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 is directed to a specific representation of a specific original named resource or a specific portion of that resource. That resource might be a physically stored file or object, or might be something that is created virtually by the server upon request.

The specific representation, whether that is the original encoded form or a transcoded form, or whether that is a specific byte range or is the entire resource, is referred to as the logical target. The logical target is specified through three request fields: Target ID, Target and Sub-target.

The Target request field specifies the original named resource to which the request is directed. It is specified using a `PATH`, which could be a simple string or a URI. If the Target field is not specified and the request is carried over HTTP (or HTTPS), then the JPIP server shall direct the request to the resource specified through the path component of the JPIP request URL. This original named resource might be an actual file or other object physically stored on the server, or it might be something that the server creates in response to a JPIP request.

The Sub-target request field specifies the specific byte range of the original named resource (specified through the Target request field) to which the request is directed. If the Sub-target request field is not specified, the server shall direct the request to the entire byte range of the original resource.

The Target ID request field can be used to further specify a particular encoding of the resource in situations where the client and server have previously exchanged data from this resource. For example, the server might have previously supplied a transcoded version of the file to the client based on information supplied and the conditions around a previous request. If that client has preserved the data previously transmitted in its cache, it will desire to continue to receive data using that same transcoding so that it can continue to use the data in the cache. The Target ID is a server defined identification string, which the server has previously associated with that specific representation of that specific original named resource, or a byte range of some specific original named resource.

If a client specifies both the original named resource (through either the Target request field or through the path component of the JPIP Request URL) and Target-ID, the server shall verify whether or not it can respond to the request in the same manner as when it originally assigned that Target ID to that resource. If the server cannot respond in the same manner, it shall use a JPIP-tid response header to inform the client of a new Target ID, at which point the client will know that it needs to discard any previously cached data.

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

If the channel ID request field is included in the request, the request need not include Target, Sub-Target or Target ID fields.

The following examples illustrate the specification of logical targets:

EXAMPLE 1: For JPIP request URL of

"http://one.jpeg.org/imageserver.cgi?target= http%3A%2F%2Fone.jpeg.org%2Fimages%2Fpicture.jp2&fsiz=200,200" the logical target is the entire byte range contained within the URI "http://one.jpeg.org/images/picture.jp2," relative to the server root document directory.

EXAMPLE 2: For JPIP request URL of

"http://one.jpeg.org/imageserver.cgi? target= http%3A%2F%2Fone.jpeg.org%2Fimages%2Fpicture.jp2&tid=4384-5849-af4d-3dca&fsiz=200,200" the logical target is the entire byte range contained within the URI "http://one.jpeg.org/images/picture.jp2," relative to the server root document directory, with a representation specified by the server defined Target ID 4384-5849-af4d-3dca.

EXAMPLE 3: For JPIP request URL of

"http://one.jpeg.org/imageserver.cgi?target= http%3A%2F%2Fone.jpeg.org%2Fimages%2Fpicture.jp2&subtarget=1038-13458&fsiz=200,200" the logical target is the range of bytes, starting with byte 1038, and all bytes up to and including bytes 13458, contained within the URI "http://one.jpeg.org/images/picture.jp2," relative to the server root document directory.

EXAMPLE 4: For JPIP request URL of "http://one.jpeg.org/imageserver.cgi?cid=1234-5849-af4d-3dca&fsiz=200,200" the logical target is the resource to which the server has associated with the channel with ID 1234-5849-af4d-3dca.

EXAMPLE 5: For JPIP request URL of "http://one.jpeg.org/images/picture.jp2?fsiz=200,200" the logical target is the entire byte range contained within the file "images/picture.jp2," relative to the server root document directory.

EXAMPLE 6: For JPIP request URL of "http://one.jpeg.org/images/picture.jp2?subtarget=1038-13458&fsiz=200,200" the logical target is the range of bytes, starting with byte 1038, and all bytes up to and including byte 13458, contained within the file "images/picture.jp2," relative to the server root document directory.

C.2.2 Target (target)

```
target = "target" "=" PATH
```

This field is used to specify the original named resource (often the name of a file on the server). If the Target request field is missing, then the server shall determine the original named resource by other means.

C.2.3 Sub-target (subtarget)

```
subtarget = "subtarget" "=" byte-range / src-codestream-specs
```

```
byte-range = UINT-RANGE

src-codestream-specs = UINT-RANGE
```

This field can be used to qualify the original named resource through the specification of a byte range or a range of codestreams in the original resource. The logical target is to be interpreted as the indicated byte range or a range of codestreams of the original named resource. For the purpose of requests and responses involving this logical target, the server shall assign consecutive indices starting from zero to the codestreams in this target.

NOTE – Defining a logical target as a range of codestreams thus relabels the codestreams, and effectively replaces the codestream indices, if any, in the original resource by consecutive indices starting from zero.

The lower and upper bounds of the supplied range are inclusive, where bytes or codestreams are counted from zero.

C.2.4 Target ID (tid)

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

target-id = IDTOKEN
```

This field can 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, including any discretionary transcoding performed by the server. 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, together with the original resource name and byte range, should absolutely identify both the imagery and its encoding.

If `target-id` is "0", the logical target is specified through the use of the Target, Sub-target and JPIP URL path component, and the client is explicitly requesting that the server inform it of the assigned `target-id`, if there is one. The server shall include a Target ID header in its response to all client requests with a `target-id` of "0".

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

C.3 Fields for working with sessions and channels

C.3.1 Introduction

A request shall 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`.

See B.2 for discussions on sessions and channels.

C.3.2 Channel ID (cid)

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

channel-id = IDTOKEN
```

- This field is used to associate the request with a particular JPIP channel, and hence the session to which the channel belongs.

C.3.3 New Channel (cnew)

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

transport-name = TOKEN
```

This 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 Recommendation | International Standard defines only the transport names, "http", "https", "http-tcp" and "http-udp". Details of the use of JPIP over the "http" transport appear in Annex F. Annex G describes the use of JPIP over the "http-tcp" transport and Annex K describes the use of JPIP over the "http-udp" transport.

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 D.2.3). 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 server shall treat the request 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 used for the new channel. The cache models for both targets are associated with this common session.

C.3.4 Channel Close (cclose)

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

This 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 "*", the session shall be identified by the inclusion of a Channel ID request field, and the server shall close all channels associated with the session.

The server shall complete its response on any channel specified in the Channel Close request before actually closing the channel.

The combination of "wait = yes" with "cclose=*" is not recommended. If this situation is encountered, the server can decide which of the two takes priority.

C.3.5 Request ID (qid)

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

This field is used to specify a Request ID value. Each channel has its own request queue, with its own Request ID counter. The server may process requests which do not contain a Request ID, or whose Request ID is zero, on a first-come-first-served basis. However, it shall not process a request which arrives with a Request ID value of n until it has processed all requests with request ID values of n_0 to n–1, inclusive. Here n_0 is the `qid` supplied in the request which originally created the channel, or is equal to 1 if no `qid` was present when creating the channel.

NOTE – The response to a request containing `cnew` which results in the creation of a new channel is processed as if the request were issued in the new channel. This means the next request with a non-zero `qid` value to be processed in the new channel has the `qid` value n_0+1.

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 can be less than the full image in terms of image frame 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 JPEG 2000 codestream(s) identified in the request, not the sampled grid of any particular image component. More generally, however, clients can request higher level image objects (e.g., JPX compositing layers or MJ2 video tracks) via the Codestream Context request field (see C.4.10). In this case, the requested image portion might 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.4.10, and they shall be understood in terms of the following description of codestream image regions.

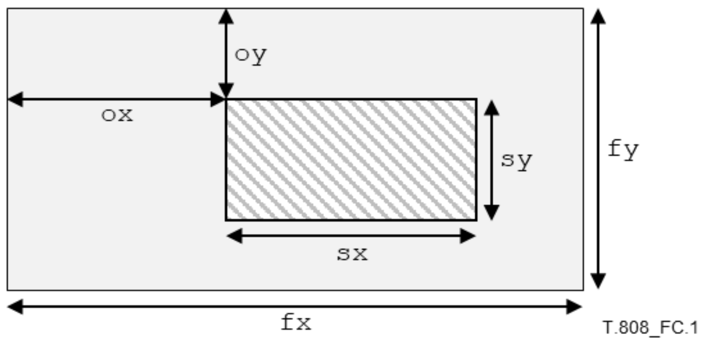

Figure C.1 –Desired region within an image

Codestream image regions are described using 3 n-dimensional parameters where *n* is the number of dimensions required to describe this image. The size parameters and offset parameters specify the extent and location of the desired codestream image region with respect to a whole image that has the given frame size. Figure C.1 demonstrates this set-up for regular images with *n* = 2, but the construction carries over naturally to a higher number of dimensions. For most of this subclause, only this case is considered, denoting the frame size by parameters fx and fy, the offset of the region by parameters ox and oy and denoting the size of the region by parameters sx and sy as indicated in Figure C.1.

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 resolutions in the JPEG 2000 codestream correspond exactly to the requested frame size, the returned image data might be larger or smaller than the requested frame size, and might 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 `sx'`, `sy'`, `ox'` and `oy'`, as shown in Figure C.2. Although the client can 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.

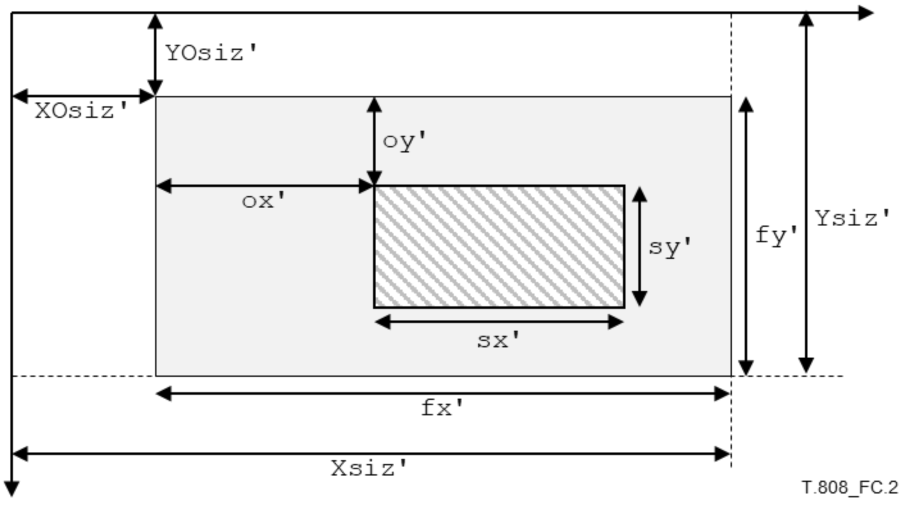

Figure C.2 – Desired region with respect to the subsampled reference grid

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

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

where:

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.

Here, XOsiz, YOsiz, Xsiz and Ysiz are taken from the relevant codestream's SIZ marker segment. It is natural to interpret *r* as a number of discarded highest DWT levels, and indeed *r* shall be an integer 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 the suitable frame size, `fx'` and `fy'`, have been found, the region size, `sx'` and `sy'`, and offset, `ox'` and `oy'`, associated with the codestream image region are determined by Equation C-2.

$$\mathrm{ox}' = \left\lfloor \mathrm{ox} \cdot \frac{\mathrm{fx}'}{\mathrm{fx}} \right\rfloor; \quad \mathrm{oy}' = \left\lfloor \mathrm{oy} \cdot \frac{\mathrm{fy}'}{\mathrm{fy}} \right\rfloor; \quad \mathrm{sx}' = \left\lceil (\mathrm{sx}+\mathrm{ox}) \cdot \frac{\mathrm{fx}'}{\mathrm{fx}} \right\rceil - \mathrm{ox}'; \quad \mathrm{sy}' = \left\lceil (\mathrm{sy}+\mathrm{oy}) \cdot \frac{\mathrm{fy}'}{\mathrm{fy}} \right\rceil - \mathrm{oy}' \qquad \text{(C-2)}$$

EXAMPLE 2: 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.

EXAMPLE 3: 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.

The above considerations for two dimensional images carry over naturally to images of higher dimensionality, e.g., to the case *n* = 3 where a third coordinate is added to each group of parameters. Specifically, the frame size is then represented by three numbers fx, fy and fz, the offset by ox, oy and oz and the region size by sx, sy and sz. In that case, Equation C-1 extends to:

$$\mathrm{ZOsiz}' = \left\lceil \frac{\mathrm{ZOsiz}}{2^{\mathrm{r}}} \right\rceil \qquad \mathrm{Zsiz}' = \left\lceil \frac{\mathrm{Zsiz}}{2^{\mathrm{r}}} \right\rceil$$

where ZOsiz and Zsiz specify the original image offset and canvas size in the Z direction, respectively. Equation C-2 extends to:

$$oz' = \left\lfloor oz \cdot \left(\frac{fz'}{fz}\right) \right\rfloor \qquad sz' = \left\lceil (sz + oz) \cdot \frac{fz'}{fz} \right\rceil - oz'$$

For images represented by Rec. ITU-T T.809 | ISO/IEC 15444-10 codestreams, ZOsiz and Zsiz are taken from the relevant NSI marker segment.

For images represented by Rec. ITU-T T.801 | ISO/IEC 15444-2 codestreams that involve wavelet-based multiple component transformation, the generated output components of the codestream can be interpreted as the third (Z) dimension of the volumetric image. In this situation, clients can either choose to use the two-dimensional or three-dimensional request syntax to fetch data from the server and, in the case of three-dimensional requests, servers can make their own determination of suitable values for ZOsiz and Zsiz as well as the association between volumetric slices and generated output components of the codestream. For two-dimensional requests, it is up to the client to determine the components that are associated with slices of interest and make the necessary requests; for three-dimensional requests, it is the duty of the server to find the relevant components for the requested image volume. In the latter case, servers are not required to honour the comp and mctres fields, see clauses C.4.8 and C.4.14, and their usage by clients is discouraged in this case.

Notwithstanding the fact that servers can make their own determination of suitable values for ZOsiz and Zsiz and the association between slices and generated output components, when responding to a client request involving three-dimensional request syntax for a volumetric image that uses a wavelet-based multiple-component transformation, they are recommended to make this choice using the following steps.

1. ZOsiz should be taken identical to the minimum of all $Omcc^i$ values in all MCC markers within the codestream identified by the request, see Rec. ITU-T T.801 | ISO/IEC 15444-2. This choice ensures a reasonable definition of the resolution levels in the Z direction compatible to the origin of the wavelet transformation, and eases the extraction of lower-resolution images from the stream.
2. Zsiz should be taken identical to the number of slices identified in Step 3 below.
3. In the case where a Rec. ITU-T T.801 | ISO/IEC 15444-2 conforming file format is available for the target of the request:
   a. All compositing layers of the file that use the codestream that is the target of the request should be identified and each compositing layer in this set should be identified with one slice of the volumetric image. The Z coordinate to be assigned to the first compositing layer in this set is to be ZOsiz, as determined above, after which all following slices are assigned consecutive ascending Z coordinates in the order they appear in the file.
   b. For each compositing layer, the channels that are relevant to the request should be determined using any channel definition box found within the compositing layer. If the layer contains a channel definition box, the relevant channels are those that are associated with a colour via the Asoc field of the channel definition box; otherwise, all channels of the compositing layer are relevant.
   c. For each relevant channel of a compositing layer (slice), the slice should be associated with all generated output components used to provide the data for that channel. For palette mapped images, the component mapping box is also involved in determining this set of associated components.

If no Rec. ITU-T T.801 | ISO/IEC 15444-2 conforming file format is available, servers should use whatever other descriptive data might be available to them, outside the scope of this Recommendation | International Standard, to identify volumetric slices and the set of generated output components associated which each slice, assigning Z = ZOsiz to the first such slice and assigning consecutive ascending Z coordinates to each subsequent slice in the order determined by this descriptive data. The descriptive data, in this case, might be defined in other members of the Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | Standards, or otherwise.

If no Rec. ITU-T T.801 | ISO/IEC 15444-2 conforming file format is available, nor any other descriptive data as detailed above, the following procedure can be used as a last resort to come to a reasonable definition of slices:

a. A codestream is identified as a grey scale volume image if each generated component is reconstructed by exactly one transformation stage, as defined by Rec. ITU-T T.801 | ISO/IEC 15444-2, and if the type of the transformation stage is a wavelet transformation. A codestream is identified as a colour volume image, if each generated component is reconstructed by exactly two transformation stages of which the first one, which is applied to the spatial components of the codestream, is a wavelet transformation, and of which the second one is not a wavelet transformation, but a decorrelation or dependency transformation. All other set-ups cannot be handled by this procedure.

b. For each generated output component g, the MCC marker MCC^i, that describes the wavelet transformation step taken to compute g from the spatial components of the canvas system, is identified; there should be exactly one such marker.

c. If the image is a grey-scale image, the index j is found in the output component collection of that marker such that Wmcc^{ij} equals g; equivalently, j is the output slot in the transformation that generates component g. Then generated component g contributes to the slice with Z = j + Omcc^i. This provides a Z coordinate for the component g based on the ordering of the output of the wavelet transformation step.

d. If the image is a colour image, all intermediate input components of the dependency or decorrelation transformation required to reconstruct the generated output component g are found first. For each such intermediate input component, the Z coordinate for that component is found in the manner described above for grey-scale images. If this Z coordinate differs amongst the intermediate input components involved in the reconstruction of generated output component g, then this procedure fails.

e. If this procedure succeeds in assigning a Z coordinate to each generated output component of the codestream then the Z coordinates are contiguous, ascending from the ZOsiz value found in Step 1, and Zsiz is the number of distinct Z coordinates assigned.

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 available within that codestream. The requested frame size is mapped to a codestream image resolution, following the procedure described in C.4.1, possibly with the addition of coordinate transformations requested via a Codestream Context request field (see C.4.10). A client wishing to control the exact number of samples received for a particular image component might need to increase the requested frame size, as explained in C.4.1. The `round-direction` options defined by this Recommendation | International Standard are described in Table C.1.

Table C.1 – 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.
"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. If there is none, then the smallest available codestream image resolution shall be used. 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 × fy) shall be selected. Where two codestream image resolutions have areas which are equidistant from fx × 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 (see 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 (see C.4.10).

Use of the Offset field is valid 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 can, 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; 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 (see C.4.10). 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 valid only in conjunction with the Frame Size request field.

The codestream image region may be empty, for example if `sx` or `sy` were zero. If empty, then 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 can, 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 Frame size for variable dimension data (fvsiz)

```
fvsiz = "fvsiz" "=" 1#UINT ["," round-direction]

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

This request takes a variable number of arguments. There shall be as many numerical arguments as there are dimensions in the source codestream. Specifically, if the image is a regular two-dimensional image, this request field is equivalent to the `fsiz` field with the first argument defining `fx` and the second defining `fy`. If the source stream represents volumetric data, there shall be three numerical arguments, specifying the view-window extents `fx`, `fy` and `fz`, in that order.

This field is used to identify the resolution associated with the requested view-window. The numerical arguments specify the desired image resolution, one per dimension. 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 available within that codestream. The requested frame size is mapped to a codestream image resolution, following the procedure described in C.4.1, possibly with the addition of coordinate transformations requested via the Codestream Context request field (see C.4.10).

C.4.6 Offset for variable dimension data (rvoff)

```
rvoff = "rvoff" "=" #1UINT
```

This field is used to identify the upper left (front) corner (offset) of the spatial region associated with the requested view-window; if not present, the offset defaults to 0. The actual displacement of a codestream image region from the upper left (front) 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 transformation requested via a Codestream Context request field (see C.4.10). This field takes a variable number of arguments, there shall be as many arguments to the `rvoff` field as there are dimensions in the source stream. Specifically, for regular two-dimensional images exactly, two arguments are required and this field is equivalent to `roff`. For volumetric images, exactly three arguments are required, which define `ox`, `oy` and `oz`.

Use of the Offset field for Variable Dimension Data is valid only in conjunction with the Frame Size request field for Variable Dimension Data. If the view-window 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. In particular, responses of type "jpp-stream" or "jpt-stream" should contain no messages which reference precinct, tile or tile-header data-bins. The server can, 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.4.7 Region size for variable dimension data (rvsiz)

```
rvsiz = "rvsiz" "=" #1UINT
```

This field is used to identify the extent (size) of the spatial region associated with the requested view-window; if not present, the region extends to the lower right (back) corner of the image. The actual extent of the view-window, at the actual resolution level selected by the server, are computed following the procedure described in C.4.1. The view-window need not necessarily be fully contained within the image itself, in which case the server simply takes the intersection between the full image region and the requested view-window.

This field takes a variable number of arguments, and there shall be exactly as many arguments as there are dimensions in the target stream. If the view-window 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. In particular, responses of type "jpp-stream" or "jpt-stream" should contain no messages which reference precinct, tile or tile-header data-bins. The server can, 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 for Variable Dimension Data field. In case the image is a regular two-dimensional image, this request takes two arguments, and is identical to the `rsiz` field. For volumetric images, the three arguments are `sx`, `sy` and `sz`, in this order.

C.4.8 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 (see C.4.10). 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 Rec. ITU-T T.801 | 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.

Usage of the `comps` field in combination with the Frame, Region or Region Offset request field for Variable Dimension Data with three or more numerical arguments on Rec. ITU-T T.801 | ISO/IEC 15444-2 codestreams is discouraged and servers cannot be expected to handle it.

C.4.9 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 behaviour is for servers to associate the request with the single codestream that has identifier 0.

NOTE – The Codestream Context request field (see C.4.10) 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. The client's index of the codestreams of interest is *k* and *k* is a UINT.

C.4.10 Codestream Context (context)

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

context-range = jpxl-context-range / mj2t-context / jpm-context / jpxf-context-range
              / reserved-context

jpm-context = "jpmp" "<" jpm-pages ">" [ jpm-objects ]

jpm-pages = [ jpm-page-collection ":" ] jpm-sampled-range

jpm-objects = "[" jpm-object-range "]"

jpm-page-collection = object-id

jpm-sampled-range = page-object-range [ ":" sampling-factor ]

page-object-range = 1#(object-id [ "-" [ object-id ] ])

jpm-object-range = UINT-RANGE ":" jpm-object-type / UINT-RANGE
                 / ":" jpm-object-type

jpm-object-type = "mask" / "image" / "nostrm"

object-id = UINT / TEXT-LABEL

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 / "<" / ">" / "[" / "]" / "-" / ":" / "+" )

jpxf-context-range = "jpxf" "<" jpx-frame-indices ">" [ "[" jpx-thread "]" ]

jpx-frame-indices = sampled-range

jpx-thread = UINT
```

This field can be used to request codestreams indirectly via "higher level" image entities. This Recommendation | International Standard defines contexts corresponding to JPX compositing layers (a JPX compositing layer might involve one or more codestreams), MJ2 or J2KS video tracks and JPM pages; however, the mechanism is designed for extensibility.

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` can 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 Recommendation | International Standard defines four specific types of `context-range`, which are intended to address the needs of JPX, MJ2, J2KS and JPM 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` can 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 might 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 `fx"`, `fy"`, `sx"`, `sy"`, `ox"` and `oy"` using the expressions in Equation C-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.

First, define the rotated frame size, offset, width and height of the composite image as follows:

$$\begin{bmatrix} \mathrm{ox}^{F}, \mathrm{oy}^{F}, \\ \mathrm{XO}^{F}_{\mathrm{inst}}, \mathrm{YO}^{F}_{\mathrm{inst}} \end{bmatrix} = \begin{bmatrix} (\mathrm{fx} - \mathrm{ox} - \mathrm{sx}), \ (\mathrm{fy} - \mathrm{oy} - \mathrm{sy}), \\ (\mathrm{W}_{\mathrm{comp}} - \mathrm{XO}_{\mathrm{inst}} - \mathrm{Wt}_{\mathrm{inst}}), \ (\mathrm{H}_{\mathrm{comp}} - \mathrm{YO}_{\mathrm{inst}} - \mathrm{Ht}_{\mathrm{inst}}) \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{fx}^{\theta}, \mathrm{fy}^{\theta}, \mathrm{ox}^{\theta}, \mathrm{oy}^{\theta}, \mathrm{W}^{\theta}_{\mathrm{comp}}, \mathrm{H}^{\theta}_{\mathrm{comp}}, \\ \mathrm{XO}^{\theta}_{\mathrm{inst}}, \mathrm{YO}^{\theta}_{\mathrm{inst}}, \mathrm{Wt}^{\theta}_{\mathrm{inst}}, \mathrm{Ht}^{\theta}_{\mathrm{inst}} \end{bmatrix} = \begin{cases} \begin{bmatrix} \mathrm{fx}, \mathrm{fy}, \mathrm{ox}, \mathrm{oy}, \mathrm{W}_{\mathrm{comp}}, \mathrm{H}_{\mathrm{comp}}, \\ \mathrm{XO}_{\mathrm{inst}}, \mathrm{YO}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 0^{\circ} \,|\, \mathrm{NoFlip} \\ \begin{bmatrix} \mathrm{fy}, \mathrm{fx}, \mathrm{oy}, \mathrm{ox}^{F}, \mathrm{H}_{\mathrm{comp}}, \mathrm{W}_{\mathrm{comp}}, \\ \mathrm{YO}_{\mathrm{inst}}, \mathrm{XO}^{F}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 90^{\circ} \,|\, \mathrm{NoFlip} \\ \begin{bmatrix} \mathrm{fx}, \mathrm{fy}, \mathrm{ox}^{F}, \mathrm{oy}^{F}, \mathrm{W}_{\mathrm{comp}}, \mathrm{H}_{\mathrm{comp}}, \\ \mathrm{XO}^{F}_{\mathrm{inst}}, \mathrm{YO}^{F}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 180^{\circ} \,|\, \mathrm{NoFlip} \\ \begin{bmatrix} \mathrm{fy}, \mathrm{fx}, \mathrm{oy}^{F}, \mathrm{ox}, \mathrm{H}_{\mathrm{comp}}, \mathrm{W}_{\mathrm{comp}}, \\ \mathrm{YO}^{F}_{\mathrm{inst}}, \mathrm{XO}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 270^{\circ} \,|\, \mathrm{NoFlip} \\ \begin{bmatrix} \mathrm{fx}, \mathrm{fy}, \mathrm{ox}^{F}, \mathrm{oy}, \mathrm{W}_{\mathrm{comp}}, \mathrm{H}_{\mathrm{comp}}, \\ \mathrm{XO}^{F}_{\mathrm{inst}}, \mathrm{YO}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 0^{\circ} \,|\, \mathrm{Flip} \\ \begin{bmatrix} \mathrm{fy}, \mathrm{fx}, \mathrm{oy}, \mathrm{ox}, \mathrm{H}_{\mathrm{comp}}, \mathrm{W}_{\mathrm{comp}}, \\ \mathrm{YO}_{\mathrm{inst}}, \mathrm{XO}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 90^{\circ} \,|\, \mathrm{Flip} \\ \begin{bmatrix} \mathrm{fx}, \mathrm{fy}, \mathrm{ox}, \mathrm{oy}^{F}, \mathrm{W}_{\mathrm{comp}}, \mathrm{H}_{\mathrm{comp}}, \\ \mathrm{XO}_{\mathrm{inst}}, \mathrm{YO}^{F}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 180^{\circ} \,|\, \mathrm{Flip} \\ \begin{bmatrix} \mathrm{fy}, \mathrm{fx}, \mathrm{oy}^{F}, \mathrm{ox}^{F}, \mathrm{H}_{\mathrm{comp}}, \mathrm{W}_{\mathrm{comp}}, \\ \mathrm{YO}^{F}_{\mathrm{inst}}, \mathrm{XO}^{F}_{\mathrm{inst}}, \mathrm{Ht}_{\mathrm{inst}}, \mathrm{Wt}_{\mathrm{inst}} \end{bmatrix} & \text{if } \mathrm{R}_{\mathrm{inst}} = 270^{\circ} \,|\, \mathrm{Flip} \end{cases} \quad \text{(C-3a)}$$

In the above, W_{comp} and H_{comp} are the width and height of the composited image, specified in the composition box; Wt_{inst} and Ht_{inst} are the composited width and height as determined by the compositing instruction; XO_{inst} and YO_{inst} are the horizontal and vertical compositing offsets as determined by the compositing instruction; Ws_{inst} and Hs_{inst} are the width and height of the potentially cropped compositing layer as determined by the compositing instruction; XC_{inst} and YC_{inst} are the horizontal and vertical compositing layer cropping offsets as determined by the compositing instruction; and R_{inst} is derived from the ROT field of the compositing instruction, if any. If the compositing instruction contains no ROT field or the ROT field is 0, R_{inst}=0º|NoFlip. Otherwise, the rotation angle for R_{inst} (expressed in degrees clockwise) is obtained from the least significant 3 bits of the ROT field defined in Rec. ITU-T T.801 | ISO/IEC 15444-2, while the Flip|NoFlip status for R_{inst} is set to Flip if bit 4 of the ROT field is non-zero and NoFlip otherwise.

Then, define the modified frame size `fx″`, `fy″` as follows:

$$\mathrm{fx''} = \left\lceil \mathrm{fx}^{\theta} \cdot \frac{\mathrm{XR_{reg}}}{\mathrm{XS_{reg}}} \cdot \frac{\mathrm{Wt^{\theta}_{inst}}}{\mathrm{Ws_{inst}}} \cdot \frac{\mathrm{W_{cod}}}{\mathrm{W^{\theta}_{comp}}} \right\rceil; \quad \mathrm{fy''} = \left\lceil \mathrm{fy}^{\theta} \cdot \frac{\mathrm{YR_{reg}}}{\mathrm{YS_{reg}}} \cdot \frac{\mathrm{Ht^{\theta}_{inst}}}{\mathrm{Hs_{inst}}} \cdot \frac{\mathrm{H_{cod}}}{\mathrm{H^{\theta}_{comp}}} \right\rceil \qquad \text{(C-3b)}$$

To compute the modified region, first define the clipped region edges:

$$\begin{aligned} x_{\min} &= \left\lfloor \mathrm{XO^{\theta}_{inst}} \cdot \frac{\mathrm{fx}^{\theta}}{\mathrm{W^{\theta}_{comp}}} \right\rfloor; \quad y_{\min} = \left\lfloor \mathrm{YO^{\theta}_{inst}} \cdot \frac{\mathrm{fy}^{\theta}}{\mathrm{H^{\theta}_{comp}}} \right\rfloor \\ x_{\lim} &= \left\lceil \left(\mathrm{XO^{\theta}_{inst}} + \mathrm{Wt^{\theta}_{inst}}\right) \cdot \frac{\mathrm{fx}^{\theta}}{\mathrm{W^{\theta}_{comp}}} \right\rceil; \quad y_{\lim} = \left\lceil \left(\mathrm{YO^{\theta}_{inst}} + \mathrm{Ht^{\theta}_{inst}}\right) \cdot \frac{\mathrm{fy}^{\theta}}{\mathrm{H^{\theta}_{comp}}} \right\rceil \end{aligned} \qquad \text{(C-3c)}$$

The modified region size `sx″` and `sy″` and region offsets `ox″` and `oy″` are then given as:

$$\begin{aligned} \mathrm{sx''} &= \min\left\{\left(\mathrm{ox}^{\theta} + \mathrm{sx}^{\theta}\right), x_{\lim}\right\} - \max\left\{\mathrm{ox}^{\theta}, x_{\min}\right\} \\ \mathrm{sy''} &= \min\left\{\left(\mathrm{oy}^{\theta} + \mathrm{sy}^{\theta}\right), y_{\lim}\right\} - \max\left\{\mathrm{oy}^{\theta}, y_{\min}\right\} \\ \mathrm{ox''} &= \max\left\{\mathrm{ox}^{\theta}, x_{\min}\right\} - \left\lceil \left(\mathrm{XO^{\theta}_{inst}} - \left(\mathrm{XC_{inst}} - \frac{\mathrm{XO_{reg}}}{\mathrm{XS_{reg}}}\right) \cdot \frac{\mathrm{Wt^{\theta}_{inst}}}{\mathrm{Ws_{inst}}}\right) \cdot \frac{\mathrm{fx}^{\theta}}{\mathrm{W^{\theta}_{comp}}} \right\rceil \\ \mathrm{oy''} &= \max\left\{\mathrm{oy}^{\theta}, y_{\min}\right\} - \left\lceil \left(\mathrm{YO^{\theta}_{inst}} - \left(\mathrm{YC_{inst}} - \frac{\mathrm{YO_{reg}}}{\mathrm{YS_{reg}}}\right) \cdot \frac{\mathrm{Ht^{\theta}_{inst}}}{\mathrm{Hs_{inst}}}\right) \cdot \frac{\mathrm{fy}^{\theta}}{\mathrm{H^{\theta}_{comp}}} \right\rceil \end{aligned} \qquad \text{(C-3d)}$$

The modified view-window region, defined by `sx"`, `sy"`, `ox"` and `oy"`, can potentially lie slightly to the left or above the origin. That is, `ox"` and/or `oy"` may be negative. Any portion of the view-window region which lies to the left or above the origin should be ignored when determining the codestream image region following the procedure described in C.4.1.

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 C-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.

$$\begin{aligned} &\mathtt{fx"} = \left\lceil \mathtt{fx} \cdot \frac{\mathtt{XR_{reg}}}{\mathtt{XS_{reg}}} \cdot \frac{\mathtt{W_{cod}}}{\mathtt{W_{reg}}} \right\rceil; \quad \mathtt{fy"} = \left\lceil \mathtt{fy} \cdot \frac{\mathtt{YR_{reg}}}{\mathtt{YS_{reg}}} \cdot \frac{\mathtt{H_{cod}}}{\mathtt{H_{reg}}} \right\rceil \\ &\mathtt{ox"} = \mathtt{ox} - \left\lceil \frac{\mathtt{XO_{reg}}}{\mathtt{XS_{reg}}} \cdot \frac{\mathtt{fx}}{\mathtt{W_{reg}}} \right\rceil; \quad \mathtt{oy"} = \mathtt{oy} - \left\lceil \frac{\mathtt{YO_{reg}}}{\mathtt{YS_{reg}}} \cdot \frac{\mathtt{fy}}{\mathtt{H_{reg}}} \right\rceil \\ &\mathtt{sx"} = \mathtt{sx}; \quad \mathtt{sy"} = \mathtt{sy} \end{aligned} \qquad \text{(C-4)}$$

The second type of `context-range` described by this Recommendation | International Standard, `mj2t-context`, allows clients to request specific tracks from an MJ2 or J2KS file. The `mj2-track` identifier shall be a strictly positive integer, since 1 is the smallest allowable track identifier permitted within an MJ2 or J2KS file. If an `mj2-track` identifier includes the optional "+now" suffix, the `mj2t-context` consists of all codestreams belonging to the MJ2 or J2KS 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 can 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 or J2KS video track.

An `mj2t-context` can 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 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 provides a modified `mj2t-context` string in its Codestream Context response header.

NOTE 1 – The use of the Codestream Context request field together with the Codestream request field can 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.

NOTE 2 – The expressions in Equation C-4 can 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 C-3 when the limits on `sx"`, `sy"`, `ox"` and `oy"` are not bounded by x_{lim}, x_{min}, y_{lim}, y_{min}.

The third type of context-range described by this Recommendation | International Standard, `jpm-context`, allows clients to request specific layout objects from a JPM file. The simplest usage allows a request to be made for all the items needed to render a single page. More complex usage allows only some of the layout objects or only one type of object to be requested. The `jpm-context` always contains a request for specific pages, it can also contain a specification for page collections, a list of layout objects, and object types.

If `jpm-context` has no `jpm-page-collection` item then the main page collection is assumed. If `TEXT-LABEL` is specified in the `jpm-page-collection` item it shall correspond to a label of a page collection box in the target JPM file. If `UINT` is specified in the `jpm-page-collection` item it indicates the page collection box in that position in file, where page collection boxes are numbered from 0.

A range of pages is a required part of the `jpm-context`. The page range could be "0-" which would specify all the pages in the page collection. Pages are numbered by following the page collections and pages in the JPM file, and assigning the number 0 to the first page in a depth first tree walk. The root of the tree is given by the jpm-page-collection item or the main page collection if no jpm-page-collection is part of the request. Loops in the page collection tree should be detected and an error condition returned.

If a "sampling-factor" is used as part of the jpm-sampled-range, the client desires pages starting with the first number in each range, and less than or equal to the last number in the range, and at all integer multiples of the sampling-factor plus the initial page number. Thus two sampling ranges it is possible to request even and odd number pages using a sampling-factor of 2, by starting each range with an even or odd number.

If the `jpm-context` has no `jpm-object-range` item then it is considered to be "1-" which corresponds to all objects on the page except the thumbnail. If the thumbnail image for a page is needed then the `jpm-object-range` item shall include zero. The `jpm-object-range` indicates which of the layout objects on all pages in the `jpm-page-range` are requested.

If the `jpm-context` has no `jpm-object-type` then all types are used. If the `jpm-object-type` is "mask" only mask objects are of interest for the request. If the `jpm-object-type` is "image" only image objects are of interest. If the `jpm-object-type` is "nostrm" then boxes for both mask and image are of interest.

If the `jpm-context` parameter appears in a request without a Frame Size request (fsiz) then the Frame Size values fx and fy are set to the page width and page height. If the `e`parameter appears in a request without a Region Size request (rsiz) then the Region Size values sx and sy are set to the frame size values fx and fy (after fx and fy have been set to the page width and height if necessary).

When the `jpm-context` parameter is used, the requested corresponds to the view-window applied to each page independently. The Frame Size values `fx` and `fy` are mapped to the page width and height as specified by the `Pwidth` and `Pheight` elements of the Page Header Box of Rec. ITU-T T.805 | ISO/IEC 15444-6.

A layout object within a page is considered part of the request if and only if all of the following are true:

```
ox' <= LHoff + LWidth      ox' + sx' >= LHoff
oy' <= LVoff + LHeight     oy' + sy' >= LVoff
```

where:

```
ox' = ox * Pwidth / fx
oy' = oy * Pheight / fy
sx' = sx * Pwidth / fx
```

```
sy' = sy * Pheight / fy
```

and `fx`, `fy`, `ox`, `oy`, `sx`, and `sy` are from the view window requests, `LHoff`, `LVoff`, `LHeight`, and `LWidth` are from the Layout Object Header Box of Rec. ITU-T T.805 | ISO/IEC 15444-6.

Layout object 0 is reserved for a thumbnail image of the page, it should be considered part of the request regardless of the view-window if and if 0 is included in `jpm-object-range`.

The client is considered to have requested any codestream associated with the mask or image which intersects the view-window unless jpm-object-type is "nostrm". If the codestream is not compressed with JPEG 2000 then the request is for the complete codestream. If the codestream is compressed with JPEG 2000 then an equivalent view-window can be determined for the specific codestream by mapping the request window on the page to the request window on the object as follows:

```
fx' = fx * Lwidth / Pwidth
fy' = fy * Lheight / Pheight

ox' = MAX( ox - LHoff * fx / Pwidth , 0)
oy' = MAX( oy - LVoff * fy / Pheight, 0)

sx' = MIN ( ox + sx - LHoff * fx / Pwidth, Lwidth * fx / Pwidth) - ox'
sy' = MIN ( oy + sy - LVoff * fy /Pheight, Lheight * fy / Pheight) - oy'
```

Note that it might be necessary to issue a frame-size request with values larger than the width and height of the page in order to obtain a full resolution JPEG 2000 codestream if the JPEG 2000 file contains data at a higher resolution than the page. Alternatively, the client could determine the codestream number and issue a request directly on that codestream with a view-window chosen appropriately.

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 might be described in the Movie Header box.

EXAMPLE 3: "context=jpmp<0-10,21-30:2>[1-3:mask]"
In this case, the server is requested to return all data corresponding to mask objects in the first three layout objects on the pages 0, 2, 4, 6, 8, 10, 21, 23, 25, 27, and 29. This request includes all boxes necessary to render the desired region, e.g. Page Boxes, Layout Object Boxes, as well as any codestreams referenced by those objects.

For JPM files, the following metadata elements shall be considered to be requested along with the view-window:

– JP2 signature ("jP")
– File type ("ftyp")
– Compound Image Header ("mhdr")
– Page Collection box ("pcol")
– Page Table box ("pagt")
– Page box ("page")
– For pages that are relevant with the view-window request:
  • Page Header box ("phdr")
  • Layout Object box ("lobj")
  • Layout Object Header box ("lhdr")
  • Object box ("objc")
  • Object Header box ("ohdr")
  • Object Scale box ("scal")
  • Base Colour box ("bclr")

The above considerations, especially Equations C-3 and C-4, valid for two-dimensional image data only. They are extended naturally to higher dimensions by duplicating the computations for each additional dimension. Usage of the codestream context field is discouraged if the target of the request contains codestreams with differing numbers of dimensions, and servers cannot be expected to handle this case.

A `jpxf-context-range` can be used to compactly identify a range of compositing layers and coordinate remapping transformations which could alternately be identified via a `jpxl-context-range`. The equivalent jpx-layers and jpxl-geometry values can be obtained by expanding composited frames into their constituent JPX compositing layers and compositing instructions in the manner described below.

If the logical target does not contain a JPX Composition box, the server shall ignore any `jpxf-context-range`. Otherwise, the instructions found within the JPX Composition box together describe a sequence of composited frames, as described in Rec. ITU-T T.801 | ISO/IEC 15444-2. These composited frames are numbered $f = 0, 1, \ldots F_{comp}–1$ and are considered to belong to a base presentation thread t = 0. If the logical target also contains Composition layer extensions ("jplx") boxes, these boxes might contribute additional presentation threads. As explained in Rec. ITU-T T.801 | ISO/IEC 15444-2, a Compositing Layer Extensions box contributes Tjclx presentation threads, each of which has the same number of composited frames, Fjclx, where the values of Tjclx and Fjclx for each Compositing Layer Extensions box are specified by its Compositing Layer Extensions Info sub-box. Together, the collection of all Compositing Layer Extensions boxes in the logical target defines T global presentation threads, where T is the maximum of the associated **Tjclx** values. For each t in the range 1 through T, global presentation thread t consists of the F_{comp} composited frames from the Composition box, followed by the **Fjclx** frames defined by compositing group g = min{t, **Tjclx**} of each successive Compositing Layer Extensions box for which **Tjclx** is non-zero.

If no jpx-thread value is supplied, or jpx-thread is 0, the `jpxf-context-range` includes only those composited frames contributed by the Composition box whose indices f match jpx-frame-indices; there are at most F_{comp} of these. Otherwise, the `jpxf-context-range` includes all composited frames from global presentation thread t = min{T, jpx-thread} whose indices f match jpx-frame-indices.

C.4.11 Sampling Rate (srate)

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

streams-per-second = UFLOAT
```

If this field is supplied, the codestreams which belong to the view-window are obtained by subsampling those mentioned by the Codestream request field, in addition to those expanded from context-range values in the Codestream Context request field (see C.4.10), 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 or J2KS file formats).

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 Recommendation | 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 and the Codestream Context request field, but this request field can nonetheless affect the interpretation of a Delivery Rate request field, if present.

C.4.12 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 can come from several places; it can be defined within an ROI description box within the logical target, or it can 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 can 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 can 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 ROIs.

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.4.13 Layers (layers)

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

This field can 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 might be necessary for the server to return data beyond the boundary of the requested layer for JPT-stream requests only.

C.4.14 Multi-component transformation (MCT) Resolution Value

```
mctres = "mctres" "=" UINT
```

This field specifies the desired multi-component transformation resolution level. This field is only applicable if for all tile-components, exactly one of the multi-component transformations that are applied on this tile-component (and iteratively on the resulting intermediate components to create generated components) is a multi-component wavelet-transformation. It shall not be used otherwise. If this field is not present, it will be assumed that the full resolution representation of the image data is desired. The full number of resolution levels is one more than the number of wavelet transform levels N_L in the multi-component transformation, given by $Tmcc^i$ (see Rec. ITU-T T.801 | ISO/IEC 15444-2). For full resolution, this field should be set to 1. For half resolution, the field should be set to 2, for quarter resolution, the field should be set to 3, etc. If the value of mctres exceeds $N_L + 1$ for one tile or codestream, the lowest available resolution of that tile or codestream shall be transmitted. The same value of mctres shall apply simultaneously to all multi-component wavelet transformations found in the codestream(s).

Usage of the `mctres` field in combination with the Frame, Region or Region Offset request field for Variable Dimension Data with three or more numerical arguments on Rec. ITU-T T.801 | ISO/IEC 15444-2 codestreams is discouraged and servers cannot be expected to handle it.

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 is present, the view-window is associated with at least one codestream, as mentioned in C.4.9. 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 might 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 subclause identifies a minimal set of metadata which servers shall regard as being implicitly requested along with the view-window.

NOTE – The list of boxes defined in this clause is not exhaustive. Additional boxes might be required to decode the requested view window within the logical target correctly.

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

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

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

d) The entire contents of each of the following boxes, wherever these boxes are found within one of the superboxes mentioned above:
   1) Image Header ("ihdr");
   2) Bits per Component ("bpcc");
   3) Palette ("pclr");
   4) Component Mapping ("cmap");
   5) Channel Definition ("cdef");
   6) Resolution ("res ");
   7) Codestream Registration ("creg");
   8) Opacity ("opct").

e) For JP2 files, JP2 compatible files and JPX files, one or more Colourspace Description boxes ("colr") associated with each codestream or JPX compositing layer requested via the Codestream Context request field, as follows:
   1) If the server is able to determine exactly which box is preferred, the server should send only that box, even if it means not sending the first box for JP2 or JP2 compatible files (for example if the second box is Any ICC and the colorspace preferences specify that the client prefers Any ICC). If the server is not able to determine exactly which box is preferred, it should send the entire first Colourspace Description box.
   2) For all boxes not sent, the server should send a portion of the box contents so the client can determine if it later wants to request another colourspace specification.
      - For enumerated boxes, the server should send at least the first 7 bytes of the box contents (up to at least the EnumCS field).
      - For vendor-defined colourspace boxes, the server should send at least the first 19 bytes of the box contents (up to at least the VCLR field).
      - For Restricted and Any ICC colourspace boxes, the server should send at least the first 3 bytes of the box contents (at least the METH, APPROX and PREC fields).

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 can depend upon the particular way in which the logical target has been partitioned into metadata-bins. A discussion of these issues can be found in A.3.6.2.

For MJ2 files, the following metadata elements shall be considered to be requested along with the view-window:

- JP2 signature ("jP")
- File type ("ftyp")
- "mvhd"
- For tracks that are relevant with the view-window request:
  – "tkhd"
  – edts[0]. Only the TBox field is useful, and a placeholder signals that no access is provided to the original content of the box.
  – "mdhd"
  – "hdlr"
  – "vmhd" if present in the original MJ2 file.
  – "stsd"
  – "stts"
  – either:
    - a placeholder for "stco" or "stco64" (depending on which of them is present in the original MJ2 file) indicating that the content of the box is provided by one or more incremental codestreams;
    - or the entire "stsc", "stsz" and "stco" or "stco64" boxes.

Similar considerations apply to J2KS files.

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 = "!!"
```

This field specifies what metadata is desired in response to a request, in addition to any metadata required for the client to decode or interpret the requested image data as defined by C.5.1. The purpose of this request is to allow the client to request selected parts of the contents and the layout of the metadata encoded in the JP2, JPH and JPX file formats a server did not choose to transmit according to C.5.1.

The value string in this request field is a list of independent requests; however, the server can handle the requests as a group, and there can be overlap between the requests. It is then sufficient (but not necessary) that the server sends the requested data only once.

The way the server decides to break up the initial stream into bins is irrelevant for defining the target of the request except that the `root-bin` field can be used to limit a request to parts of the file structure, once a client has identified the layout. Once a request is confined to a specific bin, the way that bin is broken up into more bins – or if it is broken up at all – is irrelevant for the client and the way that data is addressed within the request.

However, data that a server returns upon a request will, in general, depend on that layout because the division of the logical target into metadata-bins might force the server to return additional data, including the contents or headers of some other, potentially unrequested boxes. All a server has to ensure is that **at least** the requested data is contained, and that **enough** data is returned to allow a client to parse it. Examples when additional data needs to be returned are given below in C.5.2.9. The following text uses the wording "request" to point out which data is **desired** by the client, which might be a sub-set of the data actually returned by the server due to reasons pointed out in C.5.2.9.

Example

For better illustration, examples in the following subclauses all refer to the following segment of a JPX file, see Rec. ITU-T T.801 | ISO/IEC 15444-2 for the definition of the boxes used here. The labels on the right-hand side have been added for later reference:

Content	Label
association box header ('asoc')	A
number list box header ('nlst')	B
number list box content	C
association box header ('asoc')	D
ROI description box header ('roid')	E
ROI description box content	F
association box header ('asoc')	G
label box header ('lbl\040')	H
label box content	I
XML box header ('xml\040')	J
XML box content	K

The sub-box structure of the above example is indicated by indention, e.g., items `H` to `K` establish the contents of the superbox at label `G`.

C.5.2.2 root-bin

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 meta-data bin 0. The request pertains only to data within or referenced by that particular data-bin.

Example

If the server decided to place the contents of association box 'A' in the above example into a separate bin with bin id #3, the association box header 'A' would be encoded in a placeholder box, and items 'B' to 'K' would establish bin #3. In that case, a root-bin field of 3 would limit the scan to items 'B' to 'K' only. Specifically, `metareq=[roid]R3` would request items 'E' and 'F' from the server and no other data outside of this example (but see also clauses C.5.2.3 and C.5.2.9 for additional data outside of the request potentially returned by the server along).

An alternative layout might be to include items 'B' to 'G' in bin #3 as above, but in addition place items 'H' to 'K' into the separate bin #4. Thus 'G' would be represented by a placeholder box in bin #3 and 'H' to 'K' would be part of bin #4. A root-bin field of 3 would still scan the items 'H' to 'K' because they are referenced by a placeholder in bin #3 and the way how bin #3 is broken up into sub-bins is irrelevant to the request. Thus, even though the server response would be different, the items identified by the request remain the same.

A root-bin field of 0 imposes no further restriction on the request each item, box or superbox, is somehow reachable from the metadata-bin #0. Whether placeholder boxes are used or not is completely irrelevant. Thus, `metareq=[roid]` would request all ROI description boxes from the server, and thus also include items 'E' and 'F' along with all other ROI description boxes available.

C.5.2.3 max-depth

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.

Example

If items 'B' to 'K' establish the contents of metadata-bin #3 as in the example for C.5.2.1, the root-bin field is set to 3 and max-depth is set to 0, then the request is limited to items 'B' to 'D'. If max-depth is set to 1, the request is limited to items 'B' to 'G'.

The request `metareq=[roid]R3D0` would therefore not request any data from the server because the only ROI description box within the specified bin is one level below the start of bin #3. The request `metareq=[asoc]R3D0` would request the association box starting at label 'D' and its contents, items 'E' to 'K'. This request is identical to `metareq=[asoc]R3D3` because, even though the latter example also requests the association box starting at label 'G', this box is part of the box starting at label 'D' and is thus included in the former request anyhow.

C.5.2.4 I-box-prop

The `req-box-prop` portion of the request specifies a list of box types that are of interest to the client. The special string "`*`" can be substituted for the box type, in which case all box types are implied. Thus, this field confines the request to apply only to the specific box type (or types) listed and instructs the server to deliver the box header and box contents of all matching boxes within all additional constraints.

The contents of a superbox is defined by its complete sub-box hierarchy. This implies that in case superboxes match the box type, the complete sub-box hierarchy of the matching superbox is requested, regardless of the max-depth field.

Example

Consider again the example layout of C.5.2. Then, a `req-box-prop` of type `'asoc'` would include all items 'A' to 'K' in the request because they establish the content of the matching box defined at label 'A'. Note that, once the association box at label 'A' has been identified to match the request, the depth limit does not limit the delivery of its contents. A `req-box-prop` of type `'lbl\040'` would only include items 'H' and 'I', along with all other label boxes, provided they match all other specifications of the request, e.g., are contained in the addressed root bin above the depth-limit.

The request `metareq=[*]R3D0` instructs the server to return the entire contents of all boxes it finds in the contents of bin 3, and thus requests items 'B' to 'K'. While a restriction on the desired depth has been specified, the server shall ignore that restriction because items 'E' to 'K' are part of the box starting at label 'D' and no other constraints apply.

C.5.2.5 limit

The limit attribute optionally following the `req-box-prop` field further confines the type of request, and how many bytes of the box contents identified by the `req-box-prop` field the client is interested in. The limit parameter takes the form of a colon followed by either an unsigned integer (the limit value) or the character "`r`". The same limit value applies to all boxes that match the `req-box-prop` of which it is an attribute. If it is not present, the client is requesting all matching boxes entirely.

If the limit field is an integer *n* greater than zero, then the server is requested to return the unlimited box header and only the first *n* bytes of the box content of the matching boxes. The byte count is here defined to count the data as it appeared in the original file before it was broken up into bins.

Furthermore, if `req-box-prop` matches any superboxes, the contents of a superbox is to be understood as the complete and unlimited sequence of box headers and box contents contained within that superbox, and the byte limit by that also counts the box headers of all boxes contained within the matching super box. It might thus happen that the limit field instructs the server to deliver only parts of a (sub-)box header even though the full header of the matching box itself is always included. However, using the limit field in this way is discouraged and should be avoided.

If the limit field is zero, only the box headers of the matching boxes are requested.

If the limit value is "`r`", then the server is requested to deliver the minimum data required to reconstruct the box-headers of all matching boxes, as well as the minimum data to reconstruct the box-headers of all of its descendant sub-boxes up to the maximum depth specified, regardless of their box-type. As an exception, `max-depth` does apply for the limit value "`r`" to limit the contents of superboxes, which makes this type of request special as far as the interpretation of `max-depth` is concerned.

Example

Consider again a file layout as in C.5.2 above with items '`B`' to '`K`' in data bin #3 and the metadata request `metareq=[asoc:8]R3D1`. By that, the client requires the box header and the first eight bytes of every association box found in bin #3 not deeper than one level from bin #3. In the example at hand, this requests the item 'D', eight bytes from item '`E`', namely the part of the first sub-box of '`D`' that fits into the limit because it establishes the contents of '`D`', the item '`G`' because it is exactly one level below the first item '`B`' of the bin, and eight bytes of the box content contained in the superbox starting at 'G', that is the first eight bytes of the box header '`H`'. Should the box headers '`E`' and '`H`' not fit into eight bytes, they might get truncated. This is why usage of the numerical limit field on superboxes is discouraged.

Consider the request `metareq=[roid:1]R3D1`. This will request the box header of the ROI description box at label 'E' one level below the start of the bin, and in addition the first byte of its contents at point '`F`', which happens to be the number of regions encoded in the box (see Rec. ITU-T T.801 | ISO/IEC 15444-2). If the example would contain a ROI description box at a deeper level, it would not be requested here due to the depth limit.

The request `metareq=[asoc]R3D0` does not contain a limit, and thus requests the complete body of any association box found at the box level of metadata-bin #3. Even though the association box at label '`G`' is outside the depth-limit, it is still requested because it is contained in the association box started at point '`D`', and by that items '`D`' to '`K`' are transmitted completely.

The "`r`" limit is in effect a request for a skeleton of a portion of the box hierarchy because it only supplies the minimum data, namely the box headers, to reconstruct the structure of the boxes. The request `metareq=[asoc:r]R3D1` thus requests the items at label '`D`', '`E`' and '`G`', but not '`H`' and '`J`' because they are outside the depth-limit. Item 'A' is not part of bin #3 in the example set-up, and is thus neither requested.

The difference between the limit "`0`" and the limit "`r`" is that the former only delivers the box header of all matching boxes, but not necessarily their depending sub-boxes. The "`r`" limit, however, extends the request to the box-headers of the sub-structure of the matching box up to the depth-limit.

C.5.2.6 metareq-qualifier

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. Thus, the "metareq-qualifier" defines an additional constraint on the boxes besides the box-type. 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.

The contexts "`g`", "`s`" and "`w`" are mutually exclusive, but their union is generally smaller than the catch-call context "`a`". It is at the discretion of the server to decide which box falls into which context, and no classification of box types is defined in this Recommendation | International Standard.

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, MJ2 or J2KS video tracks or JPM pages), 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).
NOTE – This metareq context is suitable for requests that wish to interrogate the file structure independently of the view-window.	

C.5.2.7 priority

If the "`priority`" flag is specified, then the client is requesting that the data collected by the meta-data request has to be transmitted with higher priority (i.e., upfront) than the image data described by other fields of the same request.

C.5.2.8 metadata-only

If "`metadata-only`" is specified at the end of the meta-data request field, the client is requesting that the server's response consists only of meta-data, 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. This field also disables the request of the silently implied meta-data defined by C.5.1.

C.5.2.9 Implications of layout-constraints

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.

To make the delivered data parseable to a client, all data from the start of the bin up to the last byte required to satisfy the request has to be known by the client, and thus has to be transmitted provided it is not already within the cache of the client. In addition, should any data that matches the request be relocated into additional bins by means of placeholder boxes, the complete placeholder box and the bytes of the bin referenced by the placeholder box has to be included in the request. Byte counts, as used by the limit attribute, always count bytes **as found in the original stream** and not as it was broken up by the server. This means that the number of bytes actually being transmitted back to the client might be different from the number of bytes implied by the byte-limit, because not only placeholder boxes have to be transmitted, but also the data in front of the requested bytes within the bin the bytes are located in might have to be included to make the resulting stream parseable.

Regardless of the box specifications provided via the metareq request field, the server may send other data, either because it has determined that the other data is required for the client to decode.

Example

Consider again the data as found at the beginning of C.5.2 and assume the server decided to place all of the data into metadata-bin #3 without making use of any (additional) placeholder boxes. Also assume that the cache of the client is empty. Then the metadata-request `metareq=[xml\040:r]R3` is requesting only the box header of the XML box at label '`J`'. However, since the bin is not broken up into more bins, all bytes in front of item '`J`' are also required by the client to parse this data successfully and to identify the transmitted data as a box header, and thus the server is required to send all data from '`B`' to '`J`'.

As the above example suggests, not using placeholders might be considerably inefficient for some requests. The following alternative layout at the server side provides a more efficient access to the same data:

The association boxes at '`D`' and '`G`' are broken up into separate bins with the bin-ids #4 and #5, respectively. Then for the very same request, the server would have to transmit the placeholder box for item '`D`' in bin #3, the placeholder box for item 'G' in bin #4 and the requested box header 'J' now located at the start of bin #5. Note that the request automatically pertains to bins #4 and #5, since they are referenced by placeholders in bins #3 and #4 respectively. Depending on the size of the remaining boxes, this layout might be considerably more efficient.

C.5.2.10 Special considerations for cross-reference boxes

If any cross-reference boxes are identified by a metadata request, the server shall also include in its response such additional metadata as might be required for the client to determine the metadata-bins, if any, which contain the original file byte contents that are referenced by such cross-reference boxes.

If a cross-reference box has a streaming equivalent placeholder, the placeholder box itself provides the identity of a metadata-bin which contains the original cross-referenced content. Otherwise, the server is required to send at least the box header (or corresponding placeholder boxes) for every box in the original file which contains data referenced by the cross-reference box.

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, in units of bytes, that the client requests from the server. For JPP and JPT image return types, the limit includes payload and VBAS headers. The EOR message (header and body, see Annex D) does not contribute to the limit.

If the `len` field is 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). The client should use `len=0` if it requires response headers only and no entity body (see Annex F). Nevertheless, transport protocols that require framing of responses require an EOR message (see Annex G).

C.6.2 Quality (quality)

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

This field can 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.2.16). 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 might 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.

If a server supports this request and two different clients make identical requests to the same target having the same quality value, e.g., "`quality=80`", the server should have a consistent implementation policy in returning data from 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 server supports aligned responses and 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. Servers that do not support data alignment but receive an alignment request with the value "yes" shall indicate this by the Alignment Response defined in D.2.24.

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.

NOTE – 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 might be necessary if a prevailing byte limit prevents the delivery of complete packets.

Table C.3 – 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")
```

This field is used to indicate whether the server shall complete a response to the previous request. 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".

The combination of "wait = yes" with "cclose=*" is not recommended. If this situation is encountered, the server can decide which of the two takes priority.

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 *( ";" parameter )

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

parameter = attribute "=" value

attribute = TOKEN

value = TOKEN
```

This field is used to indicate the type (or types) of the requested response 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.4 – Valid image return types

Type	Interpretation
"jpp-stream"	A JPP-stream as defined in Annex A. "jpp-stream" can 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.2.2)
"jpt-stream"	A JPT-stream as defined in Annex A. "jpt-stream" can 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.2.2)
"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.

NOTE – Other image media types (e.g., jp2, jph, jpeg, tiff, png), if available, can be provided by a server as a transcoding service with JPIP functionality.

For the raw codestream return type, the response data should consist of the requested entity in full. Therefore, many of the other possible client request fields would have no meaning and would be ignored by a server.

C.7.4 Delivery Rate (drate)

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

rate-factor = UFLOAT
```

This field is used to specify the delivery rate of various codestreams. 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 subsampled 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 or J2KS file, the timing information is provided by that file. The MJ2 or J2KS 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 might depend upon the `view-window-pref` value supplied in a Client Preferences request field (see C.10.2). 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.7.5 Send To (Sendto)

```
Sendto = "sendto" "=" host ":" port ";" mbw ";" bpc

host = token

port = UINT

bpc  = UINT
```

If this request field is present, the server is requested to deliver response data for this request as UDP datagrams to the supplied `host` (name or IP literal), using the supplied `port` number, with a maximum delivery bandwidth of `mbw`, and a maximum of `bpc` bytes in each data chunk, including the 8-byte chunk header. The bandwidth can be expressed in terms of bits/second, kilobits/second, megabits/second, gigabits/second or terabits/second; for a definition of `"mbw"`, see 10.2.4. The `bpc` value shall be no smaller than 32 and no larger than 4096.

This request field shall only be used to direct the response data associated with an established "http-udp" transport. Servers shall ignore the request field if the transport type associated with the request is not "http-udp". Otherwise, response data is framed into chunks and delivered via UDP datagrams in the manner described in Annex K. Moreover, in this case, the client shall not send acknowledgement datagrams in response to these delivered chunks, nor should the server expect them.

The effect of this request field is non-persistent; it applies only to the response data associated with the request in which it is found.

NOTE 1 – A request is associated with the "http-udp" transport type in one of two possible circumstances: a) the request contains a "new-channel" request field and the server grants the request with a new channel that uses the "http-udp" transport, as indicated by the `JPIP-cnew` response header; or b) the request specifies a channel-id that has been issued for a channel using the "http-udp" transport and no new JPIP channel is issued by the server in response to this request.

NOTE 2 – Because response data delivered to the address specified by a Sendto request field is not explicitly acknowledged, clients can benefit from using the abandon and barrier request fields, which can be used to effect reliable communications. Also, because the server receives no acknowledgement information from which to estimate channel conditions, such as bandwidth and loss probability, it is the client's responsibility to perform whatever estimation might be necessary and supply an appropriate delivery bandwidth and chunk size.

C.7.6 Abandon (abandon)

```
abandon = "abandon" "=" 1#chunk-range

chunk-range = chunk-qid ":" chunk-seq-range

chunk-qid = UINT

chunk-seq-range = UINT-RANGE
```

This request field allows the client to explicitly inform the server about the absence of one or more data chunks that might have been sent in response to previous requests. Each occurrence of `chunk-range` informs the server of one or more data chunks that should be considered not to have arrived at the client. The server shall not consider any of the data associated with JPIP messages contained within these identified data chunks received or cached by the client, for the purpose of responding to this request or any subsequent request on this or any other JPIP channel, except in the event that the server receives, or has received, explicit acknowledgement of the arrival of these data chunks via acknowledgement datagrams.

If the request does not specify a channel-id which has been issued for a channel using the HTTP-UDP transport, the client shall not include any Abandon request field and the server shall ignore any such request field that it encounters.

NOTE 1 – The Abandon request field can be used regardless of whether the Sendto request field is present in the same request.

The `chunk-range` values identify data chunks via the 16 low-order bits of the request ID and the chunk sequence number; both of these values are found in the relevant chunk headers, as described in Annex K. The request ID component is identified by `chunk-qid` and matches the contents of the Request ID field in the chunk header the client wants to negatively acknowledge; no `chunk-range` shall have a `chunk-qid` value outside the range 0 to 65535.

The Abandon request field only applies to data chunks which have been transmitted or would be transmitted in response to previous requests within the same channel. To avoid ambiguity, servers shall ignore any Abandon request field which is part of the first request in a new JPIP channel – i.e., the request in which the channel's New Channel request field appears. Also, the Abandon request field does not apply to data chunks belonging to requests that have been excluded by means of a Barrier request field that appeared in a previous request within the channel.

NOTE 2 – It is possible that some of the data chunks affected by an Abandon request field have not been transmitted by the server by the time the request arrives. In this case, the server would typically abandon these data chunks immediately, without even transmitting them a first time. If this behaviour is not desired by the client, the client can avoid abandoning data chunks before at least one later data chunk within the same request or a data chunk from a later request have been received.

NOTE 3 – As explained in Annex B, this Recommendation | International Standard does not require the server to maintain a complete log of data which it has sent in response to client requests; nor does it require the server to exclude such data from its response to future requests. This means that a server can, at its discretion, choose to erase any log entry describing the transmitted chunks at any point. However, if the server does maintain a log of what has been sent to the client, for the purpose of avoiding redundant transmission in the future, it might need to keep track of the contents of data chunks for which it has not yet received acknowledgement information via acknowledgement datagrams or Abandon request fields, so that it can correctly respond to Abandon requests in the future. A server can choose to erase parts of its log at any time so as to reduce the burden of keeping track of unacknowledged data chunks. Alternatively, the client can use Barrier request fields to inform the server that it will never Abandon data chunks sent in response to a certain range of requests, so that the server need not keep track of unacknowledged data chunks belonging to that range.

C.7.7 Barrier (barrier)

```
barrier = "barrier" "=" barrier-qid

barrier-qid = UINT
```

This request field is provided to enable clients to inform servers of the requests for which response data chunks will not be abandoned via any subsequent request. The effect of Barrier request fields persists within the associated JPIP channel. Specifically, the effect of any Abandon request field in any subsequent request is limited to data chunks whose associated request has a request-id Q that is strictly greater than Qb, where Qb is the maximum of all barrier-qid values specified in this or any preceding request within the same JPIP channel.

If the request does not specify a channel-id that has been issued for a channel using the "http-udp" transport, the client shall not include any Abandon request field and the server shall ignore any such request field that it encounters.

NOTE 1 – The Barrier request field only affects the interpretation of Abandon request fields found in subsequent requests. Thus, for example, "barrier=3&abandon=3:4-7" means that the client is abandoning data chunks 4 to 7 from the request with request-id 3, but it will not abandon any data chunks from that request in the future.

NOTE 2 – The `chunk-qid` values supplied via a `chunk-range` in an Abandon request match any request whose request-id has the same least significant 16 bits as `chunk-qid`. On the other hand, the `barrier-qid` value supplies a full request-id, not just the least significant 16 bits.

C.7.8 Timed wait (twait)

```
twait = "twait" "=" max-wait-usecs

max-wait-usecs = UINT
```

This request field allows the client to suggest the latest point at which it would like the server to start responding to the current request, pre-empting the previous incomplete request, if any, within the same JPIP channel.

If there is no previous request within the JPIP channel this request field shall be disregarded by the server and the request shall be considered not to contain `twait` for the purpose of the ensuing description. If the previous request within the JPIP channel does not contain the `twait` request field, the latest pre-empt time is obtained by adding `max-wait-usecs` microseconds to the time at which the server began to serve that previous request. If one or more immediately preceding requests within the JPIP channel contain the `twait` request field, the latest pre-empt time is obtained by adding the `max-wait-usecs` values of all such requests, as a number of microseconds, to the time at which the server began to serve the most recent request within the channel that did not contain the `twait` request field.

Clients shall not issue requests that contain both the `twait` and `wait` request fields.

NOTE – In applications where animation is involved, clients might find it useful to send a succession of timed-wait requests, so that the server is able to optimize the actual service times to devote to each outstanding request, subject to their respective latest pre-empt times.

C.8 Cache management request fields

C.8.1 Model (model)

C.8.1.1 General

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

model-item = [codestream-qualifier ","] model-element

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

bin-descriptor = explicit-bin-descriptor    ; C.8.1.2

               / implicit-bin-descriptor    ; C.8.1.3

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

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

first-codestream-id = UINT

last-codestream-id = UINT
```

This field can be used in session-based or stateless requests. A session-based 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 conclusion of the processing of a stateless request by the server.

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 subclause. Clients can issue requests using either form and can mix the two forms of bin-descriptor within a single "model" request field if desired.

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 can add this information to its cache model and may assume that the client already has the indicated data.

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 might 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 can 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. If the last-codestream-id is not present, but the qualifier hyphen is, then this shall mean the first-codestream-id and all subsequent codestreams are included.

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

NOTE 1 – It is beneficial for the server to exploit additive cache model manipulation statements, but is free to disregard some or all of them at the possible expense of transport efficiency. It is therefore beneficial for clients to 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 can be used to determine the set of codestreams which are being modelled.

NOTE 2 – Manipulation of a session-based 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 and so "model" fields in requests that arrive using one channel (Channel ID field) might affect the response to requests that arrive using a different channel. It is worth noting that requests which use different JPIP channels (different Channel ID values) might 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. For this reason, it is beneficial for clients to 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 = %x4c UINT             ; "L"

codestream-main-header-bin = %x48 %x6d   ; "Hm"

meta-bin = %x4d bin-uid                  ; "M"

tile-bin = %x54 bin-uid                  ; "T"

tile-header-bin = %x48 bin-uid           ; "H"

precinct-bin = %x50 bin-uid              ; "P"

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

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). 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 (see Annex A).

The wildcard character, "*", shall be used only 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), relevant to the view-window.

Each bin-descriptor can be qualified by a number of bytes. An additive bin-descriptor which is qualified by the number of bytes, *B*, indicates that the client already has at least the first *B* bytes of the indicated data-bin in its cache; the server can 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.

EXAMPLE 1: A qualified subtractor bin-descriptor such as "-P23:10" means 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 assume this by adding these bytes to its cache model if they were not already present.

Precinct bin-descriptors can 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 at least the first *L* layers (first *L* packets) of the indicated precinct; the server can 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.

A bin-descriptor with no number-of-bytes or number-of-layers qualifier means the entire explicit data-bin.

EXAMPLE 2: "model=M0,Hm,H7:20,P3" means that the client has at least all of metadata-bin 0, all of the main codestream header, the first 20 bytes of tile header 7, and all of precinct 3 in its cache.

EXAMPLE 3: "model=P3:256,P5:L2,-P6:20" means that the client has at least the first 256 bytes of precinct 3 and the first two layers (packets) of precinct 5, but (at most) it does not have anything beyond the 20-th byte of precinct 6 (it might not have the first 20 bytes either).

EXAMPLE 4: "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. The wildcards used here are permissible only when the "model" statement appears in a stateless request.

EXAMPLE 5: "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 = %x74    ; t -- tile
                    / %x63    ; c -- component
                    / %x72    ; r -- resolution level
                    / %x70    ; 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 only applicable to JPP-stream requests. The bin-descriptor values that implicitly refer to data-bins are of the following types: t (tile to which the precinct belongs), c (image component to which the precinct belongs), r (resolution level of the tile-component to which the precinct belongs) or p (position of the precinct within its tile-component-resolution). Implicit bin-descriptors are used to identify precinct data-bins via the indices. All indices shall start from 0. A resolution level index of 0, `r0`, refers to the lowest resolution level (LL sub-band) of the tile-component. Position indices, `p`, run from left to right and top to bottom of the tile-component-resolution progression, in scan-line fashion, as described in Rec. ITU-T T.800 | ISO/IEC 15444-1.

In stateless requests, any or all of the tile, component, resolution level or position implicit-bin specifier can be replaced with the index range or the wildcard character, "*". In either case, the bin-descriptor is expanded to include all values of the index range relevant to the view-window. 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 can 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 shall 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 can 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 fifth 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. 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. 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. The wildcard and the range are appropriate only for stateless requests.

C.8.2 Summary of cache descriptor options (informative)

See Table C.5.

Table C.5 – Cache descriptor option summary

Form type	Wildcard		Index-range	number-of-layers (e.g., ":L3")	number-of-bytes (e.g., ":256")
	stateless	session-based			
Explicit form	Allowed	Not allowed	Not allowed	Allowed	Allowed
Implicit form	Allowed	Not allowed	Allowed only for stateless	Allowed	Not allowed

C.8.3 Tile-part model involving JPT-streams (tpmodel)

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

tpmodel-item = [codestream-qualifier "," ] tpmodel-element

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
```

This field can 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 session-based 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 can 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 or 5.2.

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

Codestream-qualifier values can 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.

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 session-based or a stateless request.

EXAMPLE 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 codestream 0 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 Need for stateless requests (need)

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

need-item = [codestream-qualifier "," ] bin-descriptor
```

This field shall only appear 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 might 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 can 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 this cache model to determine the elements that need not be sent to the client.

Codestream-qualifiers can 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 20-th 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 256-th byte (or from byte 256) of precinct data-bin 1, and layers beyond the second layer 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 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 fourth layer 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 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 Tile-part need for stateless requests (tpneed)

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

tpneed-item = [codestream-qualifier "," ] tp-descriptor
```

This field shall only appear 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 might 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 can 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 this cache model to determine the elements that need not be sent to the client.

Codestream-qualifiers can 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 Model set for requests within a session (mset)

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

This 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 an "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 Model Set response header, as discussed in D.2.18.

The "mset" request field's parameter string consists of a comma-separated list of ranges of codestream indices, possibly subsampled, following the conventions outlined in connection with the Codestream request field in C.4.9.

In addition to codestreams mentioned in the "mset" request, the server can 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 in C.4.10), unless the server indicates a reduced set of codestreams via a Codestream response header (see D.2.9). 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 can 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 can, 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 shall 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 – The server can 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                    ; 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
vendor-capability = "vc." vendor-code [":" vendor-value]
vendor-code = 1*(LOWER / DIGIT / "." / "-")
vendor-value = TOKEN
depth-capability = "depth:" UINT
config-capability = "config:" UINT
```

This field specifies the 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 can be determined or anticipated by other means. The capabilities associated with a channel can 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 Recommendation | International Standard are described in Table C.6.

Table C.6 – Valid capabilities of the `processing-capabilities` element

Capability	Meaning
`compatibility-capability`	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 string "cc.jp2_" in a Capability request field. A compatibility-code value of "jp2c" shall be used to indicate support for raw JPEG 2000 codestreams.
`vendor-capability`	The client supports the vendor capability defined by `vendor-code`. `vendor-code` 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 `vendor-code` for this feature shall be "com.example.distance". `vendor-value` specifies an optional value, 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 component for which the client has the greatest bit depth capability. As an example, if a client supports 12 bits for luminance and 8 bits for chrominance, the value of depth-capability shall be 12.

NOTE – 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 can 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.7.

Table C.7 – Valid values of the `config-capability` parameter

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

Table C.7 – Valid values of the `config-capability` parameter

Value	Meaning
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 Recommendation | International 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           ; C.10.2.2
                 / colour-meth-pref           ; C.10.2.3
                 / max-bandwidth              ; C.10.2.4
                 / bandwidth-slice            ; C.10.2.5
                 / placeholder-pref           ; C.10.2.6
                 / codestream-seq-pref        ; C.10.2.7
                 / conciseness-pref           ; C.10.2.8
                 / 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 can 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 (required), 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 D.2.23 for more on the Unavailable preference response header. Supporting a preference means that the server provides functionality which affects its behaviour in accordance with the preference. This addresses the server's functionality rather than the specific parameters established by other aspects of the request.

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 might be (see C.10.2.2 for 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 C.10.2.3 for a discussion of colourspace 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 Unavailable preference response 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 Recommendation | International Standard defines two options to specify the behaviour of the server in the event that the request cannot be serviced exactly as stated. These two options are specified in Table C.8.

Table C.8 – View-window handling preferences

Option	Meaning
"fullwindow"	The server shall honour the view-window request parameters but is allowed to deliver the data in arbitrary order. In the event that the server does modify view-window request parameters, the modified view window shall be such that the minimal set of data to completely satisfy the modified view-window shall be identical to the minimal set of data required to satisfy the originally requested view-window.
"progressive"	The server may modify the view-window request parameters in order to retain the progressive properties of the response data. In the event that the server does modify view-window request parameters, the modified view window shall be such that the minimal set of data to completely satisfy the modified view-window shall be a subset of the minimal set of data required to satisfy the originally requested view-window.

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

NOTE – The interpretation of "progressive" delivery can be affected by the presence of a Delivery Rate request field, as explained in C.7.4. The *view-window-pref* field provides strategies for a server operating under resource constraints to satisfy a request that might otherwise exceed these resources. The "`progressive`" mode allows the server to shrink the source window in order to provide a more uniform progression over the view window, whereas the "`fullwindow`" mode allows the server to reorder data arbitrarily in order to deliver the full window.

C.10.2.3 Colourspace method preference

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

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

meth-limit = UINT
```

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

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

If the Client Preferences request field does not contain any colourspace method preferences or the server does not support this field but is able to retrieve the client capabilities, then the supported colourspace methods are defined according to the information contained within the Capability field, and no preference is defined.

Each colourspace method preference consists of two parts: the particular colourspace method, and an optional limit on that preference. Valid values of the colourspace method are specified in Table C.9.

Table C.9 – Colourspace method client preferences

Method	Meaning
"color-enum"	The client prefers colourspace specifications that use the Enumerated Method
"color-ricc"	The client prefers colourspace specifications that use the Restricted ICC Method
"color-icc"	The client prefers colourspace specifications that use the Any ICC Method
"color-vend"	The client prefers colourspace specifications that use the Vendor Method

The optional `meth-limit` value specifies a limit on the APPROX value for that particular colourspace method. When using these preferences to select a colourspace specification, the server shall consider a colourspace 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 colourspace 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 might not care at all about colour fidelity and set `meth-limit` to 4, meaning that the accuracy of the colourspace methods is unimportant. Another application that displays images on a low-quality screen might set `meth-limit` to 3, 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. Allowed values are defined by the definition of the APPROX field in Rec. ITU-T T.801 |

ISO/IEC 15444-2. If the optional `meth-limit` value is not supplied, the default value shall be the largest value defined in that Recommendation | International Standard. When selecting which Colourspace Specification box to transmit to the client, the server shall use the algorithm shown in Figure C.3.

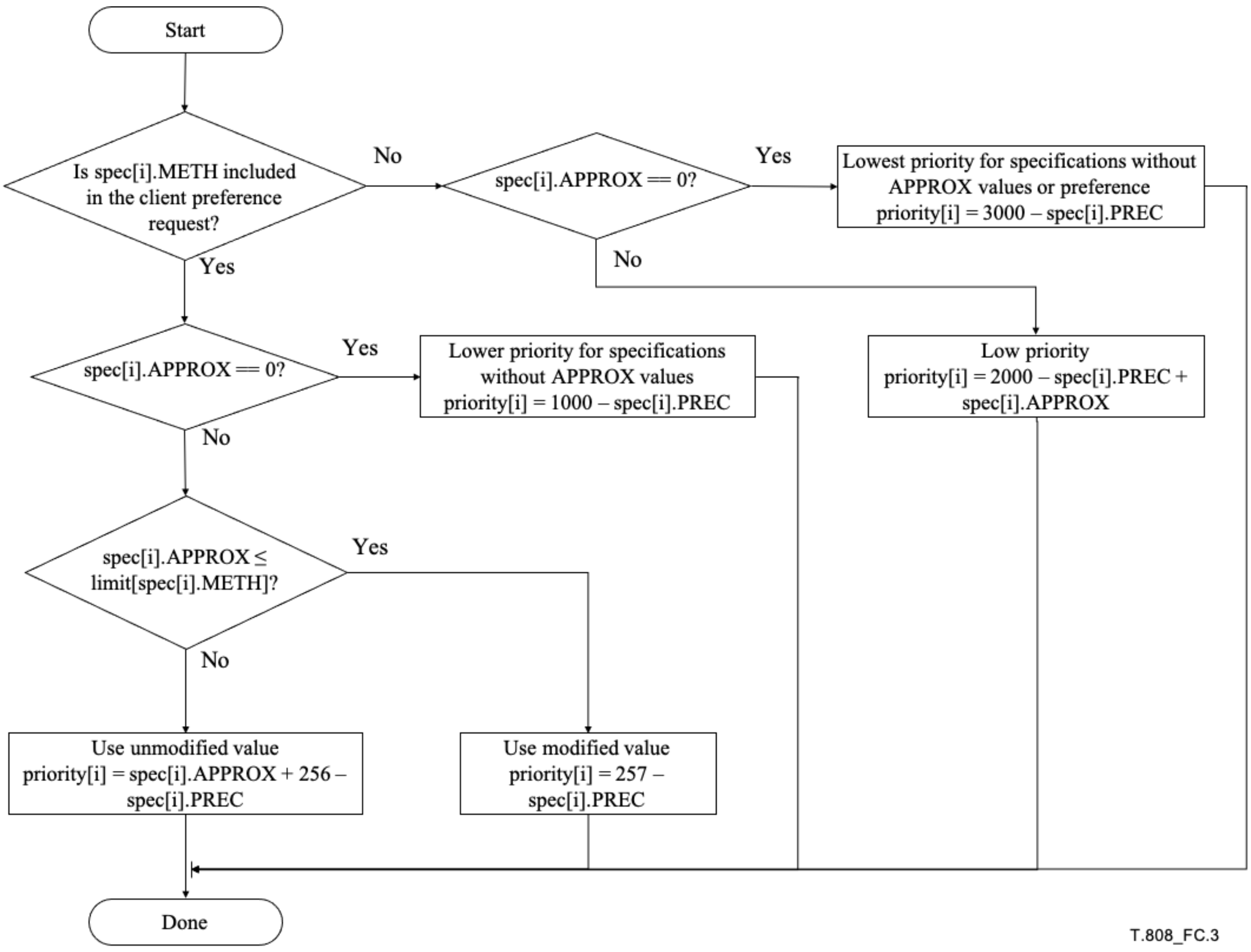
Start
Is spec[i].METH included in the client preference request?
No
spec[i].APPROX == 0?
Yes
Lowest priority for specifications without APPROX values or preference priority[i] = 3000 – spec[i].PREC
No
Yes
spec[i].APPROX == 0?
Yes
Lower priority for specifications without APPROX values priority[i] = 1000 – spec[i].PREC
Low priority priority[i] = 2000 – spec[i].PREC + spec[i].APPROX
No
spec[i].APPROX ≤ limit[spec[i].METH]?
Yes
No
Use unmodified value priority[i] = spec[i].APPROX + 256 – spec[i].PREC
Use modified value priority[i] = 257 – spec[i].PREC
Done
T.808_FC.3

Figure C.3 – 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 valid 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 colourspace 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 = NONZERO ["K" / "M" / "G" / "T"]
```

This preference signals the maximum rate at which the client would like to be sent data per logical target. 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 = 1024^2 bits. If the `mbw` value ends in "G", the value is in gigabits/second, where 1 gigabit = 1024^3 bits. If the `mbw` value ends in "T", the value is in terabits/second, where 1 terabit = 1024^4 bits. Otherwise, the value is in bits/second. Either the capacity of the server or the network might further limit the available maximum bandwidth for the JPIP service.

C.10.2.5 Bandwidth slice

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

slice = NONZERO
```

This preference can 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 can 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 can 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 clauses A.3.6 and A.4. As explained in 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 can signal an alternate preference using the mechanism described here. Valid values of the Placeholder preference are specified in Table C.10.

Table C.10 – Placeholder preferences

Method	Meaning
`"orig"`	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.
`"equiv"`	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.
`"incr"`	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 valid 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" / "reverse-sequential" / "interleaved"
```

This preference can be used to indicate how the client desires that the server deliver multiple codestreams that have been requested within a single request. Valid values of the Codestream sequencing preference are specified in Table C.11.

Table C.11 – Codestream sequencing preferences

Method	Meaning
`"sequential"`	The client would prefer to receive the multiple codestreams in a frame sequential order (e.g., serve multiple frames in a Motion JPEG 2000 file in a sequential order).
`"reverse-sequential"`	The client would prefer to receive the multiple codestreams in a frame sequential order (i.e., multiple frames in a Motion JPEG 2000 file), in the reverse order.
`"interleaved"`	The client would prefer to receive the multiple codestreams in an interleaved manner (e.g., server interleaved multiple compositing layers from a JPX file).

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

C.10.2.8 Conciseness preference

```
conciseness-pref = "loose" / "concise"
```

This preference can be used to indicate how strictly a server shall bind its response to the request made by the client, and how much excess data (i.e., data included in the response that is not required to satisfy the request) the server is allowed to include. Allowed values of the conciseness-preference are specified in Table C.12. Servers may ignore any `"/r"` modifier applied to this preference, and its usage is discouraged because its meaning is undefined.

Table C.12 – Conciseness preferences

Method	Meaning
`"concise"`	The server should produce the smallest response that it is capable of that satisfies the request.
`"loose"`	The server may include data which it deems appropriate to the request beyond the data necessary to satisfy the request.

To the extent that the requested view window is modified in accordance with response headers (see D.2), the above definitions are to be interpreted in terms of the modified view window.

Example: Consider a client that performs a series of requests whose view window follows an identifiable trajectory. If the *conciseness-pref* is not set to `"concise"`, the server may include data anticipating the future interests of the client. A client might use the *conciseness-pref* set to `"concise"` to discourage the server from following such a strategy.

C.10.3 Contrast 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 field can be used to supply information concerning contrast sensitivity. While this information might 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 can 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 $m(\omega_1, \omega_2)$ given $\omega_1 = \pi d^n \cos\psi$, $\omega_2 = \pi d^n \sin\psi$, 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.

C.10.4 Handled (handled)

```
   handled = "handled"
```

If this request field is present, the server shall include a JPIP-handled response header within its response, identifying request fields which the server is prepared to handle.

NOTE – The JPIP-handled response header is defined in D.2.26.

Annex D

Server response signalling

(This annex forms an integral part of this Recommendation | International Standard.)

D.1 Reply syntax

D.1.1 Introduction

This annex describes all possible elements in a JPIP response. Each major subclause describes the status code and its associated reason phrase, response headers and possible values for those headers, and the response data. In general, a response will consist of multiple response headers.

D.1.2 Reply structure

The JPIP response consists of the following elements:

- status-code;
- reason-phrase;
- jpip-response-header;
- response 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 response data (if any) appears in the HTTP entity-body.

```
Status-Code = 3DIGIT

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

jpip-response-header =
                     / JPIP-tid                  ; D.2.2
                     / JPIP-cnew                 ; D.2.3
                     / JPIP-qid                  ; D.2.4
                     / JPIP-fsiz                 ; D.2.5
                     / JPIP-rsiz                 ; D.2.6
                     / JPIP-roff                 ; D.2.7
                     / JPIP-fvsiz                ; D.2.8
                     / JPIP-rvsiz                ; D.2.9
                     / JPIP-rvoff                ; D.2.10
                     / JPIP-comps                ; D.2.11
                     / JPIP-stream               ; D.2.12
                     / JPIP-context              ; D.2.13
                     / JPIP-roi                  ; D.2.14
                     / JPIP-layers               ; D.2.15
                     / JPIP-srate                ; D.2.16
                     / JPIP-metareq              ; D.2.17
                     / JPIP-len                  ; D.2.18
                     / JPIP-quality              ; D.2.19
                     / JPIP-type                 ; D.2.20
                     / JPIP-mset                 ; D.2.21
                     / JPIP-cap                  ; D.2.22
                     / JPIP-pref                 ; D.2.23
                     / JPIP-align                ; D.2.24
                     / JPIP-subtarget            ; D.2.25
                     / JPIP-handled              ; D.2.26
```

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

D.1.3 Status codes and reason phrases

D.1.3.1 General

The status code is a 3-digit integer result from 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 might see other status codes as well.

D.1.3.2 200 (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.1.3.3 202 (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 consequently superseded the request (because `wait=no`). When the first request becomes irrelevant before the server is able to process and commence transmission of a response, then the 202 status code shall be used. This is a common occurrence in practice, since an interactive user can 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.1.3.4 400 (Bad request)

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

D.1.3.5 404 (Not found)

This status code should be issued if the server cannot locate the logical target to which the request refers, through the "`target`" request field, the "`target-id`" request field, or any other means such as the <resource> component of an HTTP GET or POST request.

This status code should also be issued if a "`subtarget`" request field refers to a non-existent or inappropriate byte range within the requested resource.

D.1.3.6 415 (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.1.3.7 501 (Not implemented)

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

D.1.3.8 503 (Service unavailable)

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

D.1.4 Impact of errors on the server state

In an event the server issues an error code different from 200 and 202, it shall not modify its state by processing request fields contained in the corresponding request and shall not return response data. The server shall, however, update the `qid`. In case the error code generated by a client-preferences request using the "`/r`" modifier is not supported by the server, and the server is operating in a session, it is desirable that the server keeps the session available for future requests.

NOTE – If the server is operating in a session, an alternative option for the server would be to first return an error code and then terminate the session, e.g., return 503 for all further requests on this session.

Example: Consider a client issuing the invalid request:

```
cid=1&rsiz=100,100&fsiz=100,100&cnew=2&roff=-1,-1
```

then the server shall report this invalid request, shall not process the request for the view window, shall not modify its cache model and shall not create a new channel. It shall abort the request and return an error code.

Example: Consider a client issuing the valid request:

```
cid=1&rsiz=10000,10000&fsiz=10000,10000&pref=fullwindow/r
```

and the server cannot honour the *view-window pref* "`fullwindow`" for the requested window size, then the server shall not process the request, shall not return any data for the request, and shall issue the error code 501 including the JPIP-response header "`JPIP-pref: fullwindow/r`" without modifying its internal state. It should keep the session available for future requests if feasible, though.

D.2 JPIP response headers

D.2.1 Introduction to JPIP response headers

In responding to a client request, the server may modify some aspects of the request. If the server modifies the request, 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-`". Unless otherwise specified, if the parameters identified in the response header had been originally specified in the client's request, then the server would have responded in the same way, except the response would now not contain these response headers. In addition, JPIP response headers may be sent by the server to inform the client of the values of other unspecified request fields for use in future requests.

The `JPIP-qid` response is an exception in that it shall be sent whenever the client has included a Request ID in the request, and then value of `JPIP-qid` shall always be the same as `qid.`

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 rule are found in connection with the New Channel and Quality response headers.

D.2.2 Target ID (JPIP-tid)

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

The server shall 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 server is unable to assign unique identifiers to the requested logical target, and hence cannot guarantee its integrity between multiple requests or sessions, then the Target ID response header shall specify a value of 0. If the server supplies a `target-id` which is different from that specified in the request, it shall disregard all `model`, `tpmodel`, `need` and `tpneed` request fields when responding to this request.

D.2.3 New channel (JPIP-cnew)

```
JPIP-cnew = "JPIP-cnew" ":" LWSP "cid" "=" channel-id
            ["," 1#(transport-param "=" 1*(IDTOKEN / "=" / "/" / "\"))]

transport-param = TOKEN
```

The server shall 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 Recommendation | International Standard (see Table D.1).

Table D.1 – Valid 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.
`"auxport"`	This parameter is used with transports requiring a second physical channel. If the "http-tcp" or "http-udp" transports are used, the auxiliary port is used to connect the auxiliary channel. For further details, see Annexes G and K. 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 Request ID (JPIP-qid)

```
JPIP-qid = "JPIP-qid" ":" LWSP UINT
```

The server shall send this response header if the client's request included a Request ID `qid`. The value of `JPIP-qid` shall be identical to `qid`. The server shall not include a Request ID response header when the respective client request did not include a Request ID. The server's Request ID, `JPIP-qid`, shall always be identical to the client's Request ID. Thus the Request ID is distinctive in that this response header is sent when the client has used the Request ID, not when the server modifies the value.

D.2.5 Frame size (JPIP-fsiz)

```
JPIP-fsiz = "JPIP-fsiz" ":" LWSP 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.2.6 Region size (JPIP-rsiz)

```
JPIP-rsiz = "JPIP-rsiz" ":" LWSP 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. A server shall only modify view-windows in accordance with Table C.8 and the description of the *view-window-prefs*, in C.10.2.2. Specifically, a server is not allowed to enlarge the requested view window. It may, however, at its discretion, transmit data outside of the requested view window in accordance with Table C.12 and the description of the *conciseness-pref*, in C.10.2.8.

D.2.7 Offset (JPIP-roff)

```
JPIP-roff = "JPIP-roff" ":" LWSP 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.2.8 Frame size for variable dimension data (JPIP-fvsiz)

```
JPIP-fvsiz = "JPIP-fvsiz" ":" LWSP 1#UINT
```

The server should send this response header if the actual frame size differs in any way from that requested via the Frame Size or Frame Size for Variable Dimension Data field. The server might need to modify the frame size because the client requested a frame size that does not exist. It is at the discretion of the server to either return the `JPIP-fsiz` or the `JPIP-fvsiz` response header on two-dimensional data requests, both responses shall be considered equivalent in this case. In all other cases, only the `JPIP-fvsiz` response header shall be used.

D.2.9 Region size for variable dimension data (JPIP-rvsiz)

```
JPIP-rvsiz = "JPIP-rvsiz" ":" LWSP 1#UINT
```

The server should send this response header if the size of the view-window differs in any way from that requested via the Region Size or Region Size for Variable Dimension Data request field. If two-dimensional data had been requested, it is at the discretion of the server to pick either this response header, or the `JPIP-rsiz` response header, and both shall be considered equivalent by the client. For all other cases, only the `JPIP-rvsiz` response header shall be used.

D.2.10 Offset for variable dimension data (JPIP-rvoff)

```
JPIP-rvoff = "JPIP-rvoff" ":" LWSP 1#UINT
```

The server should send this response header if the view-window offset differs in any way from that requested via an Offset or Offset for Variable Dimension Data request field. The server might need to modify the offset if it is resizing a requested view-window. For two-dimensional data, it is at the discretion of the server to pick either this response header, or the `JPIP-roff` response header, and the client shall consider both equivalent. For all other data, the `JPIP-rvoff` response header shall be used.

D.2.11 Components (JPIP-comps)

```
JPIP-comps = "JPIP-comps" ":" LWSP 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 response header if requested image components do not exist within any of the requested codestreams.

D.2.12 Codestream (JPIP-stream)

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

prefixed-range = "<" ctxt-id ":" ctxt-elt ">" sampled-range

ctxt-id = UINT

ctxt-elt = UINT
```

The server should send this response header to inform the client of the codestream or codestreams for which it will serve data, unless it is serving data in response to all codestreams requested directly via any Codestream request field and all codestreams requested indirectly via any Codestream Context request field. The server should use the `prefixed-range` syntax to identify those codestreams for which data is being served in response to a translated Codestream Context request field. In this case, the `ctxt-id` value shall identify the specific `context-range` from the Codestream Context request field whose translation is producing the relevant codestreams. Moreover, the `ctxt-elt` value shall identify the particular element within the `context-range` identified by `ctxt-id`, whose translation is producing the relevant codestreams.

A value of 0 for the `ctxt-id` means that the first `context-range` in the Codestream Context request field is the one which produced the range of codestreams which follows the prefix. Similarly, a value of 1 for `ctxt-id` means that the second `context-range` in the Codestream Context request field is the one which produced the ensuing range of codestreams, and so forth.

A value of 0 for the `ctxt-elt` means that the first context in the relevant `context-range` is the one which produced the range of codestreams which follows the prefix.

Example:

Client request:

stream=0&context=jpxl<2-7:2>[s0i0],jpxl<9-10>[s1i3]

Server response:

JPIP-context: jpxl<2-7:2>[s0i0]=0,1;jpxl<9-10>[s1i3]=0

JPIP-stream: 0,<0:1>1,<1:0>0,<1:1>0

This means that the server is responding with data resulting from:

1) the direct application of the view-window to codestream 0 (as requested via "stream=0");
2) the translation of the view-window to JPX compositing layer 4, according to compositing instruction 0 in compositing instruction set 0, as it applies to codestream 1;
3) the translation of the view-window to JPX compositing layer 9, according to compositing instruction 3 in compositing instruction set 1, as it applies to codestream 0; and
4) the translation of the view-window to JPX compositing layer 10, according to compositing instruction 3 in compositing instruction set 1, as it applies to codestream 0.

D.2.13 Codestream Context (JPIP-context)

```
JPIP-context = "JPIP-context" ":" LWSP 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:

a) the server may restrict the collection of image elements (e.g., compositing layers) which were originally requested;
b) 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.2.14 ROI (JPIP-roi)

```
JPIP-roi = "JPIP-roi" ":" LWSP
           "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.2.15 Layers (JPIP-layers)

```
JPIP-layers = "JPIP-layers" ":" LWSP 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 might 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 Alignment request field (see 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.2.16 Sampling rate (JPIP-srate)

```
JPIP-srate = "JPIP-srate" ":" LWSP 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.2.17 Metadata request (JPIP-metareq)

```
JPIP-metareq = "JPIP-metareq" ":" LWSP
               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.2.18 Maximum response length (JPIP-len)

```
JPIP-len = "JPIP-len" ":" LWSP 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. If returned, `JPIP-len` shall be a value that informs the client of a suitable maximum response length, `len`, for subsequent requests. If `len=0`, the server should respond to the request with response headers and no response data.

D.2.19 Quality (JPIP-quality)

```
JPIP-quality = "JPIP-quality" ":" LWSP (1*2DIGIT / "100" / "-1")
```

The server can 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 might 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.2.20 Image return type (JPIP-type)

```
JPIP-type = "JPIP-type" ":" LWSP 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.2.21 Model set (JPIP-mset)

```
JPIP-mset = "JPIP-mset" ":" LWSP 1#sampled-range
```

The server should include this response header if the client's request contains a Model Set request field, and the collection of codestreams identified by the client's Model Set 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" can be no larger than that identified by the client's Model Set request field.

D.2.22 Needed capability (JPIP-cap)

```
JPIP-cap = "JPIP-cap" ":" LWSP 1#capability-code
```

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

D.2.23 Unavailable preference (JPIP-pref)

```
JPIP-pref = "JPIP-pref" ":" LWSP 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 the same form as they appeared in the Client Preferences request, except that numerical parameters only need to be represented to sufficient accuracy to avoid any ambiguity in identifying the unsupported preference.

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.2.24 Alignment (JPIP-align)

```
JPIP-align = "JPIP-align" ":" LWSP "yes" / "no"
```

This response header should be provided if the server alignment guarantee differs from that requested by the client. (See C.7.1.)

D.2.25 Subtarget (JPIP-subtarget)

```
JPIP-subtarget = "JPIP-subtarget" ":" LWSP byte-range / src-codestream-specs
```

This response header should be provided if the subtarget identified by the server differs from that requested by the client. (See C.2.3.)

D.2.26 Handled request (JPIP-handled)

```
JPIP-handled = "JPIP-handled" ":" LWSP 1#handled-req

handled-req = (request-field | partially-handled-req)

partially-handled-req = request-field "=" handled-req-option

request-field = TOKEN

handled-req-option = TOKEN
```

The server shall include this response header in its response to a request containing the `handled` request field. This `JPIP-handled` response header identifies the requests which the server is able to handle correctly, in accordance with this Recommendation | International Standard. Each `request-field` can be any of the request fields mentioned in C.1.2, but may also include other tokens that some clients might not recognize; clients shall ignore any request-field they do not understand.

A `partially-handled-req` can be used to indicate partial support for a request field. If the relevant request field has a finite set of possible complete parameter strings following the "`=`" character (e.g., "`yes`" or "`no`"), the `handled-req-option` can be one of those values. Table D.3 describes additional values for the `handled-req-option` which are defined by this Recommendation | International Standard for use with specific request fields. Servers may include other tokens for the `handled-req-option` that some clients might not recognize. Clients shall ignore any `partially-handled-req` whose `request-field` or `handled-req-option` they do not understand.

D.3 Response data

For anything other than the JPP- or JPT-stream image return types, including raw codestream, the response data should consist of the requested entity in full. For JPP- or JPT-stream image return types, the response 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 of 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 Recommendation | International Standard provides no normative interpretation for the contents of the EOR message body.

The EOR message, header and body, is the only message which does not contribute to the byte count restriction associated with the *Maximum Response Length* request field as defined in C.6.1.

NOTE – The EOR message 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 message would mean that all the contents of the relevant data-bins have been served.

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

Table D.2 – Defined reason codes

Reason code	Reason	Explanation
1	Image done	The server has transferred all available image information (not just information relevant to the requested view-window) to the client. This reason code has a particular meaning to session-based requests. For a session-based request, this reason code implies that the client has received all data which could be sent in response to any session-based request associated with this logical target. With the possible exception of requests which include cache management requests fields, any subsequent session-based request will be responded with no response data and `R=1 EOR`.
2	Window done	The server has transferred all available information that is relevant to the requested view-window. This reason code has a particular meaning to session-based requests. For a session-based request, this reason code implies that the client has received all data which could be sent in response to this request and the response data was not limited by any data-limit-field (`len or quality`) in the request, or by the handling of a subsequent request. With the possible exception of requests which include cache management request fields, any subsequent repetition of the request will be responded with no response data and `R=2 EOR`.
3	Window change	The server is terminating its response in order to service a new request which does not specify `Wait=yes`.
4	Byte limit reached	The server is terminating its response because the byte limit specified in a Maximum Response Length request field has been reached.
5	Quality limit reached	The server is terminating its response because the quality limit specified in a Quality request field has been reached.
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 associated with that session.
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 associated with that session.
0xFF	Non-specified reason	The server is terminating its response for a reason that is not specified.
Other values		Reserved for ISO use.

Table D.3 – Additional `handled-req-option` values for particular request fields

request-field	handled-req-option	Meaning
Cnew	transport-name	The server correctly handles new-channel request fields that contain the indicated transport type.
Context	"jplx", "mj2t", "jpmp", "jpxf"	The server correctly handles codestream context request fields for context-range values that commence with the handled-req-option token.

Annex E

Uploading images to the server

(This annex forms an integral part of this Recommendation | International Standard.)

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 Recommendation | International 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, Sub-target and Target ID request field (see C.2.2, C.2.3, and C.2.4) can also be used. For an upload of a complete image media type, the Frame Size, Offset and Region Size request fields (see clauses C.4.2, C.4.3, and C.4.4) 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 Recommendation | 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 can include Frame Size, Region Size and Offset request fields. The Frame Size request 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 request field, the server should limit modifications to the extent specified in the Region Size request 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 are presented in the following subclauses.

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 Updating the image

After receiving the uploaded data, the server can 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 might 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 request field. This is one way to prevent multiple simultaneous edits of a target by different clients. Servers providing editing capabilities can take care of such issues as target locking by some other means.

A JPIP client can upload part of a new image by specifying a target ID of 0, or using a new URL, or target which the server does not 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.4.2 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).

When merging data, the number or size of tiles shall not be changed and data that is not replaced by the upload process shall have the same meaning as it originally had before the upload.

E.4.3 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.

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.

When merging data, the number or size of precincts shall not be changed and data that is not replaced by the upload process shall have the same meaning as it originally had before the upload.

E.4.4 JPP-stream and JPT-stream metadata-bins

Metadata-bin can be uploaded, replacing the contents in an existing metadata-bin. 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.4.5 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

Using JPIP over HTTP

(This annex forms an integral part of this Recommendation | International Standard.)

F.1 Introduction

This annex defines the method to use JPIP with the HTTP protocol, as defined in RFC 2616, for both requests and responses. The JPIP request parameters from Annex C are encapsulated in Valid HTTP request structures. The server responses (including status codes, headers, messages, and response codes) from Annex D are encapsulated in valid HTTP responses. All requests and responses shall be encoded as specified by the HTTP standard.

The text and examples in this annex describe the use of JPIP over HTTP. The same binding shall be used for secure HTTP (or HTTPS).

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 1 – The HTTP Request is defined in RFC 2616 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 2 – The HTTP `Request-Line` and `Request-URI` are defined in RFC 2616 as:

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Request-URI = "*" / absoluteURI / abs_path / authority
```

NOTE 3 – RFC 3986 defines:

```
absoluteURI   = absolute-URI
absolute-URI  = scheme ":" hier-part [ "?" query ]
hier-part     = "//" authority path-abempty
              / path-absolute
              / path-rootless
              / path-empty
path-abempty  = *( "/" segment )
path-absolute = "/" [ segment-nz *( "/" segment ) ]
path-rootless      =      segment-nz      *(      "/"      segment      )
path-empty                                            =           0<pchar>
segment       = *pchar
segment-nz    = 1*pchar
```

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 Recommendation | International 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 valid 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.

An example of a JPIP upload request is:

```
POST /images/kids.jp2?rsiz=640,480&roff=320,240&fsiz=1280,1024 HTTP/1.1

Host: post.jpeg.org

Content-type: image/jpt-stream

CRLF
```

F.3 Session establishment

A session-based HTTP (or HTTPS) session is established by using the New Channel request field with a value of `"http"` (or `"https"`), i.e., `"cnew=http"` (or `"cnew=https"`) as part of a request. This request is typically delivered by HTTP (or HTTPS). The request can 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 can open an HTTP (or HTTPS) connection and issue a request which includes the HTTP (or HTTPS) header `"Connection: keep-alive"`. This is useful for efficient sessions, but it is neither necessary nor sufficient to have a session. A single HTTP (or HTTPS) 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 can arrive on HTTP (or HTTPS) connections other than the HTTP (or HTTPS) 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 valid HTTP response.

NOTE – The HTTP `Response` is defined in RFC 2616 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 valid HTTP responses, with further limitations on some of the parts of the HTTP response as described in the following subclauses.

F.4.2 Status code and reason-phrase

All of the status codes listed in D.1.3 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.1.3 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 information

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 can 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 can 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

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 can 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. An HTTP response shall have a mechanism to determine the length of the response. If the server does not plan to interrupt a response, it can 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 RFC 2616. In particular, "The HEAD method is identical to GET except that the server shall not return a message-body in the response."

Clients might find it useful to issue HTTP "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 might cause the server to update its cache model.

NOTE – A client wishing to update the server cache model without receiving a response can use the Maximum Response 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 might be necessary to stop the current response and send more data on the same connection for a new response.

NOTE 1 – RFC 2616 provides an algorithm for removing the chunked transfer encoding.

NOTE 2 – Chunked transfer encoding might be useful with JPIP when delivered over protocols other than HTTP.

F.6 HTTP and length request field (informative)

With a HTTP return channel, the server does not receive continuous feedback from the client and can easily push a great deal of data into the pipe, which needs to be fully received before any data for a new window can be processed. To maintain responsiveness, clients should use the Maximum Response Length request 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 G

Using JPIP with HTTP requests and TCP returns

(This annex forms an integral part of this Recommendation | International Standard.)

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 ("`cnew`") request field (see C.3.3) and the New Channel ("`JPIP-cnew`") response header (see D.2.3), where transport-specific details shall be communicated. This Recommendation | International Standard defines four specific transports, which are identified by the strings "`http`", "`https`", "`http-tcp`" and "`http-udp`" in the value string associated with New Channel requests. This annex provides details of the "`http-tcp`" 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-udp`" transport type is identified in this text as HTTP-UDP and is defined in Annex K.

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.

All requests sent over the HTTP transport shall be encoded as specified by the HTTP standard.

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 can be established to a JPIP server by issuing a request that includes the New Channel request field (see C.3.3). 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 in D.2.3) 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 message (see D.3). 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 response 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

Recommendation | International Standard. They can be used for additional server-specific signalling. The client will return the entire 8-byte chunk header in its chunk acknowledgement messages.

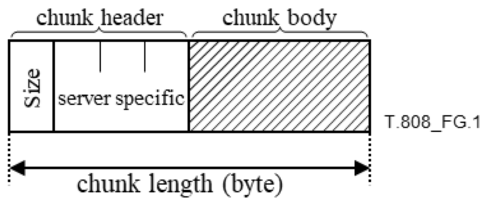

Figure G.1 – Response data structure on http-tcp connection

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 can only be used 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 is permitted to contain response data generated in response to two different requests.

The response to each and every request shall be terminated with an EOR message (see D.3), 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 might be little or no reason for using the Maximum Response Length request 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.

Annex H

Using JPIP with alternate transports

(This annex does not form an integral part of this Recommendation | International Standard.)

H.1 Introduction

The purpose of this annex is to provide guidelines on the deployment of JPIP over unreliable transports and provides a generic approach which can be applied to a wide variety of transports, in addition to the HTTP-UDP transport that is specified in Annex K.

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 can be interpreted as an extension of the "http-tcp" transport to unreliable transports. However, 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:

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

b) An unreliable packet-oriented service (for example, see "http-udp" in Annex K). In this case, packets might arrive out of order or not at all.

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

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 can be communicated exactly as in the "http-tcp" protocol, and indeed HTTP is recommended for the transport of requests and response headers.

The JPIP stream messages, including the EOR message (see D.3), shall be partitioned into packets and delivered over the unreliable data connection.

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

a) Each request should include a Request ID request field (see C.3.5).

b) 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.

c) 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.

d) 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 shall be encoded in the packet's header.

e) 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.

f) 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.

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

h) 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

This subclause is concerned with transports where both the request and data connections are unreliable. Guidelines for the data connection are exactly as described in H.2 for the case where data are delivered unreliably. With an unreliable request connection, 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 H.2 for unreliable data connections.

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

b) 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.

c) 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 need for acknowledgement messages to be sent in response to request packets. There is no need 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.

d) 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 any subsequent requests (e.g., because some packets of the request were unduly delayed). This provides the client with a mechanism for determining whether or not an important request was received by the server.

e) 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:

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

   2) The Request ID associated with the most recent request for which the flag mentioned in e1 was set.

   If the server does not receive one or more packets of a request with flag e1 set (i.e., requests with condition e2 arrive and the request with flag e1 is missing), it shall idle until the client retransmits the packets.

H.4 Request and response syntax

The request and response syntax described in Annexes C and 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

The New Channel request field (see D.2.3) and corresponding response header can be used to create channels associated with transport protocols other than the "http" and "http-tcp" transports described normatively in this Recommendation | International Standard. 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 that was used to create the channel, while response data should be delivered using the new channel's transport.

Annex I

Indexing JPEG 2000 files for JPIP

(This annex forms an integral part of this Recommendation | International Standard.)

I.1 Introduction (informative)

The Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | Standards define a family of JPEG 2000 file formats. The family utilizes 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 might 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 might be useful:

– to a server-side implementation of the JPIP protocol;
– to a client accessing an image remotely, using a simpler protocol, which 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 can be categorized as follows:

– The Codestream Index (cidx) superbox indexes codestream information corresponding to the main header, tile header, tile and precinct data-bin classes of the JPP-stream and JPT-stream. It contains a Codestream Finder (cptr) box pointing to the indexed codestream, a Manifest (manf) box summarizing the rest of the contents, and index table boxes, which are the 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 or Header Index Table listing the actual codestream elements. The Header Index Table, Precinct Packet and Packet Header index table superboxes also each contain a Manifest box.
– The File Index (fidx) superbox indexes file-level information corresponding to the metadata-bin class of the JPP-stream and JPT-stream. 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 can contain Proxy (prxy) boxes representing the contents of the indexed file or superbox.
– The Index Finder (iptr) box points to a root File Index, enabling its location to be discovered.

Figure I.1 illustrates an example JPEG 2000 file containing JPIP index boxes:

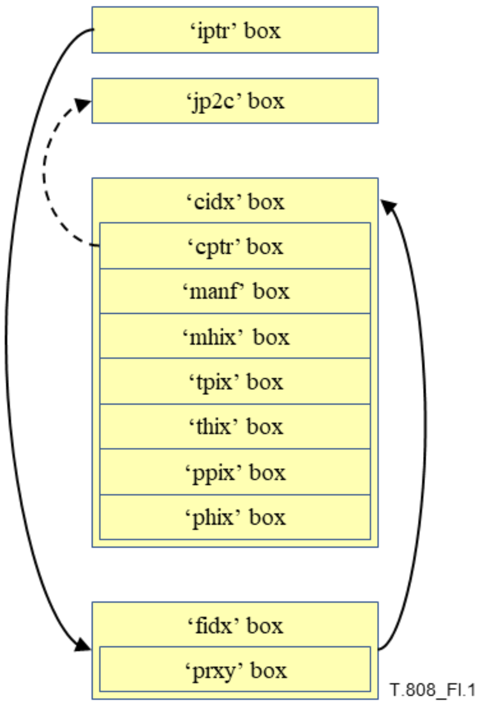

Figure I.1 – Part of an example JPEG 2000 file containing JPIP index boxes

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 Recommendation | International Standard may contain a CL^i field in the File Type box (as defined in Rec. ITU-T T.800 | ISO/IEC 15444-1) with the value 'jpip' (0x6a70 6970).

I.3 Defined boxes

I.3.1 General

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

Table I.1 is informative. Normative definitions of each box are contained within the individual subclauses referenced in the table.

Table I.1 – Defined boxes (Informative)

Box name	Type	e	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.
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.3.2 Codestream Index box (superbox)

I.3.2.1 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):

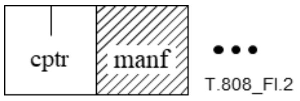

Figure I.2 – Organization of the contents of a Codestream Index box

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 summarizes the index tables following it inside the Codestream Index box. Its structure is specified in I.3.2.3.

I.3.2.2 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):

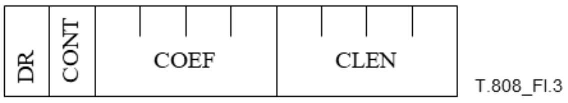

Figure I.3 – Organization of the contents of a Codestream Finder box

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. This field is stored as a 2-byte big endian unsigned integer. The values defined in this Recommendation | International Standard are described in Table I.2.

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.

Table I.2 – Container type values

CONT	Meaning
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 might 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.
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	Reserved for ISO use.

I.3.2.3 Manifest box

The Manifest box summarizes 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 can 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):

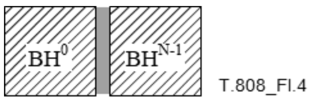

Figure I.4 – Organization of the contents of a Manifest box

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 Tile Header Index Table 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.3.2.4 Index tables

I.3.2.4.1 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, Precinct Packet Index Table and Packet Header Index Table boxes are superboxes containing Fragment Array Index boxes. The Tile Header Index Table box is a superbox containing Header Index Table boxes. Below we define first the Fragment Array Index box and then the Index Table boxes.

I.3.2.4.2 Fragment Array Index Box

The Fragment Array Index box lists the locations and lengths of the elements of a codestream. It is used within the Tile-part 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):

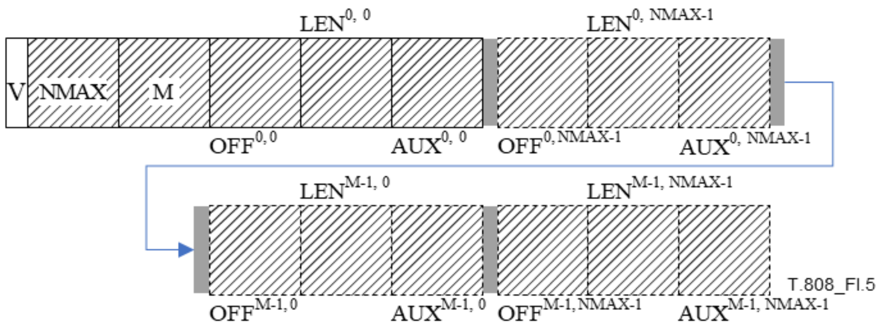

Figure I.5 – Organization of the contents of a Fragment Array Index box

V: Version. This field is encoded as a 1-byte unsigned integer. The values defined in this Recommendation | International Standard are described in Table I.3.

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 might have a smaller number of elements to specify. In this case, where for any row *i* containing *J* valid elements where *J* is less than NMAX, the values of OFF^{i,J} to OFF$^{i,NMAX-1}$ and LEN^{i,J} to LEN$^{i,NMAX-1}$ shall be set to zero.

Table I.3 – Version values

CONT	Meaning
0	NMAX, M and all 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.
1	NMAX, M and all 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.
2	All fields other than V are encoded as 4-byte big endian unsigned integers.
3	NMAX, M and all OFF^{i,j} and LEN^{i,j} fields are encoded as 8-byte big endian unsigned integers and all AUX^{i,j} fields are encoded as 4-byte big endian unsigned integers.
All other values	Reserved for ISO use.

I.3.2.4.3 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 can only occur 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 might 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):

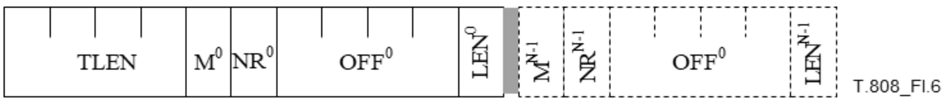

Figure I.6 – Organization of the contents of a Header Index Table box

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 Mi 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.3.2.4.4 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):

faix
T.808_FI.7

Figure I.7 – 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 the tile-parts in the corresponding tile, in codestream order. If the Fragment Array Index box has Version equal to 2 or 3, the Auxiliary fields specify for each tile-part the smallest *n* such that, in all components for which ($N_L - n$) is non-negative, resolution level ($N_L - n$) and all lower resolution levels have been completed when this tile-part is combined with all preceding tile-parts of the same tile, where N_L is the number of decomposition levels, which can vary by component. If no resolution levels of any component have been completed, the value of the Auxiliary field is one plus the maximum value of N_L across all components. The value zero is reached when all resolutions in all components have been completed. Because resolutions do not necessarily appear in order in a tile, some resolution levels above the value signalled by the Auxiliary field might have been completed.

I.3.2.4.5 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):

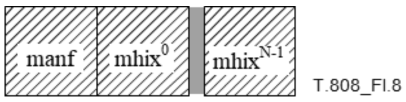

Figure I.8 – Organization of the contents of a Tile Header Index Table box

The number of Header Index Table boxes, N, is the number of tiles.

manf: Manifest box. This box summarizes the boxes specified by mhix[i] inside this Tile Header Index Table box. Its structure is specified in I.3.2.3.

mhix[i]: Header Index Table box. This box indexes the tile-part headers for the *i*-th tile. Its structure is specified in I.3.2.4.3.

I.3.2.4.6 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):

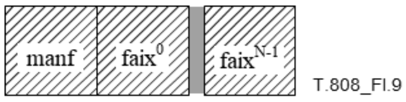

Figure I.9 – Organization of the contents of a Precinct Packet Index Table box

The number of Fragment Array Index boxes, *N*, shall be no greater than the number of codestream components.

manf: Manifest box. This box summarizes the boxes specified by faix[i] inside this Precinct Packet 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 packets in the corresponding tile-component. Packets appear contiguously, ascending in layer order, within their respective precincts, and precincts appear in the order associated with the sequence number *s*, defined in A.3.2.1. However, the fixed order of the packets is not necessarily the same as that specified in any COD/POC marker segments within the codestream. The structure of the Fragment Array Index box is specified in I.3.2.4.2.

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 can 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.3.2.4.7 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):

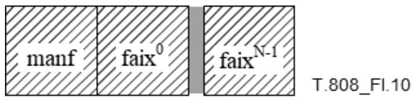

Figure I.10 – Organization of the contents of a Packet Header Index Table box

The number of Fragment Array Index boxes, N, shall be no greater than the number of codestream components.

manf:	Manifest box. This box summarizes 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, ascending in layer order, within their respective precincts, and precincts appear in the order associated with the sequence number *s*, defined in A.3.2.1. 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. The structure of the Fragment Array Index box is specified in I.3.2.4.2.

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 can 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.3.3 File Index box (superbox)

I.3.3.1 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):

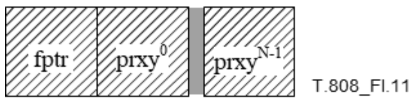

Figure I.11 – Organization of the contents of a File Index box

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 might be helpful to applications if, preceding any such proxied boxes, no boxes within the scope of the index are omitted from the index.

I.3.3.2 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):

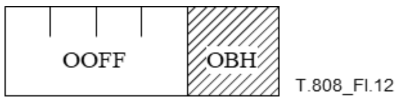

Figure I.12 – Organization of the contents of a File Finder box

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.3.3.3 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):

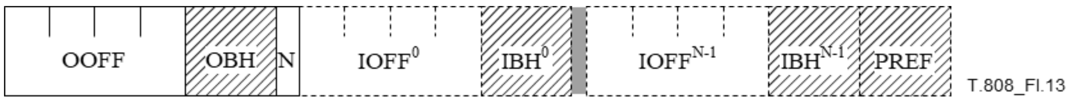

Figure I.13 – Organization of the contents of a Proxy box

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 *i*-th 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.3.4 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):

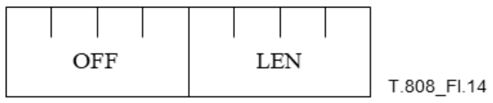

Figure I.14 – Organization of the contents of an Index Finder box

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.4 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.5 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 might 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 can easily be found by file readers. To minimize 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 Rec. ITU-T T.801 | ISO/IEC 15444-2) could be used as a placeholder for it in a yet-to-be-indexed file.

Annex J

Profiles and variants for interoperability and testing

(This annex forms an integral part of this Recommendation | International Standard.)

J.1 Introduction

This annex provides the framework, concepts, and methodology for establishing interoperability, and the criteria to be achieved to claim compliance with this Recommendation | International Standard. This annex also provides a methodology for testing compliance within a set of defined profiles and variants. The objective of standardization in this field is to promote interoperability between JPIP servers and clients and to enable testing of these systems for compliance to this Specification.

This annex also defines profiles and variants. Profiles define the fields that a JPIP server is expected to implement and support beyond parsing and interpretation; profiles also limit the requests a client can expect a server within this profile to support and fully implement. Clients making a request within a profile that receive a 501 ("Not Implemented", see Annex D) or 400 ("Bad Request") error code can use this as an indication that the server does not fully implement the profile and can fall back to requests of a lower profile. Variants define which features of the JPIP specification are used to request and transmit data between client and server. Profiles and variants are orthogonal to each other. Servers are classified according to the highest level profile they support, and all the variants they implement. Clients are classified according to all the variants they implement, and according to the highest level profile they can work with.

Even though the testing procedures, profiles and variants compiled in this annex are defined for images encoded in only some parts of the Rec. ITU-T T.8xx | ISO/IEC 15444-x family of Recommendations | Standards, and only a limited subset of features of JPIP is tested, this shall not imply that servers or clients using means of JPIP to deliver images in other formats or by other means of JPIP not listed here are not compliant. It only means that their compliance is not defined within the limits of this annex, and that there is currently no recommended testing policy and classification for them.

J.1.1 Profiles

Profiles define which requests a server can be expected to support, and therefore, which requests a client can expect to be supported and fully implemented by the server. Requests defined in a lower profile are also supported and fully implemented in a higher profile. Profiles are defined in detail in J.3.

J.1.2 Variants

Variants define which means of the JPIP specification a client and a server use to transmit data. Clients and servers need to provide a common subset of variants in order to interoperate. Variants are defined in J.2.

J.2 Definition of variants

JPIP allows for three different image return types, for requests within a session or stateless communications and for the exchange of metadata and/or codestream data between the server and client. Variants classify clients and servers in a 3-dimensional space based on:

1) the image return types they support;
2) whether the client requires and the server implements a persistent cache model for requests within sessions and/or whether the communication is stateless (see B.1);
3) whether the transmitted data includes codestreams and/or metadata encoded in the boxes of the file format.

To classify a client or a server, the implemented variants in each of the three axes are specified. Interoperability of a client-server pair requires that both operate according to a common subset of variants. Unlike profiles that are ordered by complexity, variants do not form a hierarchy of features.

J.2.1 Image return type variant (P, T or R)

This parameter defines the image return type a server is able to deliver and a client is able to interpret. Servers in the **P** variant are able to deliver JPP streams; servers in the **T** variant are able to deliver JPT streams; servers in the **R** variant are able to deliver "raw" image return types. Clients in the **P** variant accept JPP streams, clients in the **T** variant accept JPT streams and clients in variant R accept raw images. The **P, T** and **R** variants are not mutually exclusive; servers or clients can support several variants.

J.2.2 State model variant (N or S)

This parameter defines whether a server is able to use channels for communication. A server in variant **S** is able to grant a channel in response to a `New Channel` request (see C.3.3) and maintain a persistent cache model between requests in the channel. A server in variant **N** is able to respond to requests that do not involve a new channel or a channel ID request field.

Clients operating in variant **S** are required to cache data between requests in the same session to iteratively request data from a server; for efficient ongoing communications, clients in variant **N** might need to use cache model manipulation requests. The **S** and **N** variants are not mutually exclusive, and servers and clients might support both variants.

J.2.3 Bitstream variant (M or C)

This parameter defines the types of logical targets the server is able to serve. Servers operating in variant **M** are able to deliver original box contents of a JPEG 2000 file format as metadata-bins. Servers in variant **C** are able to deliver data contained in the codestream using the incremental codestream representation. A server in variant **C** shall deliver at least metadata-bin #0 (see A.3.6), though this bin will be empty for a logical target consisting of a codestream only. Clients in variant **C** accept at least incremental codestream representation of the image data; clients in variant **M** accept at least metadata-bins. The **M** and **C** variants are not mutually exclusive, and servers and clients might support both variants.

Table J.1 provides a summary of the requirements of the variants.

NOTE – Servers or clients might be in variant **M** only, in which case they can only return or accept metadata-bins, but no precinct or tile databins. This type of server might be useful to quickly scan for image metadata in a database of images. Clients in variant **M** only (and not in **C**) should include "`meta:orig`" in the *client-preferences* field to restrict responses to metadata-bins only. Clients in variant **C** only (and not in **M**) can use the "`src-codestream-specs`" of the "`subtarget`" field to indicate servers to construct logical targets consisting of codestreams only, see C.2.3.

Table J.1 – Defining requirements of variants

Variant	Server requirements	Client requirements	Remarks
P	Shall implement the image return type "`jpp-stream`"	Shall be able to parse jpp-streams	P, T and R are not mutually exclusive. Clients and servers can implement several variants.
T	Shall implement the image return type "`jpt-stream`"	Shall be able to parse jpt streams	
R	Shall implement image return type "`raw`"	Shall be able to handle raw incoming data	
N	Shall implement additive explicit cache model manipulation requests using byte counts fully in profile 1 and up.		Variants N and S are not mutually exclusive and clients or servers can implement both.
S	Shall implement *cnew, cclose* and *cid* fields fully. Shall implement a persistent cache model.	Shall generate *cnew* field to establish sessions. Shall be able to cache data between multiple requests.	
C	Shall be able to transmit image data in an incremental codestream representation. Shall implement the behaviour of *client-preferences* "`meta:incr`".	Shall be able to parse an incremental codestream representation	Variants **M** and **C** are not mutually exclusive. A server or client can implement more than one variant at once. For the most efficient communications, servers should implement **C** in the first instance, supplemented by **M**. A server operating in **M** only (and not **C**) might not be able to efficiently respond to view-window limiting requests apart from those which select codestreams. Clients interested in retrieving image data should at least implement variant **C**.
M	Shall be able to transmit metadata and image data as boxes. Shall implement the *metareq* field in profile 1 and up. Shall implement the behaviour of *client-preferences* "`meta:orig`".	Shall be able to parse metadata-bins, including image data encoded as JPEG 2000 codestream boxes and placeholder boxes.	

J.3 Definition of profiles

Profiles define the set of request fields a server is expected to implement and support. An overview of the request fields per profile is given in Table J.2. Generally, higher profiles require the server to support more advanced technology of the standard. A client generating requests from a lower profile can expect responses satisfying the request from a server that belongs to an equal or higher profile.

Profiles provide a mechanism for clients to adapt their requests to the capabilities of the server. For this to be successful, servers shall provide sufficient indication of their inability to satisfy a particular request within a profile. Upon discovering that a server is unable to serve a particular request within a profile, a reasonable strategy for a client would be to restrict future requests to those in a lower profile. The server can indicate its inability to satisfy a particular request as given by the client by either issuing an error return code or, where applicable, by modifying the request and issuing appropriate JPIP-response headers (see Annex D).

J.3.1 Profile 0: "Basic Communication"

This profile provides a mechanism for basic communication of a request by a client and a response by a server. Only basic operations on JPEG 2000 Part-1 codestreams or files are supported. This covers delivery of image regions or whole images fitting to a particular display window size. Cache manipulations are not allowed in the request stream for profile 0. The only fields the server is expected to support in profile 0 are:

target, type, target id, frame size, region offset, region size, len, pref (with restrictions), (all variants)

cid, cnew, cclose (additionally, in variant **S**)

The client is required to be able to parse the data returned by the server and is required to handle JPP, JPT or raw image return types according to their variant. Servers cannot be expected to honour the request for extended precinct or tile databins, i.e., the `"ptype"` or the `"ttype"` field defined in C.7.3, Table C.4. Conforming clients shall accept unaligned messages; servers shall not be required to honour the `"align"` request. The only client preference request *pref* that a server is required to satisfy within this profile is `"concise"`, see C.10.2.8, and `"fullwindow"`, see C.10.2.2. Servers in profile 0 variant **S** shall implement the fields *cid, cnew* and *cclose*, but only need to support a single channel per session.

J.3.2 Profile 1: "Enhanced Communications"

Profile 1 extends profile 0 by requiring that servers support cache model manipulation requests, and the request can be limited by layers or components. Depending on the profile variant, the cache model is either explicitly communicated to the server by cache model requests in variant **N** or implicitly expected to be implemented by the server in variant **S**. Profile 1 also extends Profile 0 to include the following additional fields:

components, layers, wait, model (with restrictions, see text), (all variants)

metareq (additionally, in variant **M**)

Profile 1 servers are also expected to handle additive cache model manipulation requests with explicit bin addressing and byte counts, i.e., the *model* field using explicit bin descriptors as defined in C.8.1.2. As in profile 0, servers in profile 1 variant **S** shall implement the *cid, cnew, cclose* fields, but only a single channel per session needs to be supported. Servers in profile 1 variant **M** shall additionally implement the *metareq* field.

J.3.3 Full profile

The full profile provides capabilities beyond all lower profiles up to everything specified in this Recommendation | International Standard.

Table J.2 – Set of fields included in each profile

Profile		0:Basic Communications	1:Enhanced Communications	Full profile
Server support field	target	yes	yes	yes
	subtarget			yes
	fsiz	yes	yes	yes
	roff	yes	yes	yes
	rsiz	yes	yes	yes
	comps		yes	yes
	layers		yes	yes
	len	yes	yes	yes
	tid	yes	yes	yes

Table J.2 – Set of fields included in each profile

Profile		**0:Basic Communications**	**1:Enhanced Communications**	**Full profile**
	metareq		yes	yes
	ptype=ext, ttype=ext			yes
	align			yes
Multi-channel	cnew	yes (only one session)	yes (only one per session)	yes
	cid	yes (only one per session)	yes (only one per session)	yes
	cclose	yes	yes	yes
Pre-emptive	wait		yes	yes
	qid			yes
	stream			yes
	context			yes
	implicit model			yes
	tpmodel			yes
	mset			yes
Cache management	model		explicit, byte counts and additive only	All
Server Control Request fields	align			yes
	type	jpp-stream jpt-stream raw	jpp-stream jpt-stream raw	All
Client Preferences	pref	concise fullwindow	concise fullwindow	All

J.4 Testing methodology

JPIP interoperability testing is performed at databin level, i.e., on the data transmitted from server to client. This clause provides a set of Rec. ITU-T T.800 | ISO/IEC 15444-1 example images and example JPIP requests, along with corresponding example response headers and data from the server. The response data from the servers are in the form of jpp-files and jpt-files, as described in A.5.

J.4.1 Server conventions required for testing

All example streams have been created with the *conciseness-pref* client preference set to `"concise"` and the *view-window-pref* set to `"fullwindow"`. Resource constraints at the server might require a realistic server implementation to restrict requests that require too much data from the server at once. The example data provided with this clause should not trigger such conditions for implementations targeted to state-of-the-art desktop computers. If a server implementation includes means to restrict a request to limit resources below what is needed to perform the test, this type of processing needs to be disabled for the purpose of compliance testing. This clause does not introduce any resource bounds for the environment in which a server is expected to operate.

J.4.2 Server testing

Server testing is specific to a given variant and profile combination. Testing a JPIP server is performed by initiating JPIP requests for data from an example image and comparing the data delivered by the server under testing with the example responses according to the variant and profile using the algorithm described in J.4.3. To claim compliance to a specific profile and variant, the server shall pass all server tests for this profile and all lower profiles using the specified variant.

Although JPIP servers are allowed to modify what they deliver, the example streams shall not be modified by the server during testing.

NOTE 1 – Example streams for the JPP image return type have been created in such a way that each precinct of the example stream consists of only one codeblock in each sub-band. Therefore, any transcoding is unnecessary, though insertion or removal of COM, PLT, PLM and TLM markers is allowed within the context of this testing procedure.

For the purpose of testing, the JPIP data that the server would send over a network shall be saved in the appropriate jpp- or jpt-file format. Any wrapper such as HTTP response codes is excluded from these files.

The EOR codes concluding the communication shall be included in the file. The contents of these files are then compared with the contents of the example responses. The response returned by a compliant server might, however, differ from the example response.

Per the test procedure defined in J.4.3, the following differences with the example stream are allowed:

– Reordering of the databins within the response.

– Relocating the contents of metadata boxes into placeholder boxes.

– Using equivalent encodings for the VBAS headers of the databins.

– Using a different break-up of metadata into placeholder bins, provided that the requested metadata is included in the response.

– Using stream-equivalent representations of metadata.

A normative definition of which differences between the streams are acceptable is implicitly given by the test procedure in J.4.3.

NOTE 2 – A test tool ("vbasdiff.py") is provided in an electronic attachment to this Recommendation | International Standard that can be downloaded from https://handle.itu.int/11.1002/2000/7460 or https://standards.iso.org/iso-iec/15444/-9\ed-2/en along with example test data. The tool implements this test procedure and analyses the difference between two server responses.

NOTE 3 – Servers can additionally provide stream equivalent representations; however, determining their correctness is outside the scope of this annex; they are ignored by the testing methodology defined in J.4. Furthermore, testing for JPIP conformance when applied to data that requires other placeholder flag values, for example as used in Rec. ITU-T T.802 | ISO/IEC 15444-3, is outside the scope of this annex.

J.4.3 Comparing server responses

This clause defines a normative algorithm that compares the response data of the server under testing with the example responses provided by this Recommendation | International Standard.

Included with the test stream set is a python test script ("vbasdiff.py") to perform this comparison implementing the algorithm described in the following: The nature of the test depends on the existence of the length field. If a *len* field is present in the request, test only the size of the server response and the EOR code. Otherwise, perform the following four steps defined in detail later:

1) Parse the messages in the jpp- or jpt-files, testing their encoding and assigning them to bins.
2) Bring metadata-bins into canonical form as defined in J.4.3.3.
3) Measure the amount of excess data.
4) Compare the fraction of excess data to total data with a threshold.

J.4.3.1 Comparing the size of the server response

In case the request includes the *len* field, compare the length in bytes of the server response excluding any EOR message with the requested length. If the size in bytes of this output is larger than the requested length, the comparison fails. If no data except EOR was returned, the comparison fails. Then compare the EOR code of the tested stream with the EOR code of the example stream. If the EOR code of the tested stream is neither 1 ("Image Done") nor 2 ("Window Done", see Table D.2) nor equal to the EOR code of the example stream, the comparison fails. The EOR message body shall be ignored.

NOTE – The example streams contain similar requests with and without the *len* field.

J.4.3.2 Parsing stage

For test requests that do not include the *len* field, test and example data are parsed. The VBAS message headers shall be decoded, first separating EOR messages from regular messages, and for regular messages, identifying the in-class identifier, the message class, the codestream sequence number (CSn), the "final message bit" (bit 4, labelled "c" in Figure A.3), the AUX value, if present, and the message offset and length. The message body shall then be inserted into databins according to the message offset and message length field of the message header, where each bin is identified by the triplet of bin class, in-class identifier and codestream sequence number. The mapping between message class and bin class is given by Table A.2. It shall be acceptable if a later message replaces parts of the data delivered by a former message, but it is not acceptable to deliver data that enlarges bins for which a final message has been received already.

First, EOR codes are compared. If the EOR code of the tested stream is neither 1 ("Image Done") nor 2 ("Window Done", see Table D.2) nor equal to the EOR code of the example stream, the comparison fails. The EOR message body shall be ignored.

AUX values need either to be included in all messages contributing to a databin, or to be included in none. Otherwise, the file is ill-formed and the comparison fails. For each databin containing messages with AUX-values, the following algorithm is used to assign an AUX value to the databin:

- For precinct databins, the AUX value of the bin shall be the maximum of all AUX values found in all messages contributing to the bin.
- For tile databins, the AUX value of the bin shall be the minimum of all AUX values found in all messages contributing to the bin.

  NOTE – Messages containing AUX values do not occur in testing for profile 0 and 1.

J.4.3.3 Abstracting from the metadata-bin layout

In the next step, metadata-bins are brought into a canonical form in order to abstract from the particular way a server broke up metadata into placeholders. The messages contributing to a databin might not define all of its data; it can happen that messages in the stream only define databins partially, and that it is acceptable that metadata-bins contain "holes" of missing data that have been relocated by the placeholder mechanism. Such regions are referred to as "missing bytes" in the following.

The test script performs the following algorithm to reconstruct intermediate data from a jpp- or jpt-file:

- Metadata-bin #0 is scanned in the test stream for incomplete box headers. Test and example streams are then made comparable by marking corresponding ranges as missing data in both streams. The modified streams are then checked for placeholder boxes. If the flags value of a placeholder box has its LSB set, indicating that the OrigID is valid, the placeholder box will be handled as indicated in the next steps; otherwise, it will remain in its unmodified form:
  - If the databin referenced by the placeholder box is included in the stream, the box will be replaced by the box header and bin contents referenced within.
  - If the databin referenced in the placeholder box is not included in the stream, the placeholder box will be removed, the box header in the placeholder box will be inserted into the stream, and the byte range in the missing target bin will be marked as missing data.
- After metadata-bin #0 of test and example streams have been parsed as above, all remaining metadata-bins except bin #0 are removed from the stream.

The above algorithm requires that the software performing the comparison contains a database describing which boxes are superboxes and which are plain boxes. This knowledge is required to be able to scan superbox contents correctly for placeholder boxes. In the test procedure, it is of advantage not to include excess data in the response. Servers should use placeholders as appropriate to avoid excess data. For profile 1 and above, every superbox has been replaced by a placeholder box in the example streams. Example streams for profiles 0 have been created without placeholder boxes and only the minimal amount of metadata, as defined in C.5, is present.

J.4.3.4 Comparing databins

In the third step, all remaining databins shall be compared, locating for each bin-class, bin-Id and CSn value in one stream the corresponding bin in the second stream: a metadata, tile or main header databin needs to be present in the test stream if and only if it is present in the example stream; otherwise, the comparison fails. An (extended or regular) precinct or tile databin needs to be present and non-empty in the test stream if it is present and non-empty in the example stream. An (extended or regular) precinct or tile databin either empty or non-present in the test stream needs also to be either empty or non-present in the example stream. If these conditions are not met, the comparison fails.

NOTE – According to the clause above, it is equivalent not to transmit any message for a precinct or tile databin, or to create a message with no payload data. That is, even in concise mode JPIP servers have the freedom to signal tile or precinct databins that contain no significant image data by transmitting messages of length zero – and thus to potentially increase the size of the overall stream. By contrast, this freedom does not exist for main header, tile header or metadata bins. See A.3.6.1 for additional details about metadata bins.

Databins are compared as follows:

- If one of the databins carries AUX values, the other bin needs to carry AUX values. If both bins carry AUX values, they need to be equal. Otherwise, the comparison fails. See J.4.3.1 for information on how to compute the AUX value of a bin from the AUX values of the messages contributing to the bin.
- If the example databin contains a message that indicates that the "last" byte of the bin has been included, the corresponding databin in the stream under testing needs also to contain a message with such an indicator. Otherwise, the comparison fails.
- For all bin types except the main header databin and tile header databins, all defined bytes in the databins being compared need to compare equal. Otherwise, the comparison fails. The number of excess bytes in the test stream but not in the example stream is to be summed up.

– The main header databin and tile header databins are compared by decomposing them into marker segments and comparing the marker segments independent of their order. All marker segments except COM, PLT, PLM and TLM need to compare equal.

After comparing all databins, the amount of excess bytes N_e measured in step three above shall be divided by the total number of bytes in all bins N_t. If this quotient is above a threshold T, the comparison fails. For the example requests and example responses currently defined in this Recommendation | International Standard, the threshold T shall be zero.

In order to test servers for compliance as defined in this annex, servers should operate in accordance with the `concise` clause of the *pref* field, see C.10.2.8. Servers that do not operate in this way might still be compliant to this Recommendation | International Standard, but their testing is beyond the scope of this annex.

J.4.4 Client testing

Client testing is specific to a given variant. Compliance of clients shall be tested by feeding a provided example response header and a jpp- or jpt-file for a particular variant and profile to the implementation under testing. The client then processes this response. For the purposes of testing, the clients shall create files or codestreams compliant to Rec. ITU-T T.800 | ISO/IEC 15444-1. This feature is not a mandatory requirement for a client to be compliant, but it is required to perform testing. The created codestream or file shall then be compared with the example codestream or file provided in this clause using the algorithm defined in the following. To claim compliance to variant, the client shall pass all client tests for the specified variant.

Comparing the example streams with the streams generated by the client is performed in two stages: first comparing metadata if it is present, and second comparing image data when available.

NOTE – The client testing procedure described here only tests the ability of clients to parse jpp or jpt streams successfully, and to regenerate a JPEG 2000 compliant file or codestream from such data. It does not test the ability of clients to create requests or exploit other capabilities offered by this Recommendation | International Standard.

J.4.4.1 Comparing metadata

If the target is encoded in a JPEG 2000 file format, the contents of the boxes of the example file and the contents of the boxes except for the codestream box(-es) generated by the test implementation are compared. The client is, however, allowed to perform the following modifications:

– Include additional UUID boxes not present in the example stream.

– Reorder the boxes, provided this does not change the semantics of the file.

Exclusive of these modifications, the box contents of test and example streams need to be identical. Otherwise, the comparison fails.

J.4.4.2 Comparing reconstructed image data

If the request that was used to generate the example jpp- or jpt-file included a request-field for a non-empty view-window, the reconstructed image data shall be compared. The example stream and the codestream generated by the client implementation are both decoded with a conformant JPEG 2000 decoder. The same implementation shall be used for both streams. The resulting images need to be identical on a pixel by pixel basis within the view window of the request, or else the comparison fails. Comparison in this stage is to be performed as follows:

– Set `fx'=Xsiz-XOsiz` and `fy'=Ysiz-YOsiz` where `Xsiz,XOsiz` and `Ysiz,YOsiz` are taken from the SIZ marker of the relevant codestream.

– Set `ox,oy` and `sx,sy` to the region offset and region size of the view window that had been defined in the request, and set `fx` and `fy` to the frame size that had been defined in the request.

– The region size `sx'` and `sy'`, and offset `ox'` and `oy'`, associated with the codestream image region are then determined by:

$$\begin{aligned}&\mathrm{ox}'=\left\lfloor \mathrm{ox}\cdot\frac{\mathrm{fx}'}{\mathrm{fx}}\right\rfloor;\quad \mathrm{oy}'=\left\lfloor \mathrm{oy}\cdot\frac{\mathrm{fy}'}{\mathrm{fy}}\right\rfloor;\\ &\mathrm{sx}'=\left\lceil (\mathrm{sx}+\mathrm{ox})\cdot\frac{\mathrm{fx}'}{\mathrm{fx}}\right\rceil-\mathrm{ox}';\quad \mathrm{sy}'=\left\lceil (\mathrm{sy}+\mathrm{oy})\cdot\frac{\mathrm{fy}'}{\mathrm{fy}}\right\rceil-\mathrm{oy}'\end{aligned}\qquad\text{(J-1)}$$

– Compare all pixels in the reconstructed images constrained to the view window having the left, top image corner `ox'` and `oy'` and having the dimensions `sx'` and `sy'`. All pixels within this region need to be identical; otherwise, the comparison fails.

NOTE 1 – The above procedure is similar, but not identical to that defined in C-4, Equations C-1 and C-2. The difference is that the resolution level *r* in Equation C-1 is here constrained to be zero, enforcing the comparison at the full image resolution.

NOTE 2 – A tool ("jp2file.py") is provided in the electronic attachment to Rec. ITU-T T.804 | ISO/IEC 15444-5 that generates a textual representation of the contents of JPEG 2000 files or codestreams. This tool can be used to ease the analysis of the data generated by the client. A tool with the same name ("jp2file.py") might also be found in the electronic attachment to this Recommendation | International Standard, but the tool in the electronic attachment to Rec. ITU-T T.804 | ISO/IEC 15444-5 should be preferred, to the extent that they differ.

NOTE 3 – Although this entire client test procedure needs vendors to provide a JPEG 2000 conformant decoder implementation, this is not required for conformance to this Recommendation | International Standard. The ability of a client to create a JPEG 2000 file or codestream is also not required to be compliant to this Recommendation | International Standard, but is required only for performing the tests of this annex.

Annex K

Using JPIP with HTTP requests and UDP returns

(This annex forms an integral part of this Recommendation | International Standard.)

K.1 Introduction

This annex provides details of the "http-udp" transport, which is identified in this text as HTTP-UDP. The HTTP-UDP transport uses 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 are delivered as UDP datagrams over an auxiliary UDP connection, as defined in RFC 768. The information transported on this auxiliary UDP 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 and a record of the Request-id associated with the corresponding client request. Each chunk shall contain a whole number of JPIP messages and at most one EOR message as defined in Annex A. Message class identifiers and codestream sequence numbers shall be present in at least the first JPIP message of each data chunk.

NOTE – Since the UDP transport is not reliable (i.e., UDP packets might be dropped or out of order) clients might not receive all data chunks corresponding to a request. Clients can use the Abandon request field to explicitly inform the server of this condition, see C.7.6.

K.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. Client requests that are issued within an HTTP-UDP transported JPIP channel shall include the Request-id (qid) request field.

Clients should issue requests with consecutive Request-id values.

K.3 Response data delivery and channel establishment

A new channel can be established to a JPIP server by issuing a request that includes the New Channel request field (see C.3.3). 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 in D.2.3) indicates that a new channel has been created to work with the HTTP-UDP transport, the request which included the New Channel request field is treated as though it had been issued within the newly created HTTP-UDP transported channel. This ensures that the response data for that request and all subsequent requests in the same channel is framed into data chunks and delivered as UDP datagrams. This response data cannot be empty, since every request issued within an HTTP-UDP transported channel shall have a response data stream that consists of at least the EOR message (see D.3).

The destination to which response datagrams are delivered depends upon whether or not the associated request contains a Sendto request field.

1) For requests that contain a Sendto request field, the datagrams is delivered to the specified address without any explicit acknowledgement by the client.
2) For requests that do not contain a Sendto request field, the response datagrams cannot be delivered until an auxiliary UDP connection has been established. To do this, the client sends one or more connection establishment datagrams to the server host identified via the New Channel response header, on the port identified by the New Channel response header. Each connection establishment datagram commences with a four byte header, which is followed by the channel-id string associated with the new HTTP-UDP channel. Once the server receives a connection establishment datagram with the correct channel-id string, it sends all subsequent response datagrams (other than those associated with Sendto request fields) to the IP address and port from which the channel establishment datagram arrived and it expects to receive acknowledgement datagrams from the same client IP address and port.

NOTE – Since UDP is an unreliable transport, connection establishment datagrams might be lost. For this reason, clients might need to send multiple connection establishment datagrams, and servers might need to discard superfluous connection establishment datagrams which could arrive. Once a valid connection establishment datagram has been received, the server can choose to filter incoming datagrams as part of an overall defence strategy.

The first two bytes of a connection establishment datagram's header shall be FF, while the third and fourth bytes identify the length of the channel-id string, recorded as a 16-bit big-endian quantity. The four byte header is followed by the channel-id string, encoded as UTF-8 characters. Additional content might be provided, beyond the end of the channel-id string, but the interpretation of such content is unspecified by this Recommendation | International Standard.

K.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-UDP response. Neither shall the "Content-length:" or the "Transfer-encoding:" HTTP response headers be used.

The response data itself are delivered via UDP, framed into chunks in the manner described in K.5. Since the HTTP-UDP transport can only be used with sessions, the image return type is constrained to JPP-stream and JPT-stream as defined in Annex A. Thus, 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 is permitted to contain response data generated in response to two different requests.

The response to each and every request following the one in which the channel was requested, 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.

This means that every request issued on an HTTP-UDP 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.

K.5 Framing of response data into chunks

All response data sent by the server via the auxiliary UDP 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 response data, as shown in Figure K.1. The chunk header and body are sent as a single UDP datagram, whose length shall be exactly 8 plus the length of the chunk body measured in bytes. Moreover, no UDP datagram shall have a length larger than 4096 bytes or a length smaller than 8 bytes. The first 2-byte word of the chunk header holds an unsigned big-endian integer, whose interpretation as a "control" field is provided in Table K.1.

The remaining 6 bytes of the chunk header contain the least significant 16 bits of the Request ID provided by the client in the request with which the chunk's response data is associated, together with a one-byte "repeat" field and a 24-bit chunk sequence number encoded as big-endian unsigned integer. The chunk sequence number is a number generated by the server, shall start at zero and shall be incremented by one for each subsequent chunk sent to the client in response to the same request.

New requests from the client shall cause the chunk sequence numbering to be reset starting at 0 for the first chunk sent in response to the new request. Chunk sequence numbers do not wrap around, and servers shall indicate an error using the EOR reason code 7 (Response limit reached), see D.3 in the event of running out of available chunk sequence numbers.

The one-byte "repeat" field can be used by the server in any manner desired. Typically, the "repeat" field would be used to distinguish between original and retransmitted versions of a data chunk, allowing the server to determine which instance of a chunk is being acknowledged within an acknowledgement datagram. However, the field can potentially be used in other ways. Clients should not attempt to interpret the "repeat" field but shall reproduce it within returned acknowledgement datagrams.

NOTE – The Request ID and chunk sequence number allow for chunks of data to be properly reassembled in order. They also provide a means for dropped chunk detection. Clients can detect the loss of chunks by examining the set of chunk sequence numbers for gaps or by detecting that the server has not transmitted a chunk containing an EOR message; the latter will always be included in the last chunk sent in response to a request. Clients can use the Abandon request field to explicitly inform the server of missing chunks. Clients can choose to defer the use of these mechanisms or not to use them at all, at their discretion.

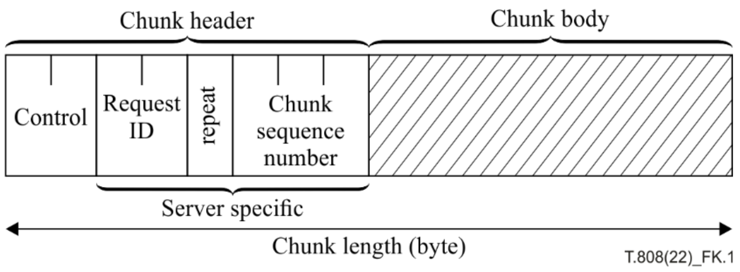

Figure K.1 – Response data structure on http-udp connection

Table K.1 – Interpretation of the "control" field in each data chunk header

"control" field value	Interpretation in response chunk headers	Interpretation in acknowledgement chunk headers
`0000 xxxx DDDD DDDD`	Maximum time that the server would prefer the client to wait from the time it receives this data chunk before acknowledging the chunk's arrival in an acknowledgement datagram for the first time is given by $2^{(D/8)}$ microseconds, where D is the unsigned integer represented by the second byte of the control field. This interpretation does not apply if the corresponding request included a Sendto request field with the value "no".	Estimated time between client receipt of the data chunk and delivery of the acknowledgement datagram, expressed as $2^{(D/8)}$ microseconds.
`0000 AAAA xxxx xxxx`	Acknowledgement repetitions A, in the range 0 to 15. The server would prefer the client to acknowledge the arrival of this data chunk at least A+1 times, in A+1 separate acknowledgement datagrams. The server would prefer all A+1 acknowledgement datagrams to be sent within the time given by the D parameter, as defined above. If the corresponding request included a Sendto request field, the value of A has no meaning, since the client sends no acknowledgement datagrams in that case.	Number of previous datagrams in which the client has already acknowledged the receipt of this data chunk.
`1111 1111 1111 1111`	Reserved for ITU/ISO use	Connection establishment datagram
`Other values`	Reserved for ITU/ISO use	Reserved for ITU/ISO use

K.6 Client acknowledgement of server responses

For response datagrams issued in response to requests containing a Sendto request field, there is no explicit client acknowledgement, although servers might deduce the arrival or non-arrival of data chunks using information provided via Barrier and/or Abandon request fields in subsequent requests.

In all other cases, clients are expected to acknowledge the successful arrival of data chunks by sending acknowledgement datagrams back to the server. UDP acknowledgement datagrams are sent to the same host address and port as the connection establishment datagram. Moreover, clients shall send acknowledgement datagrams from a socket bound to the same local address and port as that used to send the connection establishment datagram. Each acknowledgement datagram consists of one or more chunk headers from received data chunks, except that the "control" field is modified as follows. The 8-bit D value in the returned chunk header's "control" field should be modified to reflect (at least approximately) the number of microseconds between client receipt of the data chunk and delivery of the acknowledgement datagram, expressed as $2^{(D/8)}$ microseconds. The 4-bit A value in the returned chunk header's "control" field should be modified to reflect the number of previous datagrams in which the client has already acknowledged the receipt of the data chunk in question. This information is reflected in Table K.1. No acknowledgement datagram shall be more than 512 bytes. That is, no acknowledgement datagram shall contain more than 64 chunk headers. Connection establishment datagrams are distinguished from acknowledgement datagrams on the basis of the first 4 bits of the control field, as shown in Table K.1.

Even if the client has already identified a data chunk via an Abandon request field, if that chunk is subsequently received, the client can acknowledge its arrival via an acknowledgement datagram; this is generally advisable as it can reduce redundant transmission of information from the server. In this event, however, the client is expected to update its cache accordingly. That is, the client shall not acknowledge data chunks which it discards, since then the server's log of what the client should have received might contain erroneous entries.

Clients can acknowledge the same data chunk multiple times in separate acknowledgement datagrams; in the event of multiple acknowledgements, the modified D and A values will generally be different. Clients need not send a separate acknowledgement datagram for each received data chunk, but they should endeavour to acknowledge data chunks within the period specified by the D value in the chunk header's "control" field.

NOTE 1 – Acknowledgement can be used by servers for flow-control purposes. Note further that once a data chunk has been acknowledged by a client, subsequent Abandon request fields that refer to already acknowledged chunks might be ignored by the server. In some server implementations, this means that the receipt of acknowledgement information allows the server to release temporary storage resources that might be needed to maintain cache model consistency.

NOTE 2 – Notwithstanding the guidelines presented above, it is acceptable for a client to promptly return a single acknowledgement datagram for each received data chunk, containing just the one chunk header, with the "control" field set to 0. The D values are intended primarily to allow server flow control algorithms to take into account the additional delay which might be incurred where a client chooses to aggregate data chunk acknowledgements into a smaller number of acknowledgement datagrams. The A values are intended to allow a server to estimate loss probability for acknowledgement datagrams and feed suggested repetition counts to the client to increase the robustness of the acknowledgement mechanism.

K.7 UDP and Maximum Response Length Field (informative)

There might be little or no reason for using the Maximum Response Length field with a UDP return channel, unless the Sendto request field is also being used. Apart from this case, the server should be able to use the times at which acknowledgement datagrams arrive to regulate the flow of response data to the client, so as to maintain responsiveness. If the Sendto request field is used, however, the server does not receive continuous feedback from the client and can easily push a great deal of data over the channel. To maintain responsiveness or avoid excessive loss in these circumstances, clients should use the Maximum Response Length field (see C.6.1) to regulate the flow of traffic, much as they would with the HTTP transport.

K.8 Implementation strategies for acknowledged communication (informative)

Response data chunks delivered in response to requests that do not contain the Sendto request field are acknowledged via explicit acknowledgment datagrams. This model is quite common in network communication protocols and facilitates the implementation of flow control management within the server. Although not required by this Recommendation | International Standard, servers are recommended to adopt a retransmission strategy, in which data chunks that have not been acknowledged after an appropriate period of time are retransmitted, unless the server is able to determine that the data chunks are no longer relevant to the client – e.g., due to a change in the client's window of interest. Typically, the retransmission would stop, once a data chunk is either acknowledged or abandoned by means of an Abandon request field.

Clients cannot expect servers to retransmit data chunks that are not acknowledged. Moreover, the client cannot generally have any guarantee of the point at which a server might decide that an unacknowledged data chunk has not arrived at the client. This is important, since it can affect the interpretation of responses to future requests. For example, the response to a future request might include an EOR message with the "Window Done" reason code, yet the client cannot be sure that it has indeed received all relevant content if an earlier request with overlapping Window of Interest still has outstanding missing data chunks. To avoid the ambiguity which might be created by such situations, clients can use the Abandon request field to explicitly abandon missing data chunks from requests that have not received any content for some time. This allows the client to be sure that the server will eventually include any relevant content that was missing from those requests in its response to future requests.

If a client does not need to rely upon the reason codes supplied by EOR messages (e.g., because the client can directly compute whether its cache contents contain a complete response to a requested Window of Interest), there might be no need for it to consider use of the Abandon request field.

Clients should bear in mind that aggressive use of the Abandon request field can significantly increase server workload. For example, a typical server might generate data chunks in batches which are then scheduled for transmission. If a client automatically issues Abandon request fields referring to all previous requests whenever its Window of Interest changes in any way, the server might discard many of the data chunks it has generated without transmitting them at all. Although this does not break interoperability, the server might have to regenerate content many times over during an interactive session. To avoid this problem, it is recommended that clients wait until at least one data chunk is received from a request A before issuing an Abandon request that refers to data chunks from an earlier request B, unless no data chunks are received at all for a considerable period of time. It is generally possible for the server to make better decisions than the client regarding data chunks which should not be transmitted because the client's Window of Interest has changed.

The Barrier request field is not primarily intended for use with acknowledged communication. Nevertheless, if this request field is used by a client, it provides a supplementary mechanism to implicitly acknowledge the arrival of data chunks that have not been abandoned – it can be that acknowledgement datagrams for some of these have been lost. Ignoring a Barrier request field does not damage interoperability in JPIP but might lead to some redundant transmission, so servers are recommended to implement this feature.

K.9 Implementation strategies for unacknowledged communication (informative)

Response data chunks delivered in response to requests that contain the Sendto request field are not acknowledged via acknowledgement datagrams. As with all JPIP communications, the server can assume that data which it has sent to the client arrive and are cached by the client, unless and until it learns otherwise. Without this assumption, the server would find itself transmitting the same content over and over again in a typical interactive session. Since UDP is an unreliable transport medium, the server needs to be prepared for the possibility that data chunks are actually lost. In particular, it needs to be prepared to process Abandon request fields.

A typical server would keep a record of the data-bin byte ranges that are contained within each data chunk that has been sent to the client. If the data chunk is abandoned, the record can be erased and the server should remove the relevant data-bin byte ranges from its log of content the client has received (i.e., its client cache model); this content can be delivered again in response to future requests, if deemed relevant. Since Abandon requests provide only a mechanism for the client to declare that it has not received some data chunks, if no other steps are taken by the client, the servers record of data chunks that have been sent and not yet abandoned could grow indefinitely; as a result, a typical server would eventually need to internally abandon data chunks, so that everything gets abandoned in the long run. This would eventually cause a great deal of redundant transmission, but clients can avoid the problem using one of the following two strategies:

1) A client implementation can arrange to send additive cache model manipulation statements that directly add all received data-bin byte ranges to the server's cache model. This avoids redundant computation, so long as servers do not accidentally erase this information again while abandoning the relevant data chunks at a later point. To avoid such possibilities and reduce the burden on servers, clients are strongly recommended to use the Abandon and Model request fields together, to declare all data chunks from a given previous request abandoned but simultaneously add all the data-bin byte ranges from the arrived data chunks to the server's cache model via the Model request field. More generally, it is preferable for clients to abandon data chunks explicitly rather than leave servers to do so implicitly at an undefined point in the future; and it is preferable for clients to abandon data chunks during the same or an earlier request to that in which the arrived content is identified via the Model request field. Keeping this in mind, it is preferable for servers to process the effects of an Abandon request field prior to the processing of any Model request field found in the same request.
2) A client implementation can use the Barrier request field to assure the server that no subsequent request will contain an Abandon request field which abandons data chunks belonging to requests prior to a certain point. This effectively acknowledges the successful arrival of all data chunks not yet abandoned from requests prior to the request-id specified by Barrier. To help conserve server bookkeeping resources, clients are recommended to use Barrier regularly. A typical client implementation might abandon all non-arrived data chunks associated with a request which is sufficiently old and simultaneously use the Barrier request field to indicate to the server that there will be no further abandonment for data chunks from that request.

Annex L

Registration of extensions

(This annex does not form an integral part of this Recommendation | International Standard.)

Intentionally left blank

Editorial note – The first edition of this Recommendation | International Standard used this annex to describe a JPIP Registration Authority. However, the registration authority was not used, and the preferred approach by ITU and ISO for adding new features to an existing standard is through amendments and revisions. Consequently, the original contents of the annex have been removed from this edition.

Annex M

Application examples

(This annex does not form an integral part of this Recommendation | International Standard.)

M.1 Introduction

This annex presents some informative examples of aspects of JPIP implementations.

M.2 Use of JPIP with codestreams in other file formats

JPIP can 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 Recommendation | International Standard can be used to locate the JPEG 2000 codestream. JPIP requests and responses can be used on the object once the codestream is located. The Sub-target request field is intended for just such a situation. Alternatively, a server could provide access to the codestreams via a different URL.

M.3 Tile-part implementation techniques

M.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 simple. 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 view-window.

NOTE – Although on the reference grid all tiles have the same dimensions, on the subsampled reference grid, after sub-band decomposition, tiles do not necessarily all have the same dimensions. A tile intersecting the view-window, even a tile contained completely within it, might contribute no samples to the view-window at the lowest resolution levels; however, implementations need not take advantage of this occurrence by omitting the tile altogether from the response.

The resolution level and quality are used to determine the tile-parts needed. The Tile-part Index Table box, if available, can be used to obtain information about the location of tile-parts in the codestream and (if Auxiliary fields are included) the completion of resolution levels within tile-parts. The SOT marker segments also give the tile and tile-part indices and the number of bytes in each tile-part. From the codestream, the appropriate bytes, corresponding to the tile-parts that need to be sent, are transmitted to the client. In case the view-window changes and the corresponding relevant tiles also change, then only relevant tile-parts that have not been sent earlier need be sent to update the display image.

M.3.2 Decoding an image from returned JPT-stream messages

JPIP specifies mechanisms to communicate compressed 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 between applications. This subclause 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), can concatenate that data-bin with complete tile-parts from tile data-bins to form a valid JPEG 2000 codestream. This codestream can be provided to a conformant JPEG 2000 decoder and the result displayed. Of course, for efficiency purposes, a client might 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.

M.3.3 Auxiliary signalling for tile-parts

Tables M.1 and M.2 illustrate the use of Auxiliary fields in extended tile data-bin messages and in the Tile-part Index Table box.

NOTE – In this example, the definition of *r* differs from that used in other places in this Recommendation | International Standard, but is consistent with Rec. ITU-T T.800 | ISO/IEC 15444-1.

Table M.1 illustrates a simple case in which all tile-components of a resolution-progressive tile have the same number of decomposition levels and in which the message boundaries (in the data-bin case) or tile-part boundaries (in the index box case) occur only between each successive resolution level.

Table M.1 – Example of the use of auxiliary fields in a simple case

Message sequence number in data-bin, or tile-part number in tile	Resolution level *r*	*n* = *N*$_L$ – *r*	Auxiliary value
0	0	2	2
1	1	1	1
2	2	0	0

Table M.2 illustrates a more complicated case in which the number of decomposition levels varies by tile-component. A comment is made in the final column of the table on the first occurrence of each new Auxiliary value. This case corresponds to a tile from a three-component image in an RC… progression order, for example the LRCP progression order with a single layer, or the RPCL progression order with a single precinct in the tile. The message boundaries (in the data-bin case) or tile-part boundaries (in the index box case) occur between each component of each resolution level as well as between resolution levels. Components 0 and 1 have two decomposition levels (N_L = 2) and component 2 has a single decomposition level (N_L = 1).

Table M.2 – Example of the use of auxiliary fields in a more complicated case

Message sequence number in data-bin, or tile-part number in tile	Component index *c*	Resolution level *r*	*n* = *N*$_L$ – *r*	Auxiliary value	Comment
0	0	0	2	3	No level complete
1	1	0	2	2	*n* = 2 now complete
2	2	0	1	2	
3	0	1	1	2	
4	1	1	1	1	*n* = 1 now also complete
5	2	1	0	1	
6	0	2	0	1	
7	1	2	0	0	All levels now complete

M.4 Precinct-based implementation techniques

M.4.1 Server determination of relevant precincts for a view-window request

When communication involves the JPP-stream media type, the server translates the client's requested image region into a set of precincts which are relevant to the request. The first part of this process involves translation of the fx, fy, sx, sy, ox and oy parameters supplied by the Frame Size, Region Size and Region Offset request fields, into codestream frame size, region size and offset parameters fx', fy', sx', sy', ox' and oy', for each relevant codestream. This translation proceeds in the same way for both precinct- and tile-based services, and is based on Equations C-1 and C-2, possibly modified according to Equations C-3 and C-4. This subclause describes how a server should determine the precincts which are relevant to the region defined by parameters fx', fy', sx', sy', ox' and oy', within a particular codestream.

Let *r* be the non-negative integer in Equation C-1 which was used by the server to find fx' and fy', based on the client's request. As mentioned in connection with Equation C-1, *r* is most easily interpreted as the number of discarded highest resolution DWT levels, even though *r* is allowed to exceed the actual number of DWT levels which are available for any given tile-component. It is convenient to first map the region described by sx', sy', ox' and oy' onto the codestream's high-resolution grid. This yields a region whose upper left hand corner is given by $(E_1^{\text{reg}}, E_2^{\text{reg}})$ and whose lower right hand corner is given by $(F_1^{\text{reg}} - 1, F_2^{\text{reg}} - 1)$, where:

$$E_1^{\text{reg}} = \text{XOsiz} + 2^r \cdot \text{ox'}, \quad E_2^{\text{reg}} = \text{YOsiz} + 2^r \cdot \text{oy'}, \quad F_1^{\text{reg}} = E_1^{\text{reg}} + 2^r \cdot \text{sx'}, \quad \text{and} \quad F_2^{\text{reg}} = E_2^{\text{reg}} + 2^r \cdot \text{sy'}$$

The server need only consider those tiles which intersect with this region on the codestream's high-resolution grid. For each such tile, the server need only consider those image components which are requested by the client, in the manner described in connection with the Component and Codestream Context request fields. For each considered tile-component, denoted by *t* and *c*, let $D_{t,c}$ be the number of DWT levels which were used to compress that tile-component. If $D_{t,c} \geq r$, the server should discard all precincts belonging to the tile-component's *r* highest resolution levels; otherwise, it should discard all precincts belonging to the tile-component's $D_{t,c}$ highest resolution levels, leaving only those precincts which represent the tile-component's lowest LL sub-band.

For each precinct which remains after the discarding of tiles, components and resolution levels mentioned above, the server should identify whether or not the code-blocks which belong to that precinct contribute to the reconstruction of the region defined by E_1^{reg}, $E_2^{\mathrm{reg}}$and F_1^{reg}, F_2^{reg} on the codestream's high-resolution grid. A code-block contributes to this region if any of its samples affects the reconstruction of any full-resolution image component sample whose coordinates (x,y) satisfy:

$$E_1^{\mathrm{reg}} \le \mathrm{XRsiz}^c \cdot x < F_1^{\mathrm{reg}} \quad \text{and} \quad E_2^{\mathrm{reg}} \le \mathrm{YRsiz}^c \cdot y < F_2^{\mathrm{reg}}$$

where XRsiz^c and YRsiz^c denote the horizontal and vertical subsampling factors for the relevant component, c, in the codestream's SIZ marker segment.

It is important to bear in mind that the reconstruction of a full-resolution image component involves wavelet synthesis, which is an inherently expansive process. Thus, the region to which any given precinct contributes generally overlaps the regions to which its neighbouring precincts contribute. The server should be prepared to account for these expansive effects of the wavelet transform when determining the precincts which are relevant to a client's request.

Section 10.6.4 of the book "JPEG2000: image compression fundamentals, standards and practice" [11] describes one way to calculate the samples of any given sub-band which contribute to a given region on the codestream's high-resolution grid. From the sub-band regions, it is a simple matter to deduce the contributing code-blocks and hence precincts.

M.4.2 Decoding an image from returned JPP-stream 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 between applications.

M.5 JPIP protocol transcripts

M.5.1 Introduction

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 might indicate that some of the data transmitted is not shown.

M.5.2 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" might 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 second request uses the new path, "jpip.cgi", 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 second request indicates that the view-window has been changed and a smaller window centred in the requested view-window is being returned. The server starts returning the data for this window.

Before receiving the complete response to the second request, the client issues a third request. The client has adjusted its view-window to the size specified by server.

The server continues to respond to the second request for a while, then starts a response to the third request. During this response, the client issues a fourth request with a slightly different region. The server continues to respond to the third request for a while then starts responding to the fourth request.

The client waits until the fourth 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
<<
```

The following is an example of session-based HTTP GET with a model request.

```
<< GET /jpip.cgi?fsiz=1024,768&cid=JPH_5&model=Hm,H*,M*,P* HTTP/1.1
<< Host: jpip.jpeg.org
<< Cache-Control: no-cache

    >> HTTP/1.1 200 OK
    >> Cache-control: no-cache
    >> Transfer-Encoding: chunked
    >> 3
    -- 3 bytes of binary data
    >> 0
```

The following is an example of a stateless HTTP GET with a model request.

```
<< GET /images/kids.jp2?fsiz=1024,768&model=M0,Hm,H0:20,P0 HTTP/1.1
<< Host: jpip.jpeg.org
<< Cache-Control: no-cache

    >> HTTP/1.1 200 OK
    >> Cache-Control: no-cache
    >> Transfer-Encoding: chunked
    >> Content-Type: image/jpp-stream
    >> 400
    -- 1024 bytes of binary data
    >> 3f8
    -- 1016 bytes of binary data
    >> 0
```

M.5.3 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.cgi?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.cgi?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.cgi?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.cgi?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.cgi?cclose=JPHT033C38BE481154F9 HTTP/1.1
<< Host: dst-m
<< Cache-Control: no-cache
<<
```

M.6 Using JPIP with HTML

A JPIP system can be used with HTML and XHTML 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 fragment:

```
<img src="http://jpip.jpeg.org/name.jp2?fsiz=128,128&rsiz=128,128&type=image/jpeg"
width="128" height="128">
```

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&rsiz=128,128&type=image/jpeg
Host: jpip.jpeg.org
```

and will include many other HTTP header lines, typically identifying the browser, and the types of things the browser accepts. This HTTP request is a valid JPIP request and a JPIP server which receives this request shall either return an error message or determine the relevant 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 valid JPIP response, and is also a valid 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-encoding so that this request could be interrupted. The preceding example is not an example of chunked transfer-encoding.

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 fragment:

```
<img src="http://jpip.jpeg.org/name.jp2?rsiz=128,128" width="128" height="128">
```

In this case, there is no explicit type requested. A JPIP server using HTTP should therefore examine the "Accept:" line of the HTTP request issued by the client. 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 N

JPIP ABNF collection

(This annex does not form an integral part of this Recommendation | International Standard.)

N.1 JPIP Request ABNF

```
;=================================
; C.1.1 Request structure
;=================================

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                        ; C.2.2
                   / subtarget                     ; C.2.3
                   / tid                           ; C.2.4

channel-field      = cid                           ; C.3.2
                   / cnew                          ; C.3.3
                   / cclose                        ; C.3.4
                   / qid                           ; C.3.5

view-window-field  = fsiz                          ; C.4.2
                   / roff                          ; C.4.3
                   / rsiz                          ; C.4.4
                   / fvsiz                         ; C.4.5
                   / rvoff                         ; C.4.6
                   / rvsiz                         ; C.4.7
                   / comps                         ; C.4.8
                   / stream                        ; C.4.9
                   / context                       ; C.4.10
                   / srate                         ; C.4.11
                   / roi                           ; C.4.12
                   / layers                        ; C.4.13
                   / mctres                        ; C.4.14

metadata-field     = metareq                       ; C.5.2

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

server-control-field = align                       ; C.7.1
                   / wait                          ; C.7.2
                   / type                          ; C.7.3
                   / drate                         ; C.7.4
                   / sendto                        ; C.7.5
                   / abandon                       ; C.7.6
                   / barrier                       ; C.7.7
                   / twait                         ; C.7.8

cache-management-field = model                     ; C.8.1
                   / tpmodel                       ; C.8.3
                   / need                          ; C.8.4
                   / tpneed                        ; C.8.5
                   / mset                          ; C.8.6

upload-field       = upload                        ; C.9.1

client-cap-pref-field = cap                        ; C.10.1
                   / pref                          ; C.10.2
                   / csf                           ; C.10.3
                   / handled                       ; C.10.4
```

```
==================================
; C.2.2 Target(target)
;==================================

target = "target" "=" PATH

;==================================
; C.2.3 Sub-target (subtarget)
;==================================

subtarget = "subtarget" "=" byte-range

byte-range = UINT-RANGE

;==================================
; C.2.4 Target ID (tid)
;==================================

tid = "tid" "=" target-id

target-id = TOKEN

;==================================
; C.3.1 Channel ID (cid)
;==================================

cid = "cid" "=" channel-id

channel-id = TOKEN

;==================================
; C.3.2 New Channel (cnew)
;==================================

cnew = "cnew" "=" 1#transport-name

transport-name = TOKEN

;==================================
; C.3.3 Channel Close (cclose)
;==================================

cclose = "cclose" "=" ("*" / 1#channel-id)

;==================================
; C.3.4 Request ID (qid)
;==================================

qid = "qid" "=" UINT

;==================================
; C.4.2 Frame Size (fsiz)
;==================================

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

fx = UINT

fy = UINT

round-direction = "round-up" / "round-down" / "closest"

;==================================
; C.4.3 Offset (roff)
;==================================

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

ox = UINT

oy = UINT

;==================================
; C.4.4 Region Size (rsiz)
;==================================

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

sx = UINT

sy = UINT
```

```
;==================================
; C.4.5 Frame Size for Variable Dimension Data (fvsiz)
;==================================

fvsiz = "fvsiz" "=" 1#UINT ["," round-direction]

round-direction = "round-up" / "round-down" / "closest"

;==================================
; C.4.6 Offset for Variable Dimension Data (rvoff)
;==================================

rvoff = "rvoff" "=" 1#UINT

;==================================
; C.4.7 Region Size for Variable Dimension Data(rvsiz)
;==================================

rvsiz = "rvsiz" "=" 1#UINT

;==================================
; C.4.8 Components (comps)
;==================================

comps = "comps" "=" 1#UINT-RANGE

;==================================
; C.4.9 Codestream (stream)
;==================================

stream = "stream" "=" 1#sampled-range

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

sampling-factor = UINT

;==================================
; C.4.10 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 / "<" / ">" / "[" / "]" / "-" / ":" / "+" )

;==================================
; C.4.11 Sampling Rate (srate)
;==================================

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

streams-per-second = UFLOAT

;==================================
; C.4.12 ROI (roi)
;==================================

roi = "roi" "=" region-name

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

```
;==================================
; C.4.13 Layers (layers)
;==================================

layers = "layers" "=" UINT

;==================================
; C.4.14 Multi-component transformation (MCT) resolution value (mctres)
;==================================

mctres = "mctres" "=" UINT

;==================================
; 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 = "!!"

;==================================
; C.6.1 Maximum Response Length (len)
;==================================

len = "len" "=" UINT

;==================================
; C.6.2 Quality (quality)
;==================================

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

;==================================
; C.7.1 Alignment (align)
;==================================

align = "align" "=" ("yes" / "no")

;==================================
; C.7.2 Wait (wait)
;==================================

wait = "wait" "=" ("yes" / "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 *( ";" parameter )

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

parameter = attribute "=" value

attribute = TOKEN

value = TOKEN
```

```
;==================================
; C.7.4 Delivery Rate (drate)
;==================================

drate = "drate" "=" rate-factor

rate-factor = UFLOAT

;==================================
; C.7.5 Send To (sendto)
;==================================

sendto = "sendto" "=" host ":" port ";" mbw ";" bpc

host = token

port = UINT

bpc  = UINT

;==================================
; C.7.6 Abandon (abandon)
;==================================

abandon = "abandon" "=" 1#chunk-range

chunk-range = chunk-qid ":" chunk-seq-range

chunk-qid = UINT

chunk-seq-range = UINT-RANGE

;==================================
; C.7.7 Barrier (barrier)
;==================================

barrier = "barrier" "=" barrier-qid

barrier-qid = UINT

;==================================
; C.7.8 Timed wait (twait)
;==================================

twait = "twait" "=" max-wait-usecs

max-wait-usecs = UINT

;==================================
; C.8.1.1 Model (model)
;==================================

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

model-item = [codestream-qualifier ","] model-element

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

bin-descriptor = explicit-bin-descriptor    ; C.8.1.2
               / implicit-bin-descriptor    ; C.8.1.3

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

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

first-codestream-id = UINT

last-codestream-id = UINT

;==================================
; 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 = %x4c UINT                ; "L"

codestream-main-header-bin = %x48 %x6d      ; "Hm"

meta-bin = %x4d bin-uid                          ; "M"

tile-bin = %x54 bin-uid                          ; "T"

tile-header-bin = %x48 bin-uid              ; "H"

precinct-bin = %x50 bin-uid                 ; "P"

bin-uid = UINT / "*"

;=================================
; 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 = %x74    ; t -- tile
                    / %x63    ; c -- component
                    / %x72    ; r -- resolution level
                    / %x70    ; p -- position

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

first-index-pos = UINT

last-index-pos = UINT

data-uid = UINT / "*"

;=================================
; C.8.3 Tile-part Model involving JPT-streams (tpmodel)
;=================================

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

tpmodel-item = [codestream-qualifier "," ] tpmodel-element

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

;=================================
; C.8.4 Need for Stateless Requests (need)
;=================================

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

need-item = [codestream-qualifier "," ] bin-descriptor

;=================================
; C.8.5 Tile-part Need for Stateless Requests (tpneed)
;=================================

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

tpneed-item = [codestream-qualifier "," ] tp-descriptor
```

```
;==================================
; C.8.6 Model Set for Requests within a session (mset)
;==================================

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

;==================================
; C.9.1 Upload (upload)
;==================================

upload = "upload" "=" upload-type

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

;==================================
; 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

vendor-capability = "vc." vendor-code [":" vendor-value]

vendor-code = 1*(LOWER / DIGIT / "." / "-")

vendor-value = TOKEN

depth-capability = "depth:" UINT

config-capability = "config:" UINT

;==================================
; C.10.2.1 General
;==================================

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

related-pref-set = view-window-pref            ; C.10.2.2
                 / colour-meth-pref            ; C.10.2.3
                 / max-bandwidth               ; C.10.2.4
                 / bandwidth-slice             ; C.10.2.5
                 / placeholder-pref            ; C.10.2.6
                 / codestream-seq-pref         ; C.10.2.7
                 / conciseness-pref            ; C.10.2.8
                 / other

other = TOKEN

;==================================
; C.10.2.2 View-window handling preferences
;==================================

view-window-pref = "fullwindow" / "progressive"

;==================================
; 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

;==================================
; C.10.2.4 Max bandwidth
;==================================

max-bandwidth = "mbw:" mbw

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

```
;==================================
; C.10.2.5 Bandwidth slice
;==================================

bandwidth-slice = "slice:" slice

slice = NONZERO

;==================================
; C.10.2.6 Placeholder preference
;==================================

placeholder-pref = "meta:" placeholder-branch

placeholder-branch = "incr" / "equiv" / "orig"

;==================================
; C.10.2.7 Codestream sequencing
;==================================

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

codestream-seq-option = "sequential" / "reverse-sequential"
                      / "interleaved"

;==================================
; C.10.2.8 Conciseness preference
;==================================

conciseness-pref = "loose" / "concise"

;==================================
; C.10.3 Contrast 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

;==================================
; C.10.4 Handled (handled)
;==================================

handled = "handled"
```

N.2 JPIP Response BNF

```
;==================================
; D.1.1 Reply structure
;==================================

Status-Code = 3DIGIT

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

jpip-response-header =
                     / JPIP-tid                      ; D.2.2
                     / JPIP-cnew                     ; D.2.3
                     / JPIP-qid                      ; D.2.4
                     / JPIP-fsiz                     ; D.2.5
                     / JPIP-rsiz                     ; D.2.6
                     / JPIP-roff                     ; D.2.7
                     / JPIP-fvsiz                    ; D.2.8
                     / JPIP-rvsiz                    ; D.2.9
                     / JPIP-rvoff                    ; D.2.10
                     / JPIP-comps                    ; D.2.11
                     / JPIP-stream                   ; D.2.12
                     / JPIP-context                  ; D.2.13
                     / JPIP-roi                      ; D.2.14
                     / JPIP-layers                   ; D.2.15
                     / JPIP-srate                    ; D.2.16
                     / JPIP-metareq                  ; D.2.17
```

```
                    / JPIP-len                    ; D.2.18
                    / JPIP-quality                ; D.2.19
                    / JPIP-type                   ; D.2.20
                    / JPIP-mset                   ; D.2.21
                    / JPIP-cap                    ; D.2.22
                    / JPIP-pref                   ; D.2.23
                    / JPIP-align                  ; D.2.24
                    / JPIP-subtarget              ; D.2.25
                    / JPIP-handled                ; D.2.26

;==================================
; D.2.2 Target ID (JPIP-tid)
;==================================

JPIP-tid = "JPIP-tid" ":" LWSP target-id

;==================================
; D.2.3 New Channel (JPIP-cnew)
;==================================

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

transport-param = TOKEN

;==================================
; D.2.4 Request ID (JPIP-qid)
;==================================

JPIP-qid = "JPIP-qid" ":" LWSP UINT

;==================================
; D.2.5 Frame Size (JPIP-fsiz)
;==================================

JPIP-fsiz = "JPIP-fsiz" ":" LWSP fx "," fy

;==================================
; D.2.6 Region Size (JPIP-rsiz)
;==================================

JPIP-rsiz = "JPIP-rsiz" ":" LWSP sx "," sy

;==================================
; D.2.7 Offset (JPIP-roff)
;==================================

JPIP-roff = "JPIP-roff" ":" LWSP ox "," oy

;==================================
; D.2.8 Frame Size for Variable Dimension Data (JPIP-fvsiz)
;==================================

JPIP-fvsiz = "JPIP-fvsiz" ":" LWSP 1#UINT

;==================================
; D.2.9 Region Size for Variable Dimension Data(JPIP-rvsiz)
;==================================

JPIP-rvsiz = "JPIP-rvsiz" ":" LWSP 1#UINT

;==================================
; D.2.10 Offset for Variable Dimension Data (JPIP-rvoff)
;==================================

JPIP-rvoff = "JPIP-rvoff" ":" LWSP 1#UINT

;==================================
; D.2.11 Components (JPIP-comps)
;==================================

JPIP-comps = "JPIP-comps" ":" LWSP 1#UINT-RANGE

;==================================
; D.2.12 Codestream (JPIP-stream)
;==================================

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

```
prefixed-range = "<" ctxt-id ":" ctxt-elt ">" sampled-range

ctxt-id = UINT

ctxt-elt = UINT

;==================================
; D.2.13 Codestream Context (JPIP-context)
;==================================

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

;==================================
; D.2.14 ROI (JPIP-roi)
;==================================

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

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

;==================================
; D.2.15 Layers (JPIP-layers)
;==================================

JPIP-layers = "JPIP-layers" ":" LWSP UINT

;==================================
; D.2.16 Sampling Rate (JPIP-srate)
;==================================

JPIP-srate = "JPIP-srate" ":" LWSP UFLOAT

;==================================
; D.2.17 Metadata request (JPIP-metareq)
;==================================

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

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

;==================================
; D.2.18 Maximum Response Length (JPIP-length)
;==================================

JPIP-len = "JPIP-len" ":" LWSP UINT

;==================================
; D.2.19 Quality (JPIP-quality)
;==================================

JPIP-quality = "JPIP-quality" ":" LWSP (1*2DIGIT / "100" / "-1")

;==================================
; D.2.20 Image Return Type (JPIP-type)
;==================================

JPIP-type = "JPIP-type" ":" LWSP image-return-type

;==================================
; D.2.21 Model Set (JPIP-mset)
;==================================

JPIP-mset = "JPIP-mset" ":" LWSP 1#sampled-range

;==================================
; D.2.22 Needed Capability (JPIP-cap)
;==================================

JPIP-cap = "JPIP-cap" ":" LWSP 1#capability-code
```

```
;==================================
; D.2.23 Unavailable Preference (JPIP-pref)
;==================================

JPIP-pref = "JPIP-pref" ":" LWSP 1#related-pref-set

;==================================
; D.2.24 Alignment (JPIP-align)
;==================================

JPIP-align = "JPIP-align" ":" LWSP "yes" / "no"

;==================================
; D.2.25 Subtarget (JPIP-subtarget)
;==================================

JPIP-subtarget = "JPIP-subtarget" ":" LWSP byte-range / src-codestream-specs

;==================================
; D.2.26 Handled request (JPIP-handled)
;==================================

JPIP-handled = "JPIP-handled" ":" LWSP 1#handled-req

handled-req = (request-field | partially-handled-req)

partially-handled-req = request-field "=" handled-req-option

request-field = TOKEN

handled-req-option = TOKEN
```

Annex O

Media type specifications and registrations

(This annex does not form an integral part of this Recommendation | International Standard.)

O.1 General

Many Internet protocols are designed to carry arbitrary labelled content. The mechanism used to label such content is a media type, which is defined in IETF RFC 6838 and consists of a top-level type, a subtype, and in some instances, optional parameters.

The media type specifications of the following clauses have a matching registration in the IANA central registry, as specified in IETF RFC 6838.

O.2 JPP-stream

O.2.1 General

The `image/jpp-stream` media type refers to content that consists of a sequence of JPIP messages whose class identifiers are limited to those listed in Table A.2 as compatible with the JPP-stream type.

O.2.2 Registration

```
Type name: image
Subtype name: jpp-stream
Required parameters: None
Optional parameters: ptype=ext
Encoding considerations: binary
Security considerations: a jpp-stream contains structures of variable
length and an extensible syntax. Both of these aspects present potential
security risks for implementations. In particular, variable length
structures present buffer overflow risks and extensible syntax could
result in the triggering of adverse actions.
Interoperability considerations: jpp-stream media is self-describing and
may be used to describe one or more JPEG 2000 codestreams, or parts
thereof, along with arbitrary elements from JPEG 2000 family file formats
that may contain those codestreams. Readers need to be prepared to receive
and interpret fragments of precincts from an original JPEG 2000
codestream.
Published specification: Rec. ITU-T T.808 | ISO/IEC 15444-9
Applications: Multimedia and scientific
Fragment identifier considerations: None
Restrictions on usage: None
Provisional registration: Yes
Additional information: None
Magic number(s): None
File extension(s): jpp
Macintosh File Type Code(s): N/A
Object Identifiers: N/A
Contact name: ISO/IEC JTC 1/SC 29/WG 1 Convenor
Contact email address: sc29-sec@itscj.ipsj.or.jp
Intended usage: COMMON
Change controller: ITU-T & ISO/IEC JTC 1
```

O.3 JPT-stream

O.3.1 General

The `image/jpt-stream` media type refers to content that consists of sequence of JPIP messages whose class identifiers are limited to those listed in Table A.2 as compatible with the JPT-stream type.

O.3.2 Registration

```
Type name: image
Subtype name: jpt-stream
Required parameters: None
Optional parameters: ttype=ext
Encoding considerations: binary
Security considerations: a jpt-stream contains structures of variable
length and an extensible syntax. Both of these aspects present potential
security risks for implementations. In particular, variable length
structures present buffer overflow risks and extensible syntax could
result in the triggering of adverse actions.
Interoperability considerations: jpt-stream media is self-describing and
may be used to describe one or more JPEG 2000 codestreams, or parts
thereof, along with arbitrary elements from JPEG 2000 family file formats
that may contain those codestreams. Readers need to be prepared to receive
and interpret fragments of tiles from an original JPEG 2000 codestream.
Published specification: Rec. ITU-T T.808 | ISO/IEC 15444-9
Applications: Multimedia and scientific
Fragment identifier considerations: None
Restrictions on usage: None
Provisional registration: Yes
Additional information: None
Magic number(s): None
File extension(s): jpt
Macintosh File Type Code(s): N/A
Object Identifiers: N/A
Contact name: ISO/IEC JTC 1/SC 29/WG 1 Convenor
Contact email address: sc29-sec@itscj.ipsj.or.jp
Intended usage: COMMON
Change controller: ITU-T & ISO/IEC JTC 1
```

Bibliography

[1] Taubman, D., Remote Browsing of JPEG 2000 Images, *Proc. Int. Conf. on Image Processing*, Vol. 1, pp. 229-232, Sept. 2002.

[2] Li, J., Sun, H., Li, H., Zhang, Q., Lin, X., Vfile – A Virtual File Media Access Mechanism and its Application in JPEG2000 Images for Browsing over Internet, *ISO/IEC JTC 1/SC 29/WG 1 Document Register: N1473,* Nov. 1999.

[3] Boliek, M., Wu, G.K., Gormish, M.J., 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] Deshpande, S., Zeng, W., Scalable Streaming of JPEG2000 Images Using Hypertext Transfer Protocol, *Proc. ACM Conf. on Multimedia*, pp. 372-381, Oct. 2001.

[5] Wright, A., Clark, R., Colyer, G., An Implementation of JPIP Based on HTTP, *ISO/IEC JTC 1/SC 29/WG 1 Document Register: N2426*, Feb. 2002.

[6] Gormish, M., Banerjee, S., Tile-Based Transport of JPEG 2000, N. Garcia, J.M. Martinez, L. Salgado (Eds.), VLVB03, LNCS 2849, pp. 217-224, 2003.

[7] Taubman, D., Rosenbaum, R., Rate-Distortion Optimized Interactive Browsing of JPEG2000 Images, *Proc. Int. Conf. on Image Processing*, Sept. 2003.

[8] Taubman, D., Prandolini, (R.): Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000, presented at *Visual Communications and Image Processing*, Lugano, Switzerland, 2003.

[9] Gormish, M.J., Truew, Transport of Reversible and Unreversible EmbeddedWavelets (A JPIP Proposal), *ISO/IEC JTC 1/SC 29/WG 1 Document Register: N2602*, July 2002.

[10] Canon: Proposal for JPIP Tier 2 protocol, *ISO/IEC JTC 1/SC 29/WG 1 Document Register: N2608*, June 2002.

[11] Taubman, D., Marcellin, M., JPEG2000: image compression fundamentals, standards and practice, *Kluwer Academic Publishers*, Boston, 2001.

[12] Richter, Th., Brower, B., Martucci, S., and Tzannes, A. (2009), *Interoperability in JPIP and its Standardization in JPEG 2000,* Part 9, In: A. Tescher (Ed.): Applications of Digital Image Processing XXXII, Proc. SPIE 2009.

[13] Recommendation ITU-T T.804 | ISO/IEC 15444-5, *Information technology – JPEG 2000 image coding system: Reference software*.

Published in Switzerland
Geneva, 2023