International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.808

Amendment 4

(05/2010)

SERIES T: TERMINALS FOR TELEMATIC SERVICES

Still-image compression – JPEG 2000

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

Amendment 4: JPIP server and client profiles

Recommendation ITU-T T.808 (2005) – Amendment 4

ITU-T T-SERIES RECOMMENDATIONS

TERMINALS FOR TELEMATIC SERVICES

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

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

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

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

Amendment 4

JPIP server and client profiles

Summary

Amendment 4 to Recommendation ITU-T T.808 (2005) | ISO/IEC 15444-9:2005 defines the framework, concepts and methodology for establishing JPIP interoperability and the criteria to be achieved to claim compliance to Recommendation ITU-T T.808 | ISO/IEC 15444-9. This amendment defines JPIP *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. Variants define which features of the JPIP standard are used to request and transmit data between client and server. 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 support. A new annex is also included in this amendment that defines a methodology for testing compliance within the set of defined profiles and variants. Test data and scripts for JPIP testing are provided as an integral part of this amendment.

This amendment has an electronic attachment containing test data and scripts for JPIP testing. This attachment is also available online from the ITU-T Test Signal Database at http://itu.int/net/ITU-T/sigdb/speimage/Tseries-s.htm#T.808.

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T T.808	2005-01-08	16
1.1	ITU-T T.808 (2005) Amend. 1	2006-05-29	16
1.2	ITU-T T.808 (2005) Cor. 1	2007-01-13	16
1.3	ITU-T T.808 (2005) Amend. 2	2007-08-29	16
1.4	ITU-T T.808 (2005) Cor. 2	2008-06-13	16
1.5	ITU-T T.808 (2005) Amend. 3	2008-06-13	16
1.6	ITU-T T.808 (2005) Amend. 4	2010-05-22	16

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

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"
```

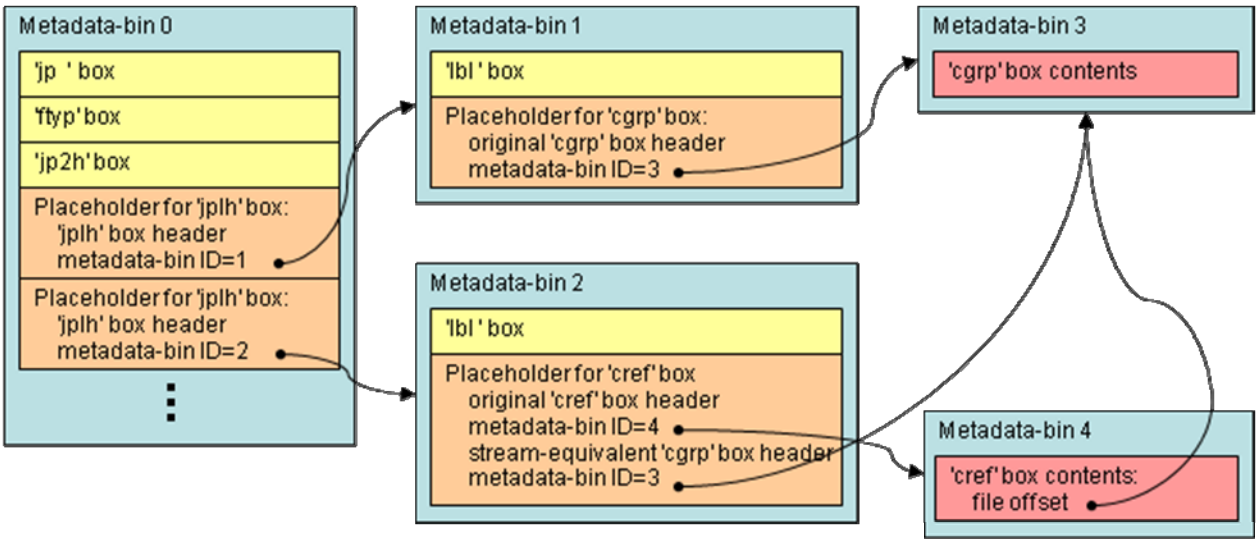
11) Clause 5.2

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:

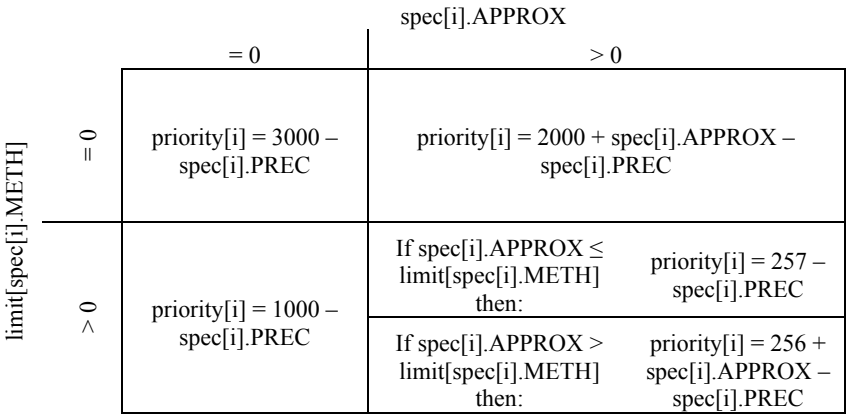


Figure C.3 – Colourspace specification box selection procedure

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

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

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

J.4 Testing methodology

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

J.4.1 Server conventions required for testing

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

J.4.2 Server testing

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

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

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

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

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

NOTE 2 – Per the test procedure defined in J.4.3, the following differences with the example stream are allowed:

- Reordering of the databins within the response.
- Relocating the contents of metadata boxes into placeholder boxes.
- Using equivalent encodings for the VBAS headers of the databins.
- Using a different break-up of metadata into placeholder bins, provided that the requested metadata is included in the response.
- Using stream-equivalent representations of metadata.

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

NOTE 3 – A test tool ("vbasdiff.py") is provided that implements this test procedure and analyses the difference between two server responses.

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

J.4.3 Comparing server responses

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

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

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

J.4.3.1 Comparing the size of the server response

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

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

J.4.3.2 Parsing stage

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

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

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

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

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

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

J.4.3.3 Abstracting from the metadata-bin layout

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

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

– Metadata-bin #0 is scanned in the test stream for incomplete box headers. Test and example streams are then made comparable by marking corresponding ranges as missing data in both streams. The modified streams are then checked for placeholder boxes. If the flags value of a placeholder box has its LSB set, indicating that the OrigID is valid, the placeholder box will be handled as indicated in the next steps; otherwise, it will remain in its unmodified form:

  • If the databin referenced by the placeholder box is included in the stream, the box will be replaced by the box header and bin contents referenced within.

  • If the databin referenced in the placeholder box is not included in the stream, the placeholder box will be removed, the box header in the placeholder box will be inserted into the stream, and the byte range in the missing target bin will be marked as missing data.

– After metadata-bin #0 of test and example streams have been parsed as above, all remaining metadata-bins except bin #0 are removed from the stream.

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

J.4.3.4 Comparing databins

In the third step, all remaining databins shall be compared, locating for each bin-class, bin-Id and CSn value in one stream the corresponding bin in the second stream: a databin must be present in the test stream if, and only if, it is present in the example stream; otherwise, the comparison fails.

Databins are compared as follows:

– If one of the databins carries AUX values, the other bin must carry AUX values. If both bins carry AUX values, they must be equal; otherwise, the comparison fails. See J.4.3.1 how to compute the AUX value of a bin from the AUX values of the messages contributing to the bin.

– If the example databin contains a message that indicates that the "last" byte of the bin has been included, the corresponding databin in the stream under testing must also contain a message with such an indicator. Otherwise, the comparison fails.

– For all bin types except the main header databin and tile header databins, all defined bytes in the databins being compared must compare equal. Otherwise, the comparison fails. The number of excess bytes in the test stream but not in the example stream is to be summed up.

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

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

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

J.4.4 Client testing

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

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

NOTE – The client testing procedure described here only tests the ability of clients to parse jpp or jpt streams successfully, and to regenerate a JPEG 2000 compliant file or codestream from such data. It does not test the ability of clients to create requests or exploit other capabilities offered by Rec. ITU-T T.808 | ISO/IEC 15444-9.

J.4.4.1 Comparing metadata

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

- Include additional UUID boxes not present in the example stream.
- Reorder the boxes, provided this does not change the semantics of the file.

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

J.4.4.2 Comparing reconstructed image data

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

- Set `fx'=Xsiz-XOsiz` and `fy'=Ysiz-YOsiz` where `Xsiz,XOsiz` and `Ysiz,YOsiz` are taken from the SIZ marker of the relevant codestream.
- Set `ox,oy` and `sx,sy` to the region offset and region size of the view window that had been defined in the request, and set `fx` and `fy` to the frame size that had been defined in the request.
- The region size `sx'` and `sy'`, and offset `ox'` and `oy'`, associated with the codestream image region are then determined by:

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

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

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

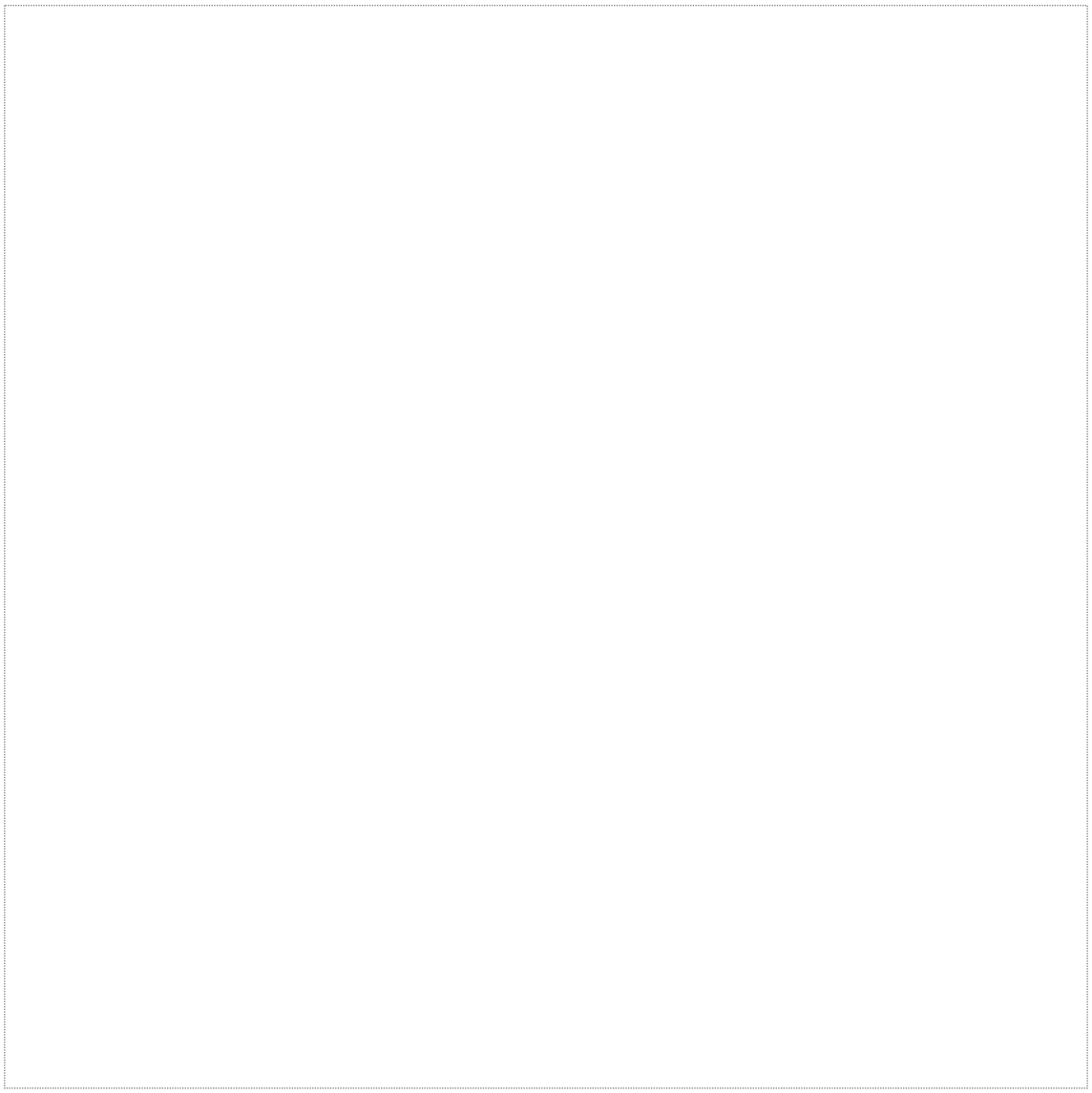
NOTE 2 – A tool ("jp2file.py") is provided in the electronic attachment to this Recommendation | International Standard that generates a textual representation of the contents of JPEG 2000 files or codestreams to ease the analysis of the data generated by the client.

NOTE 3 – The entire client test procedure requires vendors to provide a JPEG 2000 conformant decoder implementation, which is, however, not required for conformance to Rec. ITU-T T.808 | ISO/IEC 15444-9. The ability of a client to create a JPEG 2000 file or codestream is also not required to be compliant to Rec. ITU-T T.808 | ISO/IEC 15444-9 but is required for performing the tests of this annex only.

42) Bibliography

Add the following to the Bibliography:

[12] Richter, TH., Brower, B., Martucci, S., and Tzannes, A. (2009), *Interoperability in JPIP and its Standardization in JPEG 2000*, Part 9, In: A. Tescher (Ed.): Applications of Digital Image Processing XXXII, Proc. SPIE 2009.



SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	**Terminals for telematic services**
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2010