INTERNATIONAL STANDARD

ISO/IEC 15444-9

First edition
2005-12-01

AMENDMENT 4
2010-12-15

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

AMENDMENT 4: JPIP server and client profiles

Technologies de l'information — Système de codage d'images JPEG 2000: Outils d'interactivité, interfaces de programmes d'application et protocoles

AMENDEMENT 4: Profils du serveur JPIP et du client

Reference number
ISO/IEC 15444-9:2005/Amd.4:2010(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published by ISO in 2011

Published in Switzerland

CONTENTS

Foreword

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

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

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

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

ISO/IEC 15444-9:2005/Amd.4 was prepared jointly by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information* in collaboration with ITU-T. The identical text is published as Rec. ITU-T.808(2005)/Amd.4 (05/2010).

INTERNATIONAL STANDARD
RECOMMENDATION ITU-T

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

Amendment 4

JPIP server and client profiles

1) Clause 2, Normative references

Add the following reference to clause 2:

– Recommendation ITU-T T.803 (2002) | ISO/IEC 15444-4:2004, *Information technology – JPEG 2000 image coding system: Conformance testing.*

2) Clause 5.1

a) Replace the definition of `TOKEN` *by:*

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

b) Add the following token at the end of the ABNF-rules:

```
IDTOKEN    =  1*(TOKEN / ";")
```

3) Clause C.2.4

Replace the definition of the `target-id` *by the following:*

```
target-id = IDTOKEN
```

4) Clause C.3.2

Replace the definition of the `channel-id` *by the following:*

```
channel-id = IDTOKEN
```

5) Clause L.1

Add to the list of fields in L.1 the following:

```
;================================
; C.4.5 Frame Size for Variable Dimension Data (fvsiz)
;================================
fvsiz = "fvsiz" "=" 1#UINT ["," round-direction]
round-direction = "round-up" / "round-down" / "closest"
;================================
; C.4.7 Offset for Variable Dimension Data (rvoff)
;================================
rvoff = "rvoff" "=" 1#UINT
;================================
```

```
; C.4.8 Region Size for Variable Dimension Data(rvsiz)
;================================
rvsiz = "rvsiz" "=" 1#UINT
```

6) Clause L.2

a) *Replace the list of* `jpip-response` `headers` *by the following:*

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

b) *Add respectively the following items to the list of JPIP Response BNF:*

```
;================================
; D.2.8 Frame Size for Variable Dimension Data (JPIP-fvsiz)
;================================
JPIP-fvsiz = "JPIP-fvsiz" ":" LWSP 1#UINT
;================================
; D.2.9 Region Size for Variable Dimension Data(JPIP-rvsiz)
;================================
```

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

;================================

; D.2.10 Offset for Variable Dimension Data (JPIP-rvoff)

;================================

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

;================================

; D.2.24 Alignment (JPIP-align)

;================================

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

;================================

; D.2.25 Subtarget (JPIP-subtarget)

;================================

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

7) Clause 6.2

a) *Replace the third point of the second paragraph by the following:*

– 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.

b) *Add the following to 6.2:*

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

8) Clause 7

Change clause 7 to:

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 may reduce efficiency by sending additional data not explicitly requested for or redundant data depending on the network quality-of-service. Such implementation decisions are application specific and provide the JPIP system with high utility.

9) Clause 3.3

Add the following definitions:

3.3.24 **profile**: Conformance is structured according to profiles; a profile defines the 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 expecting the

server to support them. A server is conforming to a profile if it supports and implements all request fields within this profile to the extent defined in Annex J.

3.3.25 variant: Variants define the operating modes and features of the JPIP standard that a client and a server use to exchange requests and data. Clients and servers must provide a common subset of variants in order to interoperate.

10) Clause 5.1

In 5.1, delete the definition of ENCODED-CHAR, *and insert the following definition instead, and replace all occurrences of* ENCODED-CHAR *by* OCTAL-ENCODED-CHAR*:*

```
OCTAL-ENCODED-CHAR = "\" QUADDIG OCTDIG OCTDIG
QUADDIG = "0" / "1" / "2" / "3"
OCTDIG = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7"
```

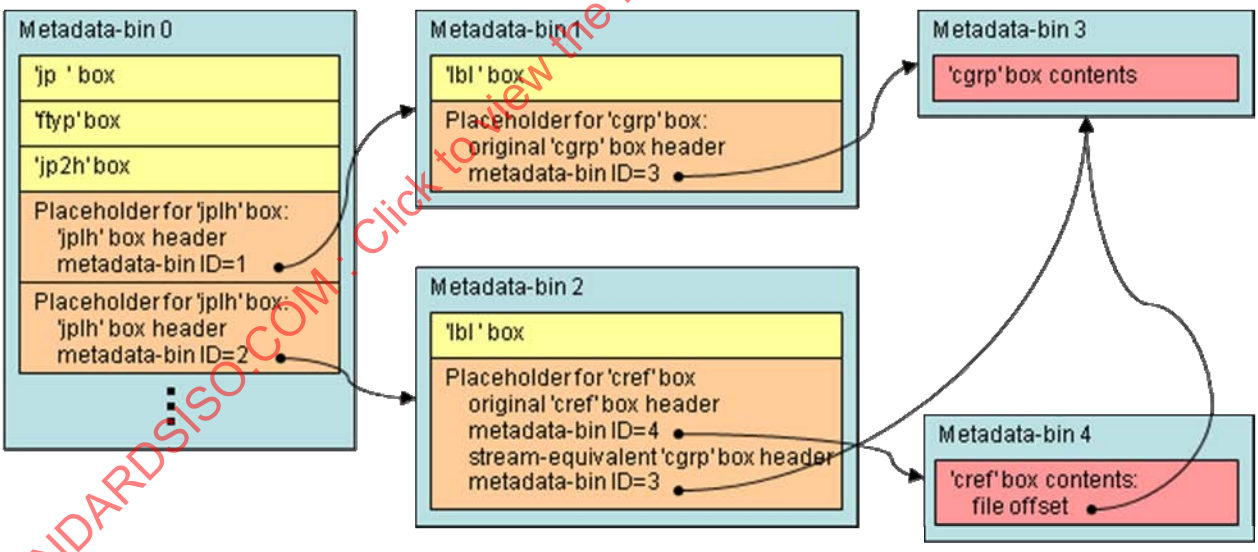
11) Clause 5.2

Replace all occurrences of ENCODED-CHAR *by* OCTAL-ENCODED-CHAR *and replace the first paragraph of 5.2 by the following:*

`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.

12) Clause A.3.6.2

At the end of A.3.6.2, replace the example Figure A.10 by the following:



13) Clause A.3.6.3

a) After the last paragraph of A.3.6.3, add a Note:

NOTE – The above definition implies that the placeholder box may be truncated after the last used field, but intermediate fields, even if unused, must be present.

b) In Table A.3, row five – the description of the flag value 01xx *– change the last sentence of the description in the right column to:*

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.

c) *Add the word "minimal" to the beginning of the description of each cut-off point in Figure A.11.*

14) Clause C.1.1

Replace the last sentence of the paragraph by:

Finally, even with a conforming request, a server might not implement all possible request fields or combinations thereof, but it must 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. Clause D.1 provides examples for servers using HTTP as transport protocol.

15) Clause C.1.2

a) *Remove the Note before the definition of the* `jpip-request-field` *and add the following:*

The fields in the request shall be sent in compliance with the selected transport protocol. For example, in HTTP, the requests may 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 may need to be escaped in order to avoid ambiguity. For example, a request field of the form `"target=me&my dog"` should be escaped in an HTTP context, becoming `"target=me%26my%20dog"`, so as to avoid confusion with the "&" used to separate request fields. As another example, `"metareq=[roid/w]"` should be escaped in an HTTP context, becoming `"metareq=%5broid/w%5d"` so as to avoid the use of non-URI character – see IETF RFC 2396 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.

b) *Replace the* `view-window field` *list by the following:*

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

16) Clause C.2.1

Replace the eighth paragraph (before the examples) of C.2.1 by the following:

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

17) Clause C.2.3

Replace the body of C.2.3 by:

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

byte-range = UINT-RANGE

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

This field may 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 codestreams in this target shall be assigned consecutive indices starting from zero.

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 byte-range are inclusive, where bytes or codestreams are counted from zero.

18) Clause C.3.5

Replace the body of C.3.5 by the following:

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.

19) Clause C.4.2

Replace the right-side entry of Table C.1 defining the meaning of `ROUND-DOWN` *by the following:*

For each requested codestream, the largest codestream image resolution whose width and height are both less than or equal to the specified size shall be selected. If there is none, then the smallest available codestream image resolution shall be used. This is the default value when the `round-direction` parameter is not specified.

20) New clause C.5.2.10

At the end of C.5.2, add C.5.2.10:

C.5.2.10 Special considerations for cross-reference boxes

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

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

21) Clause C.6.1

Replace the body of C.6.1 by the following:

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

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

If the `len` field is not present, the server should send image data to the client until such point as all of the relevant data has been sent, a quality limit is reached (see C.6.2), or the response is interrupted by the arrival of a new request that does not include a `wait` request field with a value of "yes" (see C.7.2). The client should use `len=0` if it requires response headers only and no entity body (see Annex F). Nevertheless, transport protocols that require framing of responses require an EOR message (see Annex G).

22) Clause C.7.1

Replace the first paragraph of C.7.1 by the following:

This field specifies whether the server response data shall be aligned on "natural boundaries". The default value is "no". If the server supports aligned responses and the value is "yes", any JPT-stream or JPP-stream message delivered in response to this request which crosses any natural boundary shall terminate at any subsequent natural boundary. Servers that do not support data alignment but receive an alignment request with the value "yes" shall indicate this by the Alignment Response defined in D.2.24.

The natural boundaries for each data-bin type are listed in Table C.3. A message is said to cross a natural boundary if it includes the last byte prior to the boundary and the first byte after the boundary.

NOTE – For example, a precinct data-bin crosses a natural boundary if it includes the last byte of one packet and the first byte of the next packet. Note carefully that aligned response messages are not actually required to terminate at a natural boundary unless they cross a boundary. This means, for example, that the response may include partial packets from precincts, which may be necessary if a prevailing byte limit prevents the delivery of complete packets.

23) Clause C.10.2.1

a) Replace the first paragraph (the syntax description) of the client preferences by the following:

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

related-pref-set = view-window-pref       ; C.10.2.2
                 / colour-meth-pref       ; C.10.2.3
                 / max-bandwidth          ; C.10.2.4
                 / bandwidth-slice        ; C.10.2.5
                 / placeholder-pref       ; C.10.2.6
                 / codestream-seq-pref    ; C.10.2.7
                 / conciseness-pref       ; C.10.2.8
                 / other

other = TOKEN
```

b) Replace the fourth paragraph of the client preferences by the following:

Unless otherwise stated, each `related-pref-set` specifies an ordered list of individual preference tokens, from most preferred to least preferred. Where possible, the server shall respect the client preferences identified by this request field. If a `related-pref-set` is followed by the `"/r"` modifier (required), the server shall either support one of the preferences listed in the `related-pref-set`, or else it shall respond with an error. In the latter case, the server shall return an *Unavailable preference* response header which identifies any `related-pref-set` which had the `"/r"` modifier but could not be supported. See D.2.23 for more on the *Unavailable preference* response header. Supporting a preference means that the server provides functionality which affects its behaviour in accordance with the preference. This addresses the server's functionality rather than the specific parameters established by other aspects of the request.

24) Clause C.10.2.2

Replace C.10.2.2 by the following:

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

This Recommendation | International Standard defines two options to specify the behaviour of the server in the event that the request cannot be serviced exactly as stated. These two options are specified in Table C.8.

Table C.8 – View-window handling preferences

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

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

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

25) Clause C.10.2.3

a) *Replace the third paragraph of the body of the clause by the following:*

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

b) *Replace the fifth paragraph of the body of the clause by the following:*

The optional `meth-limit` value specifies a limit on the APPROX value for that particular colourspace method. When using these preferences to select a colourspace specification, the server shall consider a colourspace method specification with an APPROX value of `meth-limit` or less as if the actual APPROX value was 1 (exact). This allows clients to specify the point at which colour fidelity is not important for a particular colourspace method, for the current application. For example, a page-layout application that is only concerned with aligning the image data with other elements on the page may not care at all about colour fidelity and set `meth-limit` to 4, meaning that the accuracy of the colourspace methods is unimportant. Another application that displays images on a low-quality screen may set `meth-limit` to 3, to indicate that as long as the colour accuracy is reasonable, it would be satisfied. The characters of the field shall be interpreted as an unsigned decimal integer. Allowed values are defined by the definition of the APPROX field in Table M.24 of Rec. ITU-T T.801 | ISO/IEC 15444-2, and by extensions and amendments to that Recommendation | International Standard. If the optional `meth-limit` value is not supplied, the default value shall be the largest value defined in that Recommendation | International Standard.

c) *Replace the flow chart of Figure C.3 by the following:*

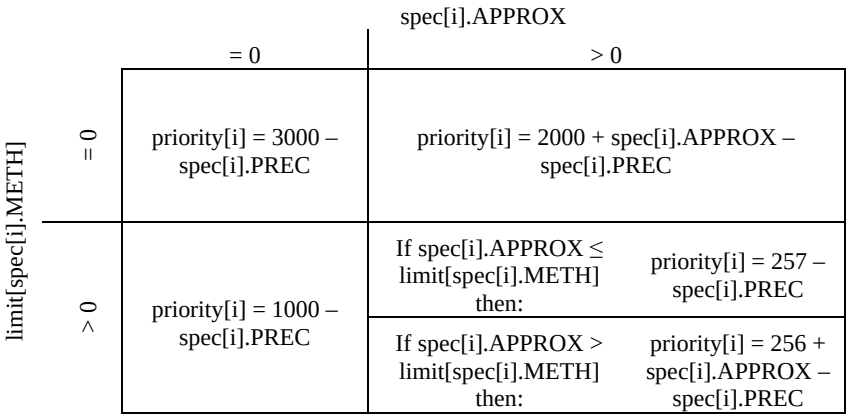


Figure C.3 – Colourspace specification box selection procedure

26) Clause C.10.2.4

Replace the definition of the `max-bandwidth` *field by the following:*

```
max-bandwidth = "mbw:" mbw
mbw = NONZERO ["K" / "M" / "G" / "T"]
```

27) New clause C.10.2.8

Insert the following text under C.10.2.8 after C.10.2.7:

C.10.2.8 Conciseness preference

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

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

Table C.12 – Conciseness preferences

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

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

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

28) Clause D.1.2

Replace the list of `jpip response headers` *by the following:*

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

29) Clause D.1.3.5

Replace the body of D.1.3.5 by the following:

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

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

30) New clause D.1.4

After D.1.3, add D.1.4:

D.1.4 Impact of errors on the server state

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

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

Example: Consider a client issuing the invalid request:

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

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

Example: Consider a client issuing the valid request:

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

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

31) Clause D.2.2

Change the first sentence of D.2.2 by:

The server shall send this response header if the server's unique target identifier differs in any way from the identifier supplied with a Target ID request field.

32) Clause D.2.3

Change the ABNF definition of JPIP-cnew *as follows:*

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

33) Clause D.2.6

At the end of D.2.6, add a note:

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

34) Clause D.2.8

Replace the definition of `JPIP-fvsiz` *by the following:*

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

35) Clause D.2.9

Replace the definition of `JPIP-rvsiz` *by the following:*

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

36) Clause D.2.10

Replace the definition of `JPIP-rvoff` *by the following:*

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

37) Clause D.2.23

Replace the first paragraph of D.2.23 (Unavailable preference) *by the following:*

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

38) New clause D.2.24

At the end of D.2, add D.2.24:

D.2.24 Alignment (JPIP-align)

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

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

39) New clause D.2.25

At the end of D.2, add D.2.25:

D.2.25 Subtarget (JPIP-subtarget)

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

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

40) Clause D.3

In D.3, change the third paragraph to the following, and replace the fourth paragraph with the Note below:

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

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

41) New Annex J

Insert the following annex between Annexes I and J. The current annexes J to M shall be renamed to Annexes K to N and references shall be fixed accordingly.

Annex J

Profiles and variants for interoperability and testing

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

J.1 Introduction

This annex provides the framework, concepts, and methodology for establishing interoperability, and the criteria to be achieved to claim compliance to Rec. ITU-T T.808 | ISO/IEC 15444-9. This annex also provides a methodology for testing compliance within a set of defined profiles and variants. The objective of standardization in this field is to promote interoperability between JPIP servers and clients and to enable testing of these systems for compliance to this Specification.

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

NOTE – Even though the testing procedures, profiles and variants compiled in this annex are defined for images encoded in some parts of the JPEG 2000 family (Rec. ITU-T T.800 | ISO/IEC 15444-1) only, and only a limited subset of features of JPIP is tested, this shall not imply that servers or clients using means of JPIP to deliver images in other formats or by other means of JPIP not listed here are not compliant. It only means that their compliance is not defined within the limits of this annex, and that there is currently no recommended testing policy and classification for them.

J.1.1 Profiles

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

J.1.2 Variants

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

J.2 Definition of variants

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

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

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

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

This parameter defines the image return type a server is able to deliver and a client is able to interpret. Servers in the **P** variant are able to deliver JPP streams; servers in the **T** variant are able to deliver JPT streams; servers in the **R** variant

are able to deliver "raw" image return types. Clients in the **P** variant accept JPP streams, clients in the **T** variant accept JPT streams and clients in variant R accept raw images. The **P, T** and **R** variants are not mutually exclusive; servers or clients may support several variants.

J.2.2 State model variant (N or S)

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

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

J.2.3 Bitstream variant (M or C)

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

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

Table J.1 – Defining requirements of variants

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

J.3 Definition of profiles

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

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

J.3.1 Profile 0: "Basic Communication"

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

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

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

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

J.3.2 Profile 1: "Enhanced Communications"

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

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

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

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

J.3.3 Full profile

The full profile provides capabilities beyond all lower profiles up to everything specified in Rec. ITU-T T.808 | ISO/IEC 15444-9.

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

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