



DPC: 25/30509801 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 09 October 2025

Origin: International

Latest date for receipt of comments: 01 December 2025

Project No. 2025/00409

Responsible committee: IST/37 Coding of picture, audio, multimedia and hypermedia information

Interested committees:

Title: Draft BS EN ISO/IEC 15444-9 Information technology - JPEG 2000 image coding system. Part 9: Interactivity tools, APIs and protocols.

Please notify admin.start@bsigroup.com if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 01 December 2025

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT International Standard

ISO/IEC DIS 15444-9

Information technology — JPEG 2000 image coding system —

Part 9: Interactivity tools, APIs and protocols

*Technologies de l'information — Système de codage d'images
JPEG 2000 —*

*Partie 9: Outils d'interactivité, interfaces de programmes
d'application et protocoles*

ICS: 35.040.30

ISO/IEC JTC 1/SC 29

Secretariat: JISC

Voting begins on:
2025-10-08

Voting terminates on:
2025-12-31

This document has not been edited by the ISO Central Secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/IEC DIS 15444-9:2025(en)

Full document details: 25/30509801 DC

© ISO/IEC 2025  **Documents**
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

CONTENTS

Page

1	Scope	1
2	Normative references	1
3	Definitions	2
	3.1 Definitions from other standards or recommendations.....	2
	3.2 JPIP definitions.....	3
	3.3 Symbols.....	4
4	Abbreviations	6
5	Conventions	6
	5.1 ABNF rules.....	6
	5.2 File format ABNF rules	7
	5.3 Key to graphical descriptions of boxes (informative)	7
6	General description	8
	6.1 JPIP protocol	8
	6.2 Purpose.....	9
7	Conformance.....	10
Annex A	The JPP-stream and JPT-stream media types	11
	A.1 Introduction	11
	A.2 Message header structure	12
	A.3 Data-bins	14
	A.4 Conventions for parsing and delivery of JPP-streams and JPT-streams.....	22
	A.5 Conventions for JPP-stream or JPT-stream interoperability (informative)	23
Annex B	Sessions, channels, cache model and model-sets	24
	B.1 Requests within a session vs. stateless requests	24
	B.2 Channels and sessions.....	24
	B.3 Cache model management	25
	B.4 Interrogation and manipulation of model-sets.....	25
Annex C	Client request.....	26
	C.1 Request syntax	26
	C.2 Target identification fields	27
	C.3 Fields for working with sessions and channels	29
	C.4 View-window request fields.....	30
	C.5 Metadata request fields	43
	C.6 Data limiting request fields	49
	C.7 Server control request fields.....	49
	C.8 Cache management request fields	53
	C.9 Upload request parameters	58
	C.10 Client capability and preference request fields.....	58
Annex D	Server response signalling	66
	D.1 Reply syntax	66
	D.2 JPIP response headers	68
	D.3 Response data	73
Annex E	Uploading images to the server	74

E.1 Introduction	74
E.2 Upload request	74
E.3 Server response	74
E.4 Merging data on the server	75
Annex F Using JPIP over HTTP/1.1	77
F.1 Introduction	77
F.2 Requests	77
F.3 Session establishment	78
F.4 Responses	78
F.5 Additional HTTP features	79
F.6 HTTP and length request field (informative)	80
Annex G Using JPIP with HTTP/1.1 requests and TCP returns	81
G.1 Introduction	81
G.2 Client requests	81
G.3 Session establishment	81
G.4 Server responses	82
G.5 TCP and length request field (informative)	82
Annex H Using JPIP with alternate transports	83
H.1 Introduction	83
H.2 Reliable requests with unreliable data	83
H.3 Unreliable requests with unreliable data	84
H.4 Request and response syntax	84
H.5 Session establishment	85
Annex I Indexing JPEG 2000 files for JPIP	86
I.1 Introduction (informative)	86
I.2 Identifying the use of JPIP index boxes in the JPEG 2000 file format compatibility list	87
I.3 Defined boxes	87
I.4 Association of codestream indexes with codestreams	95
I.5 Placement restrictions (informative)	95
Annex J Profiles and variants for interoperability and testing	96
J.1 Introduction	96
J.2 Definition of variants	96
J.3 Definition of profiles	97
J.4 Testing methodology	99
Annex K Using JPIP with HTTP/1.1 requests and UDP returns	104
K.1 Introduction	104
K.2 Client requests	104
K.3 Response data delivery and channel establishment	104
K.4 Server responses	105
K.5 Framing of response data into chunks	105
K.6 Client acknowledgement of server responses	106
K.7 UDP and Maximum Response Length Field (informative)	107
K.8 Implementation strategies for acknowledged communication (informative)	107
K.9 Implementation strategies for unacknowledged communication (informative)	108
Annex L Registration of extensions	109

Annex M Application examples.....	110
M.1 Introduction	110
M.2 Use of JPIP with codestreams in other file formats.....	110
M.3 Tile-part implementation techniques	110
M.4 Precinct-based implementation techniques	111
M.5 JPIP protocol transcripts	112
M.6 Using JPIP with HTML	115
Annex N JPIP ABNF collection	116
N.1 JPIP Request ABNF.....	116
N.2 JPIP Response BNF	123
Annex O Media type specifications and registrations.....	127
O.1 General	127
O.2 JPP-stream	127
O.3 JPT-stream.....	128
Annex P Using JPIP with HTTP/3	129
P.1 Introduction	129
P.2 Client requests	130
P.3 Server responses	130
P.4 Server pushed responses	130
P.5 Acquire barriers	131
P.6 Side effects and sequential consistency	131
P.7 JPIP channels, pre-emption and the wait request field	131
Bibliography	132

Figures

Page

Figure 1 – Example of the box description figures	8
Figure 2 – Example of the superbox description figures.....	8
Figure 3 – JPIP protocol overview.....	8
Figure 4 – JPIP protocol stack	9
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).....	11
Figure A.2 – VBAS structure.....	12
Figure A.3 – Bin-ID VBAS structure.....	12
Figure A.3bis – Precinct sequence number	14
Figure A.4 – Example precinct data-bin.....	15
Figure A.5 – Metadata-bin example colour scheme	17
Figure A.6 – A sample JP2 file.....	17
Figure A.7 – A sample JP2 file divided into three metadata-bins	18
Figure A.8 – A superbox with a referenced metadata-bin.....	19
Figure A.9 – An invalid division of the file into metadata-bins	19
Figure A.10 – Example of the use of stream equivalents	20

Figure A.11 – Placeholder box structure	20
Figure C.1 –Desired region within an image	31
Figure C.2 – Desired region with respect to the subsampled reference grid	31
Figure C.3 – Colourspace specification box selection procedure	62
Figure G.1 – Response data structure on http-tcp connection	82
Figure I.1 – Part of an example JPEG 2000 file containing JPIP index boxes.....	87
Figure I.2 – Organization of the contents of a Codestream Index box	88
Figure I.3 – Organization of the contents of a Codestream Finder box	89
Figure I.4 – Organization of the contents of a Manifest box	89
Figure I.5 – Organization of the contents of a Fragment Array Index box	90
Figure I.6 – Organization of the contents of a Header Index Table box	91
Figure I.7 – Organization of the contents of a Tile-part Index Table box.....	92
Figure I.8 – Organization of the contents of a Tile Header Index Table box	92
Figure I.9 – Organization of the contents of a Precinct Packet Index Table box	92
Figure I.10 – Organization of the contents of a Packet Header Index Table box.....	93
Figure I.11 – Organization of the contents of a File Index box	93
Figure I.12 – Organization of the contents of a File Finder box.....	94
Figure I.13 – Organization of the contents of a Proxy box	94
Figure I.14 – Organization of the contents of an Index Finder box.....	95
Figure K.1 – Response data structure on http-udp connection	106

Tables

	<i>Page</i>
Table A.1 – Bin-ID additional VBAS indication.....	13
Table A.2 – Class identifiers for different data-bin message classes.....	13
Table A.3 – Valid values for the Flags field of a Placeholder box	21
Table C.1 – Round direction options.....	34
Table C.2 – Metadata request qualifier flags.....	48
Table C.3 – Alignment boundaries based on bin type	50
Table C.4 – Valid image return types.....	50
Table C.5 – Cache descriptor option summary.....	56
Table C.6 – Valid capabilities of the processing-capabilities element.....	59
Table C.7 – Valid values of the config-capability parameter.....	59
Table C.8 – View-window handling preferences.....	61
Table C.9 – Colourspace method client preferences	61
Table C.10 – Placeholder preferences	63
Table C.11 – Codestream sequencing preferences.....	64
Table C.12 – Conciseness preferences	64
Table D.1 – Valid values of transport-param.....	68
Table D.2 – Defined reason codes	73
Table D.3 – Additional handled-req-option values for particular request fields	73
Table I.1 – Defined boxes (Informative).....	88
Table I.2 – Container type values.....	89
Table I.3 – Version values	91
Table J.1 – Defining requirements of variants.....	97
Table J.2 – Set of fields included in each profile.....	98
Table K.1 – Interpretation of the "control" field in each data chunk header	106
Table M.1 – Example of the use of auxiliary fields in a simple case.....	111
Table M.2 – Example of the use of auxiliary fields in a more complicated case	111

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 2, are:

1. specifies methods for HTTP/3 transported JPIP communication (Annex P); and
2. adds the H3 and H3S transport types to the definition of the New Channel request and response fields.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION**Information technology — JPEG 2000 image coding system — Part 9: Interactivity tools, APIs and protocols****1 Scope**

This Recommendation | International Standard¹ 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>.
- IETF RFC 9000, *QUIC: A UDP-Based Multiplexed and Secure Transport*. Available from World Wide Web: <<https://datatracker.ietf.org/doc/rfc9000>>.
- IETF RFC 9001, *Using TLS to Secure QUIC*. Available from World Wide Web: <<https://datatracker.ietf.org/doc/rfc9001>>.
- IETF RFC 9114, *HTTP/3*. Available from World Wide Web: <<https://datatracker.ietf.org/doc/rfc9114>>.

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 Definitions from other standards or recommendations

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.
- 3.1.9 connection:** Transport layer virtual circuit established between two programs for the purpose of communication.
Source: RFC 2616
- 3.1.10 entity:** Information transferred as the payload of a request or response.
Source: RFC 2616
Note on entry: An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body.
- 3.1.11 proxy:** Intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.
Source: RFC 2616
Note on entry: Requests are serviced internally or by passing them on, with possible translation, to other servers.

3.1.12 HTTP/1.1: Hypertext Transfer Protocol specified in RFC 2616.

3.1.13 HTTP/3: Hypertext Transfer Protocol specified in RFC 9114.

3.2 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.

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 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.

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

3.2.6 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).

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

3.2.8 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.

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

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

3.2.11 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.

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

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

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

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

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

3.2.17 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.

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

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

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

3.2.21 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.

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

3.2.23 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.

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

3.2.25 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.

3.2.26 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.

3.3 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
XO_{siz}	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)
Xs_{iz}	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

Y_{Osiz}	The vertical offset from the origin of the reference grid of the relevant codestream's SIZ marker segment
Y_{R_{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)
Y_{siz}	The height of the reference grid of the relevant codestream's SIZ marker segment
Y_{S_{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.

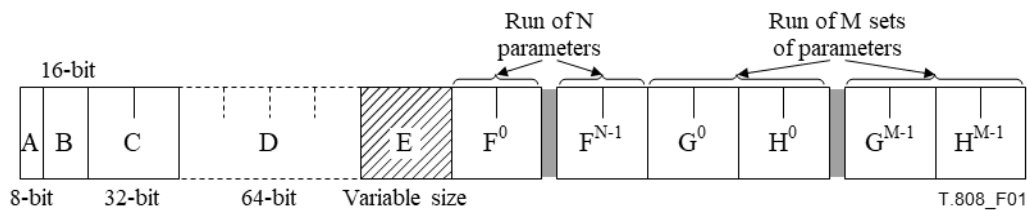


Figure 1 – Example of the box description figures

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.



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), UDP (RFC 768) and HTTP/3 (RFC 9114) 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.

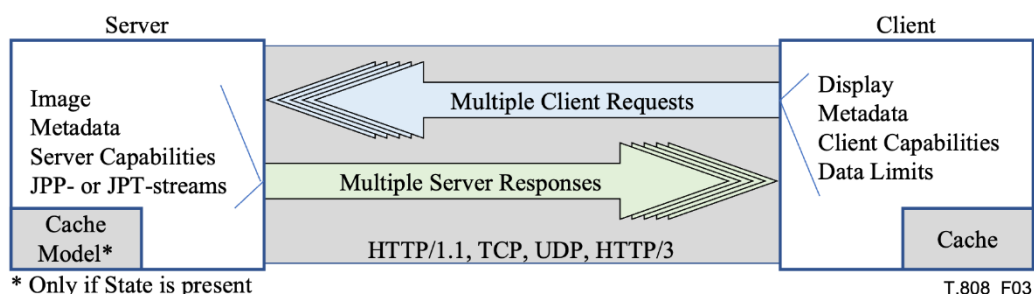


Figure 3 – JPIP protocol overview

This protocol can be used over several different transports as shown in Figure 4. This Recommendation | International Standard includes normative annexes on the use of the JPIP protocol over HTTP/1.1, TCP, UDP and HTTP/3, 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 six specific transports, which are identified by the strings "http", "https", "http-tcp", "http-udp", "h3" and "h3s" in the value string associated with New Channel requests.

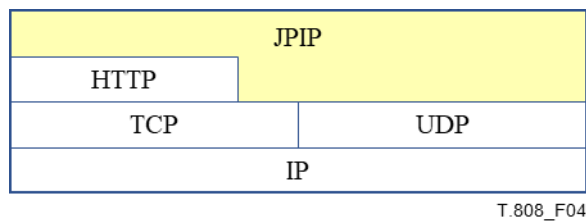


Figure 4 – JPIP protocol stack

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.

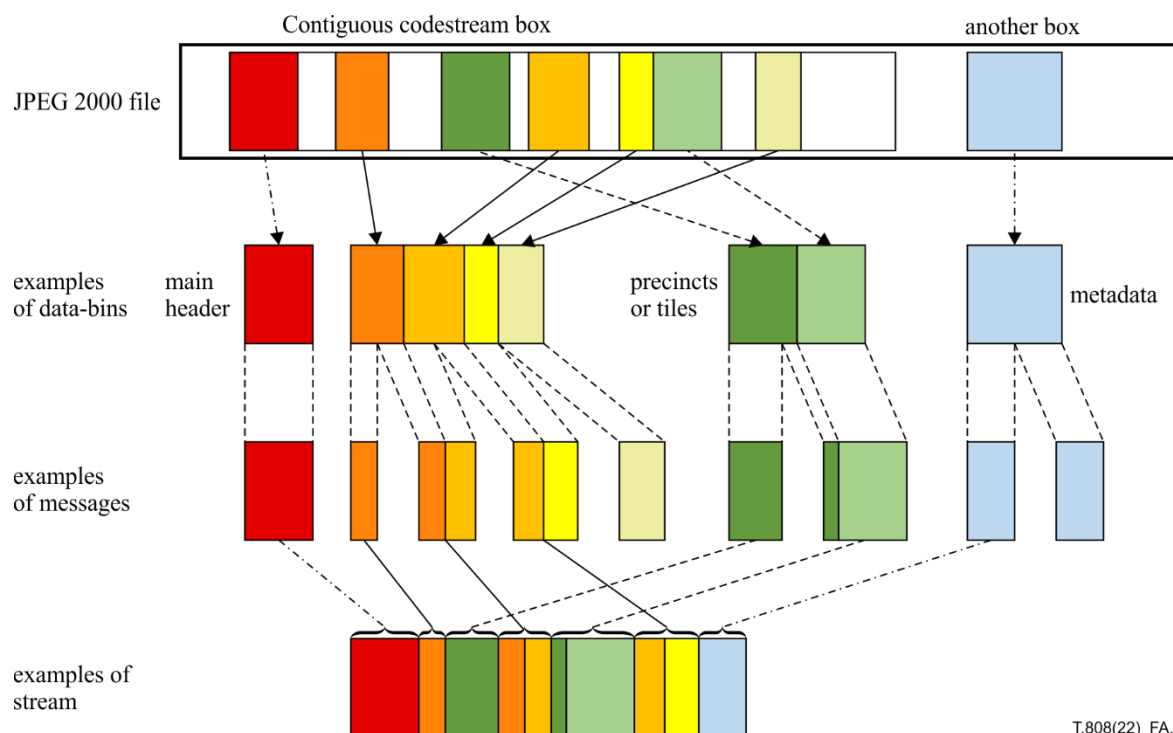
Annex A

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.



T.808(22)_FA.1

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.

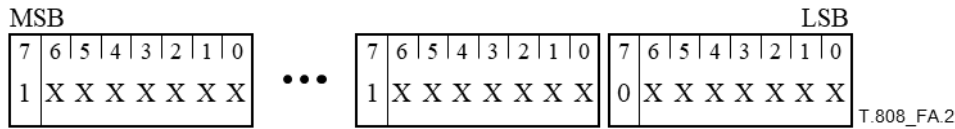


Figure A.2 – VBAS structure

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 their 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.

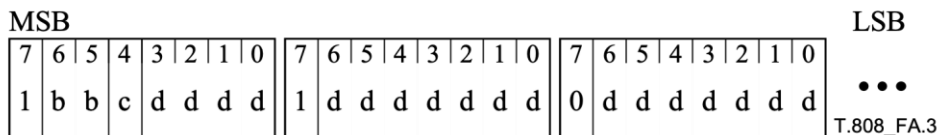


Figure A.3 – Bin-ID VBAS structure

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.

$$I = t + (c + s \times \text{num_components}) \times \text{num_tiles} \quad (\text{A-1})$$

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. Figure A.3bis provides an illustrative example of the assignment of sequence numbers s within a given tile-component. In the figure, only the two lowest resolutions of a tile-component are shown, with square precincts having the same dimensions in each resolution, so that higher resolutions have more precincts than lower resolutions.

NOTE – In general, the number of precincts in each resolution of a tile-component depends on the dimensions and location of the tile on the canvas, as determined from the codestream's SIZ marker segment, together with the component sampling factors, also determined from the SIZ marker segment, in addition to the number of decomposition levels and the precinct dimensions for each resolution, as found in the COD or COC marker segment that is associated with the tile-component.

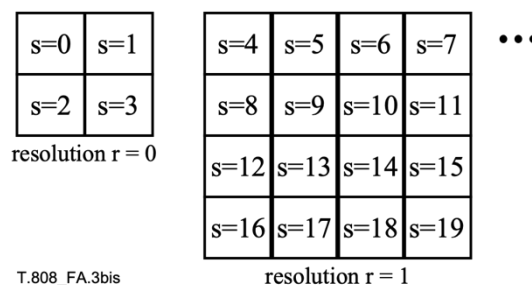


Figure A.3bis – Precinct sequence number

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).

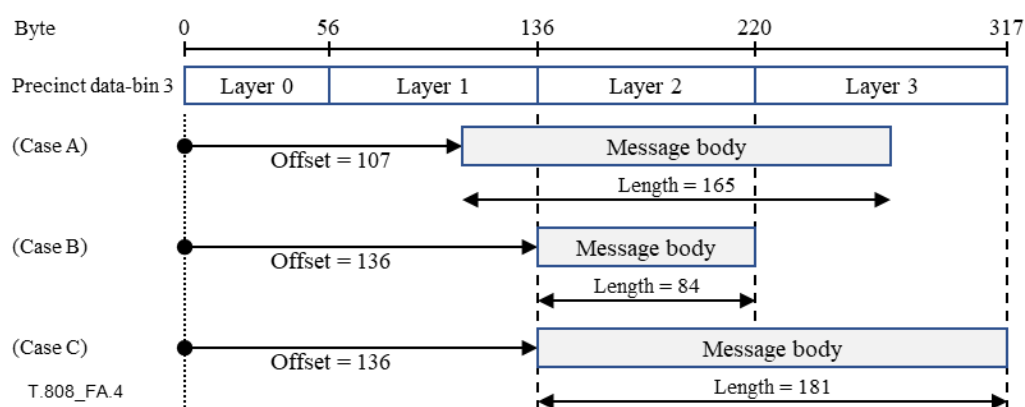


Figure A.4 – Example precinct data-bin

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 may choose any numeric ID for any particular metadata bin, with the only exception that the metadata-bin containing the root of the logical target shall be assigned ID 0. Nevertheless, servers should 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 which case this metadata-bin 0 will be empty. A client should not assume that no metadata is available if it has not yet received any metadata bin. Moreover, servers should send metadata-bin 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):

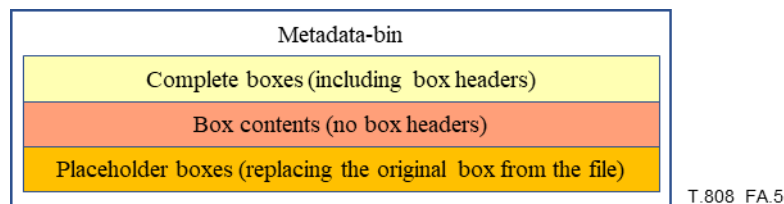


Figure A.5 – Metadata-bin example colour scheme

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

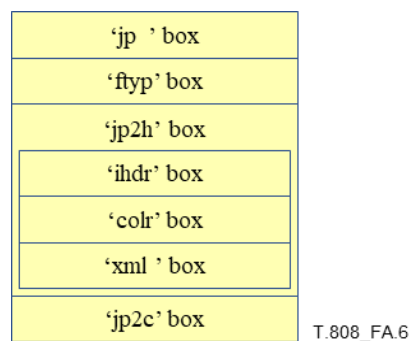


Figure A.6 – A sample JP2 file

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.

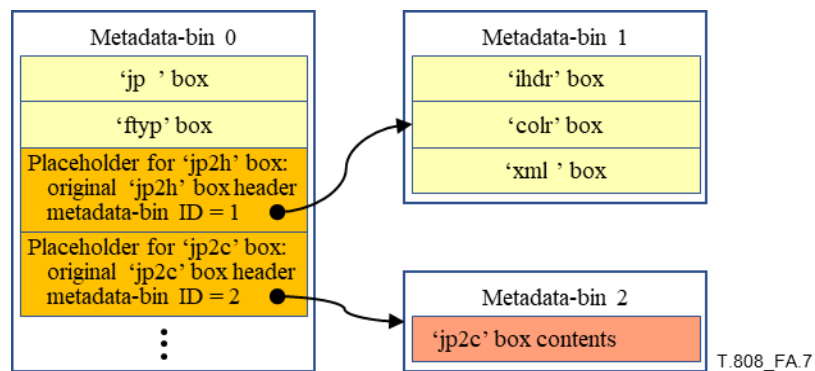


Figure A.7 – A sample JP2 file divided into three metadata-bins

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.

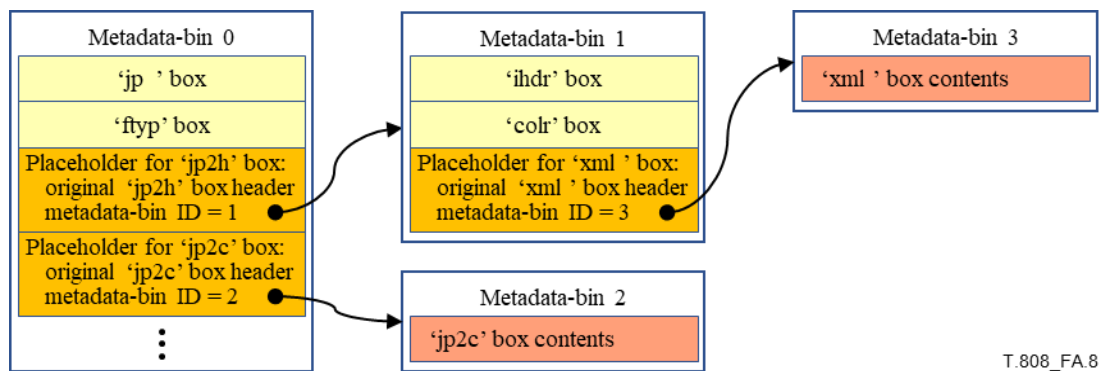
In general, placeholders that reference codestream data can do so either by referencing a separate metadata-bin (raw codestream), or by providing a codestream identifier (incremental codestream), or both. Even if both methods are provided, the JPP-stream or JPT-stream data available at a client or image-rendering 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.

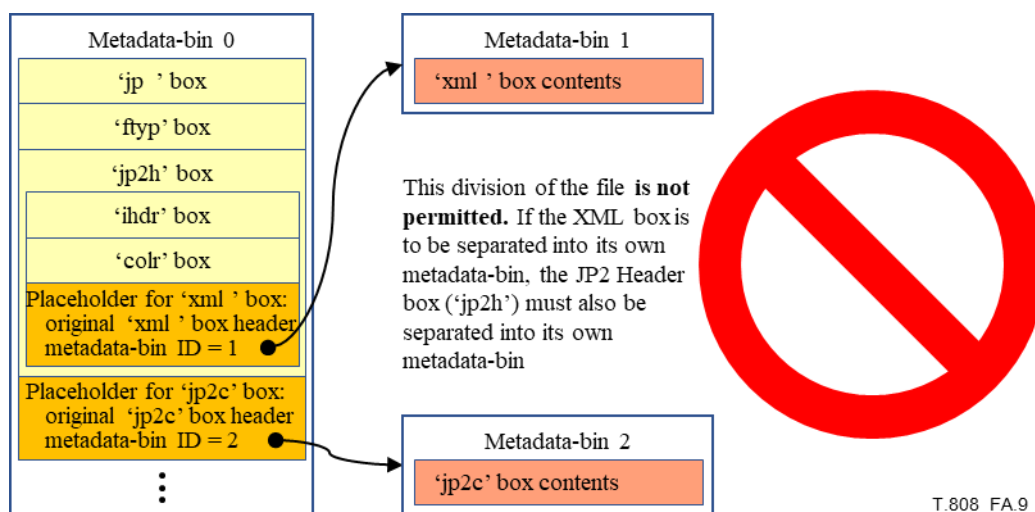
While considerable freedom exists in deciding how best to divide a file into metadata-bins, there is one restriction. Any Placeholder box that appears within a metadata-bin shall replace a top-level box within that data-bin. Equivalently, wherever a sub-box is to be replaced with a placeholder, its immediate 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 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.



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:



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.

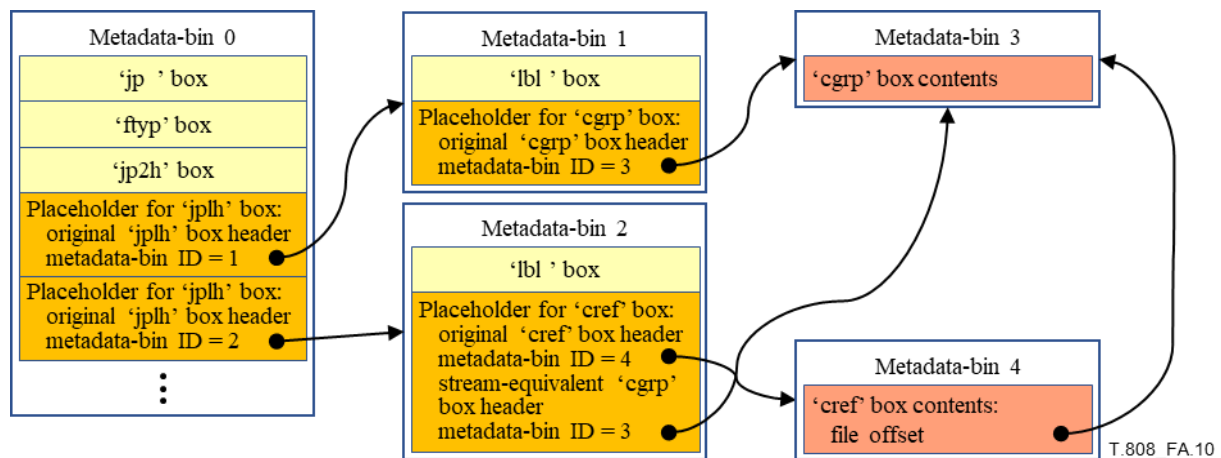


Figure A.10 – Example of the use of stream equivalents

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.

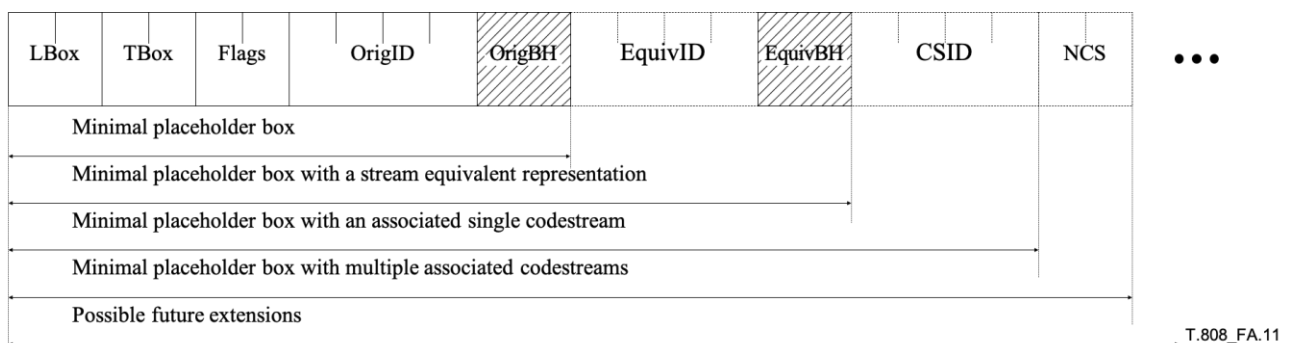


Figure A.11 – Placeholder box structure

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 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 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 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 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 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 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 $2N$ 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. The server can 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. Such a log is subsequently referred to as the cache model. Unless explicitly instructed otherwise, the server may assume that the client caches all data it receives within a session, so it can reliably 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. The server should maintain a cache model that persists for the duration of a session.

Stateless requests are not associated with any session and so shall be entirely self-contained. The term "stateless" here applies only to the server, not the client. A client can cache the responses from previous requests associated with the same logical target, regardless of whether those responses belong to a session or not. Clients that issue multiple stateless requests for the same target can include information about their cache contents with each request, so as to avoid the transmission of redundant data.

NOTE –The main benefits of using sessions is that they allow smaller, less complex requests and/or less redundant response data from the server, but this benefit relies upon the server maintaining a cache model. A benefit of stateless communication is that there is no expectation that the server maintains a cache model or any other state information between requests, so that requests for a single target image that emanate from a single client can each be handled by different servers.

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/1.1 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/1.1, 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/1.1, 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/1.1 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/1.1 context, becoming "metareq=%5broid/w%5d" so as to avoid the use of non-URI character – see 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", "http-udp", "h3" and "h3s". 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; and Annex P describes the use of JPIP over the "h3" and "h3s" transports.

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.

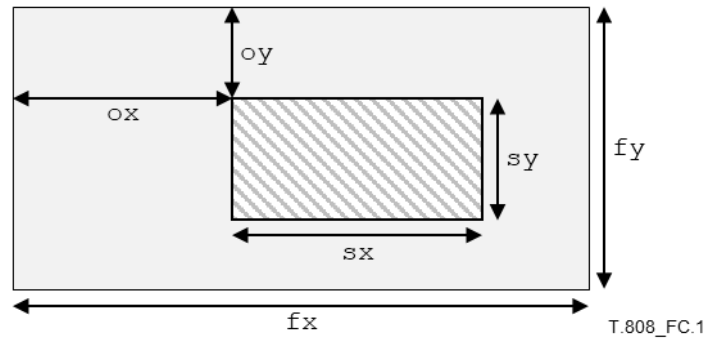


Figure C.1 – Desired region within an image

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 " $f_{siz}=640,480 \&rsiz=640,480 \&roff=0,0$ ". 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.

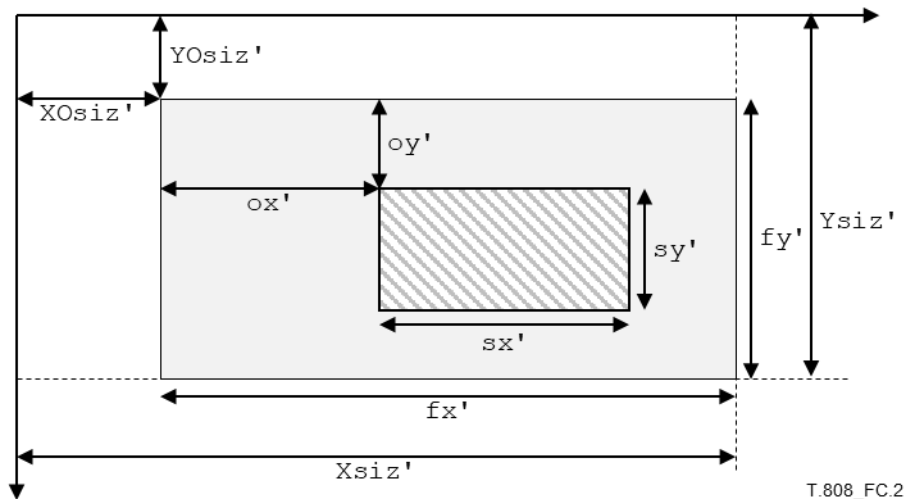


Figure C.2 – Desired region with respect to the subsampled reference grid

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.

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

where:

r is determined by the server in order to match the requested image size (f_x and f_y) as closely as possible, subject to any rounding preferences supplied via the Frame Size request field.

Here, XO_{siz} , YO_{siz} , Xs_{iz} and Ys_{iz} 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, f_x' and f_y' , have been found, the region size, s_x' and s_y' , and offset, ox' and oy' , associated with the codestream image region are determined by Equation C-2.

$$ox' = \left\lfloor ox \cdot \frac{f_x'}{f_x} \right\rfloor; \quad oy' = \left\lfloor oy \cdot \frac{f_y'}{f_y} \right\rfloor; \quad sx' = \left\lfloor (sx + ox) \cdot \frac{f_x'}{f_x} \right\rfloor - ox'; \quad sy' = \left\lfloor (sy + oy) \cdot \frac{f_y'}{f_y} \right\rfloor - oy' \quad (C-2)$$

EXAMPLE 2: Suppose the requested Frame Size is 128×128 , and the image on the codestream's high resolution reference grid is described by $XO_{siz}=127$, $Xs_{iz}=648$, $YO_{siz}=0$ and $Ys_{iz}=504$. Suppose also that 3 levels of wavelet transform exist for all image components in the codestream. The available codestream image sizes are then:

$$\begin{aligned} 521 \times 504 & \quad \left(\left\lfloor \frac{648}{1} \right\rfloor - \left\lfloor \frac{127}{1} \right\rfloor \text{ by } \left\lfloor \frac{504}{1} \right\rfloor - 0 \right) \\ 260 \times 252 & \quad \left(\left\lfloor \frac{648}{2} \right\rfloor - \left\lfloor \frac{127}{2} \right\rfloor \text{ by } \left\lfloor \frac{504}{2} \right\rfloor - 0 \right) \\ 130 \times 126 & \quad \left(\left\lfloor \frac{648}{4} \right\rfloor - \left\lfloor \frac{127}{4} \right\rfloor \text{ by } \left\lfloor \frac{504}{4} \right\rfloor - 0 \right) \\ 65 \times 63 & \quad \left(\left\lfloor \frac{648}{8} \right\rfloor - \left\lfloor \frac{127}{8} \right\rfloor \text{ by } \left\lfloor \frac{504}{8} \right\rfloor - 0 \right) \end{aligned}$$

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 XR_{siz} and YR_{siz} , 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 $Xs_{iz}=1024$, $Ys_{iz}=1024$, $XO_{siz}=0$, $YO_{siz}=0$, and $XR_{siz}^0=1$, $YR_{siz}^0=1$, $XR_{siz}^1=2$, $YR_{siz}^1=2$, $XR_{siz}^2=2$, and $YR_{siz}^2=2$. 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 f_x , f_y and f_z , the offset by ox , oy and oz and the region size by s_x , s_y and s_z . In that case, Equation C-1 extends to:

$$ZO_{siz}' = \left\lfloor \frac{ZO_{siz}}{2^r} \right\rfloor \quad Zs_{iz}' = \left\lfloor \frac{Zs_{iz}}{2^r} \right\rfloor$$

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

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

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 Omccⁱ 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 with 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.