ISO/IEC JTC 1/SC 29/WG 1 N 2935

Date: 2003-05-16

ISO/IEC JTC1/SC29/WG1
(ITU-T SG8)

Coding of Still Pictures

JBIG	JPEG
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE: **Result of Core Experiments on "Header error protection" (JPWL C01)**

SOURCE: **D. Nicholson, C. Lamy, C. Poulliat, X. Naturel**

PROJECT: **JPEG 2000 part 11**

STATUS:

REQUESTED ACTION: **To be presented at WG1 JPWL intermediate meeting in Lausanne, May 2003**

DISTRIBUTION: **WG1 web pages**

Contact:
ISO/IEC JTC 1/SC 29/WG 1 Convener - Dr. Daniel T. Lee
Yahoo!, Sunning Plaza, Ste. 2802, 10 Hysan Avenue, Causeway Bay, Hong Kong
Tel: +1 408 349 7051/+852 2882 3898, Fax: +1 253 830 0372, E-mail: dlee@yahoo-inc.com

Backward Compatible Header Error Protection in a JPEG 2000 codestream[1]

1. INTRODUCTION

DURING the establishment of JPEG 2000 standard [1], a set of error resilience tools have been selected, for the transmission of JPEG 2000 compressed images in an error prone environment. Two types of tools are available, on the packet level, which enable synchronisation, and on the entropy coding level, enabling error detection. An analysis of the performances of such tools has been done in [2] and [3]. These tools are however based on one major hypothesis, namely that the headers (Main Header and Tile-part(s) header(s)) of the codestream syntax are guaranteed to be error free. However, in the case of error within the headers, the codestream is not decodable in a proper way, which might conduct to a decoder application crash. The worse is that, generally, it might not be possible to guarantee that the headers will be kept free of errors in many applications.

In order to extend the error resilience to headers and avoid (or strongly limit) the decoding crashes due to error presence, it is consequently necessary to define an error protection mechanism for protecting Main Header and Tile-part(s) header(s). It is to be noted that this protection can be extended to the packet headers when they are relocated within the Main and Tile-part(s) header(s) thanks to the packed packets functionality of JPEG 2000. Such a protection can either be provided by an external coding scheme, such as the unequal error protection solutions proposed in [4]-[5], or directly be supplied by an embedded mechanism in the JPEG 2000 syntax. The drawbacks of external schemes are well known: in particular, the error correcting scheme must whether have full knowledge of the bistream syntax (resulting in the necessity to transmit extra information between the two coders), or protect it partially blindly (*i.e.* less efficiently, for instance by using statistical ratios to determine the various bitstream parts). Moreover, such schemes cannot take advantage of the various source coder functionalities such as the headers reordering.

As a consequence, it is proposed in this document to develop a scheme that embeds the protection within the JPEG 2000 stream. Naturally, such a mechanism can be in practice whether compatible with JPEG 2000 or not. It was chosen to present here a backward compatible mechanism.

The work presented in this document implements the backward compatible header error protection proposed in wg1n2851 document. From implementation some modifications have been done in the proposed Error Protection block marker segment. The present document is therefor an update of wg1n2851 document.

This paper is organised as follows: Section 2 presents the scheme proposed to ensure error protection of the JPEG 2000 codestream, describing the proposed new marker and its functionalities. Section 3 presents the simulation conditions and the corresponding results obtained. Finally some conclusions are drawn in Section 4 and perspectives are presented. In Annexes, ready text for inclusion in JPWL Working Draft is proposed.

2. A JPEG 2000 Backward compatible error protection scheme for headers

In this section, the structure of a new marker segment for JPEG 2000 is proposed, that will allow to perform header protection. In a first step, elements on JPEG 2000 syntax, on the implications of the wished backward compatibility and on error correction codes are given. Then, the marker segment syntax is detailed and default error correction codes are proposed.

2.1. JPEG 2000 codestream syntax

A JPEG 2000 compressed image uses markers and marker segments to delimit and signal the compressed information, organised in headers (Main and Tile-parts) and packets. This modular organisation allows flexible bitstream organisation for progressive data representation, such as quality progressive and resolution progressive data progression. A JPEG 2000 codestream always starts by the Main Header followed by one or several Tile-

[1] This work was supported by the European Commission through the European project 2KAN (IST-2001-34096).

Didier. Nicholson, Catherine. Lamy and Xavier. Naturel are with THALES Communications France, TRS/TSI, 92231 Gennevilliers Cedex, France (e-mail: surname.lastname@fr.thalesgroup.com).

C. Poulliat is with the ENSEA-Univ. Cergy Pontoise-CNRS UMR8051, Equipe de Traitement des Images et du Signal. (e-mail: poulliat@ensea.fr), 95014 Cergy-Pontoise Cedex.

part Headers, each of them followed by compressed data packets, and ends by an End Of Codestream (EOC), as shown in Fig. 1.

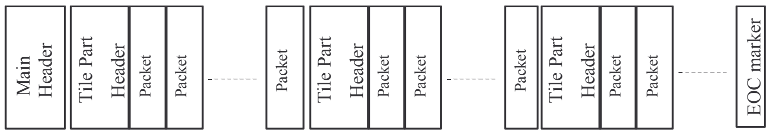

Fig. 1. JPEG 2000 codestream structure.

For more information on JPEG 2000 standard, please refer to [1].

2.2. Backward compatibility requirements

The objective being to obtain a codestream compliant with JPEG 2000 Part 1 specifications [1] after the insertion of the redundant information, it is necessary to place this information in such a way that any JPEG 2000 Part 1 decoder won't try to interpret it. A solution for this is to insert the redundant information in a dedicated marker segment. This marker will be denoted by EPB for Error Protection Block in the rest of this paper. A JPEG 2000 Part 1 compliant decoder will then skip the (from itself) unknown marker segment and be oblivious to the added data, whereas a JPEG 2000 Part 11 compliant decoder will be able to interpret and use the redundancy for header protection.

The conditions for such a mechanism to work in the context of JPEG 2000 are:

- that the decoder is able to locate the redundant information data block in the codestream without generating complex data indexing mechanism (that would also have to be protected against errors) nor with modifying the first marker segments imposed for the backward compatibility;
- that the marker itself and its length are included in the data range to be protected;
- that a defined block error code is used to protect at least up to the Error Protection Block marker segment parameters data (which, as later are proposed to be the marker, Length, index, EPB parameters), allowing the recovery of the marker boundaries.

To meet the first of these requirements, a solution is to place the marker segment with the redundant information immediately after the mandatory markers, that is to say:

- the SOC and SIZ marker segments for the Main header;
- the SOT marker for the Tile-part header.

Then, the use of a forward error-correction mechanism such as the one proposed in the following section will ensure that these two conditions are verified. As, for backward compatibility reasons, the original data must be kept untouched, and the redundant information must be concatenated to it and located in a dedicated JPEG 2000 marker segment, a systematic error correction mechanism shall be used.

2.3. Forward Error Correction mechanism

Error correction and detection codes are traditionally used to provide forward error correction capabilities in error prone environments. Considering that JPEG and JPEG 2000 codestreams are byte aligned, it is especially interesting to work with the Galois Field GF(2^8) to provide error-correction capability. A well-known and well-suited family of systematic codes in this context is the Reed-Solomon (RS) one [7]. In the following, we will consider the example of RS codes as our FEC codes for header protection, and denote them by RS(N,K), where N is the codewords symbol length and K the number of information symbols.

The RS(N,K) applied to K bytes will generate N-K redundancy bytes, that may be placed after the K original (systematic) bytes, this process being applied as long as necessary, as illustrated in Fig. 2.

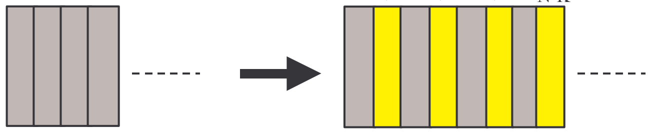

Fig. 2. Example of redundancy generation with an RS(N,K) code.

In order to remain compliant with JPEG 2000 Part 1 specifications, it was chosen to place the EPB marker segment immediately after the mandatory bytes of the considered header. Fig. 3 illustrates this disposition in the Main header case. Those mandatory bytes, together with the beginning of the EPB (in practice, the marker and its length information) constitutes then a first set of data, whose L1 bytes is protected by L2 redundancy bytes placed immediately after (in the EPB marker data). This way, the bitstream remains compliant to JPEG 2000 Part 1 and if the error-correcting code used for protection is fixed, the decoding can take place without requiring the transfer of an extra (and unprotected) information. This L1+L2 byte section can for instance be generated by an RS(L1+L2,L1) code.

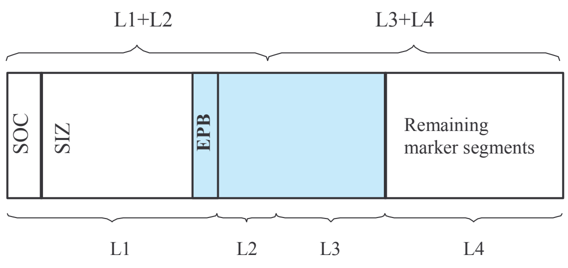

Fig. 3. Position of the EPB marker in the JPEG 2000 codestream in the Main Header case.

The remaining marker segments (L4 bytes) can then be protected by the remaining L3 bytes of the EPB marker segment, for instance with an RS(L3+L4,L4) code.

2.4. Proposed EPB marker segment description

The proposed Error Protection Block (EPB) marker segment contains information about the error protection parameters and data used to protect the headers against errors. The EPB syntax allows to use more than one EPB marker segments in a header, providing then a great flexibility in terms of size and redundancy, and to define future additional error correcting codes.

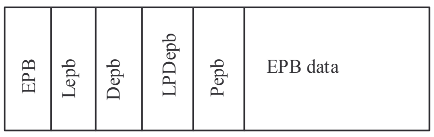

Fig. 4. Description of the proposed Error Protection Block syntax.

Fig. 4 describes the syntax proposed for the EPB marker segment, where the various fields of the marker are defined as follows:

EPB: marker code (Length 16 bits);
Lepb: length of marker segment in bytes (not including the marker) (Length 16 bits);
Depb: EPB style (for example defines if the current EPB is the latest in the current header) (Length 8 bits);
LDPepb: length of the data to be protected by the redundant information (EPB data) carried within the current EPB (Length 32 bits);
Pepb: EPB Parameters: This defines the next Error correction code to be used for protecting the remaining data. (Length 32 bits);
EPB data: contains the data enabling the correction (typically redundancy bits) for the chosen RS code.

It is to be noted that the error-correcting code settings (*i.e.* the definition of the code protecting the remaining marker segments) can be specified in the beginning of the EPB marker segment in the Pepb parameter. The only restriction is that , these parameters have then to be of fixed size, as they will have to be protected by the first (fixed) code.

For efficiency reasons, it is necessary to consider that there could be more that only one EPB in the header. In fact, the Main Header or Tile-part Header size can be quite large when optional markers such as PPM (packed packet headers) are included. As a consequence, it is useful to include within the EPB syntax an EPB index (in Depb) which will enable the presence of several EPBs in the header. By default, one could consider that if the index is set to 0, the EPB block is the only one present, otherwise the EPBs are grouped together.

Another element to be considered is the large variations of the mandatory fields size, in particular in the Main Header. As a matter of fact, the number of components in the image may vary very much, even though most of the images used have in practice up to three or four components. Considering that this number of components directly impacts the size of field SIZ, which itself must be protected by the first fixed code, the dimensioning of the first code to the maximal possible size of SIZ field would lead to dramatic compression efficiency losses. As a consequence, it is useful to consider that by default the EPB will be dimensioned to match the most common cases, typically up to three components. Should the image contain more components, a second or more EPB marker placed immediately after the first one will protect the rest of the mandatory fields (typically the end of SIZ). At the decoding side, an hypothesis test on the number of image components will be carried out using the expected associated redundancy. The decoded number of components will be the one with the lowest error detection test.

For more detailed information on the proposed EPB, please refer to [6].

2.5. Default error correction codes

A default error correction code has been defined to be used in the simulation tests for protecting data preceding EPB redundant data. As previously mentioned, it is mandatory to define this default code for the two following reasons:

- the EPB marker segment cannot define the error correction code to be used for itself;
- the position of the first EPB marker segments included in the Main Header is not known precisely and depends of the number of component of the image, which impact the size of the SIZ marker segment. This constraint imposes to perform first a synchronisation based upon the error detection process.

When the number of data to be protected by RS(N,K) is not a multiple of K bytes, it is necessary to generate padding bytes. In order to limit this byte padding, it is interesting to chose codes astutely, namely codes leading to a total size L1+L2 in bits divisible by 8. Considering that one and three image components are the most common case, and considering the proposed EPB structure, we selected as default code the code already proposed in [8] for the main header, *i.e.* the Reed-Solomon code RS(128,64). This code allows to correct up to 32 erroneous bytes. Note however that the use of a BSC channel, which means that the errors are uniform across the codestream, is not optimal for the Reed-Solomon code, as when the error bits are distributed in different bytes, they may lead to an error correction limit of only 32 bits (in the worst case). Distributed in the 128 bytes (1024 bits) of the EPB for the main header, those 32 error bits account for a BER of 3.10^{-2}, which is clearly our error correction capacity, as can be seen in the results presented in Section 3.

In order to limit byte padding two other default codes have been used:

- RS(50,25) for the first EPB marker segment of a Tile-part Header;
- RS(25,13) for the non-first EPB marker segments of both the Main Header and the Tile-part header.

3. Results

In this section, some results about above described protection scheme are provided. In a first step, compliance tests are carried out, to address the problem of the backward compatibility. Then, tests on the robustness to errors of the detection of the number of image components are proposed. Finally, a comparison with the case without header protection is done in presence of noise, and the number of decoding leading to a decoder crash are counted.

3.1. Backward compatibility compliance tests

To ensure that the insertion of the EPB marker segments described in Section 2 allows as foreseen to keep backward compatibility with JPEG 2000 Part 1 decoder, compliance bitstreams (JPEG 2000 Part 4) embedding the EPB marker segments have been generated and tested with the different JPEG 2000 reference software. Those bitstreams, which are available for testing on the JPEG 2000 member's website included in document n2935, were fully decoded and provided the same output as their counterparts without EPB marker segments.

3.2. Synchronisation tests

As mentioned before, JPEG 2000 allows to work with multi-component images. As a result, the size of the headers to be protected increases with the number of components, and it was chosen to perform as first step before error correction a detection of the number of components of the image. This detection can be done by considering a hypothesis test on the number of image components (typically 1 or 3 which are the most likely figures in practice) using the expected associated redundancy. The more likely number of components is the one with the lowest error detection test. This detection can be viewed as synchronisation test, as we try to recover the start of redundancy in the compressed bitstream.

Tests have been done to determine the robustness of such a mechanism in presence of errors. The case of a Binary Symmetric Channel (BSC) with various resultant Bit Error rates (BER) was considered. For a given image and a given number of components, the percentage of good detections of the number of components has been estimated.

TABLE I
PROBABILITY OF GOOD DETECTION FOR ONE AND THREE COMPONENTS IMAGES CORRUPTED BY A BSC CHANNEL.

BER on the BSC	Image "Bike"	Image "Lena"
10^{-3}	1.0	1.0
10^{-2}	1.0	1.0
2.10^{-2}	0.99995	0.999
3.10^{-2}	0.908	0.850
4.10^{-2}	0.389	0.267
5.10^{-2}	0.049	0.022
6.10^{-2}	0.002	0.001
7.10^{-2}	0.0	0.0
10^{-1}	0.0	0.0

Table I shows the results obtained for image "Bike" (2048x2560, 1 component) and image "Lena" (512x512, 3 components), both compressed at 0.5 bpp (bit per pixel) for various BERs. The probabilities of good detection were derived for 10000 independent noise realisations. For each realisation, the channel bit error rate constant through the image, which means that the compressed bitstream (headers and data) is uniformly corrupted.

It can be seen that a highly reliable detection of the number of image components is obtained for a large range of BER. Of course, the error correction capabilities depend from the capabilities of the considered RS codes, which explains the lower performances at high BERs. Better results (a larger BER range where good detection are obtained) would be obtained by taking a more powerful code.

3.3. Crash decoding tests

Having tested the performance of the synchronisation process, we now consider the overall performance of the proposed system by tracking performance in terms of crash decoding. Errors occurring in header may indeed conduct to a crash of the decoding process, or to place it in a never-end state. As the focus of this paper is the study of error detection and correction mechanisms for the main and tile-part headers, our purpose is to show that such crashes in the decoding process can be avoided using the proposed header protection mechanism. Note that when no decoder crash occurs, *i.e.* when an output image is produced by the decoder, this image may still result from a codestream that include errors in the headers. In this case, the decoded image may or may not be damaged seriously, depending on the location of the error(s). For example an error occurring in an optional marker segment may have no impact on the decoded image, whereas an error in the image size will conduct to a bad result. In this section we consider only the fact that images can be decoded without freezing or crashing the decoder, whatever the result. It is to be noted that the crash behaviour strongly depends of the decoder implementation, and that given performances are only valid with the decoder used in the experiments, *i.e.* the Verification Model version 8.6.

Once again, a BSC channel was considered for our experiments. The corresponding results obtained for both the mono and multi tile-part cases with the mechanism defined in Section 2 are compared to those obtained without using backward compatible error correction capabilities.

As reference, Table II shows the results obtained for the "Lena" image compressed at 0.5 bpp, with one tile and one tile-part, when no EPB marker is considered. The probability of crash avoidance was derived for 1000 independent noise realisations.

TABLE II
PROBABILITY OF AVOIDING A DECODING CRASH FOR "LENA" IMAGE CORRUPTED BY A BSC CHANNEL.

BER on the BSC	Image "Lena"
10^{-4}	0.94
10^{-3}	0.63
10^{-2}	0.03
2.10^{-2}	0.0
3.10^{-2}	0.0
4.10^{-2}	0.0
5.10^{-2}	0.0
10^{-1}	0.0

Those results illustrate the dire need for header protection, as it is obvious that even for a BER as low as 10^{-4}, the decoder may crash, leading to an interruption of the service, the obligation to re-initialise the decoder ("reboot"), …. Let now compare these results to those obtained for the same image (still compressed 0,5 bpp) with the EPB marker embedded.

Table III shows the results obtained firstly in the case where a unique tile-part is used and secondly in the case where the image is coded with five tile-parts. Once again, the probabilities of crash avoidance were obtained for 1000 independent noise realisations.

TABLE III
PROBABILITY OF AVOIDING A DECODING CRASH WITH THE EPB MARKER SEGMENT FOR "LENA" IMAGE CORRUPTED BY A BSC CHANNEL, IN THE MONO OR MULTI TILE-PART CASE.

BER on the BSC	Image "Lena" 1 tile-part	Image "Lena" 5 tile-parts
10^{-4}	1.0	1.0
10^{-3}	1.0	1.0
10^{-2}	1.0	1.0
2.10^{-2}	0.966	0.965
3.10^{-2}	0.672	0.567
4.10^{-2}	0.134	0.036
5.10^{-2}	0.003	0.0
10^{-1}	0.0	0.0

The results obtained in those various tables clearly show the interest of the proposed EPB mechanism, for it provides a clear improvement of the decoder performance in presence of transmission errors in the JPEG 2000 headers. Typically, for a bit error rate of 2.10^{-2} on the channel, almost all images are decodable (in more than 96% of the cases) whereas the classical image systematically leads to a decoding crash. This is even more interesting those results are obtained both for the case of mono and multi tile-part, and for a very reduced overhead cost (about 0.35% for "Lena" images and about 0.11% for "Bike" image), which reflects from the authors point of view the flexibility and efficiency of the proposed header EPB marker segment technique.

4. Examples of Headers protected by EPB marker segments

4.1. Main Header error protection

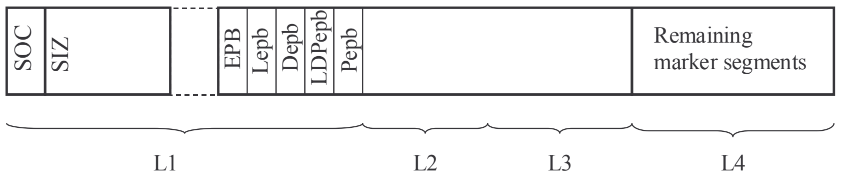

Figure 1 - EPB marker position in Main Header and protection regions

The first EPB marker segment included in the Main Header has to be located immediately after the SIZ marker segment. This last marker size depending on the number of image components, it is necessary at the beginning of the error correction process to synchronise using error detection in order to find the location of the first EPB marker segment. The Main Header can contain several EPB marker segments, that may be unpacked or packed, which mean that they appear one after each other, before the remaining Main Header information. Unpacked EPBs means that they will appear just before the data part they refer to. An example of packed and unpacked EPBs is given hereafter.

4.2. Tile-Part Header error protection

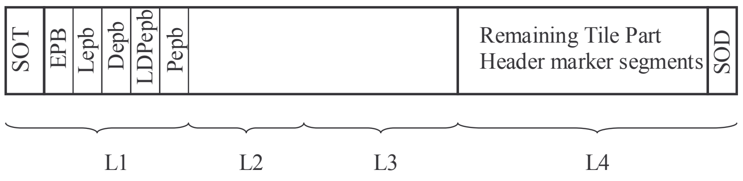

Figure 2 - EPB marker position in Tile Part Header and protection regions (single EPB case)

Figure 2 illustrates the case where a single EPB marker segment is used to protect the Tile Part header. In this case, L1 data are protected by the L2 part of the EPB data, using the default Tile-Part header error correction code. L4 data are protected using L3, with the error correction code specified in the Pepb parameter.

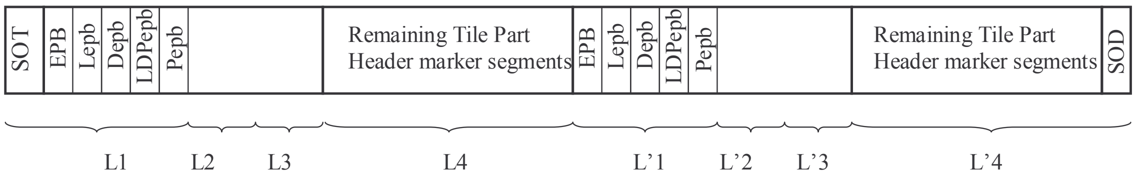
SOT
EPB
Lepb
Depb
LDPepb
Pepb
Remaining Tile Part Header marker segments
EPB
Lepb
Depb
LDPepb
Pepb
Remaining Tile Part Header marker segments
SOD
L1
L2
L3
L4
L'1
L'2
L'3
L'4

Figure 3 - unpacked EPB markers position in Tile Part Header and protection regions (several EPB case)

Figure 3 illustrates the case where unpacked two EPB marker segments used to protect the Tile Part header. In this case, L1 data are protected by the L2 part of the first EPB data, and L'1 data are protected by the L'2 part of the second EPB data , using the default Tile-Part header error correction code. L4 data are protected using L3, with the error correction code specified in the Pepb parameter of the first EPB marker segment. L'4 data are protected using L'3, with the error correction code specified in the Pepb parameter of the second EPB marker segment. This structure allows to protect differently fundamental marker segments, such as QCD, whereas optional marker segments like PLT can be protected with less redundancy or even not protected at all.

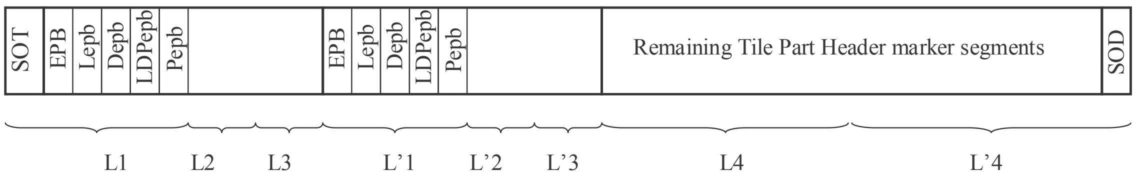

Figure 4 - packed EPB markers position in Tile Part Header and protection regions (several EPB case)

Figure 4 illustrates the case where packed two EPB marker segments used to protect the Tile Part header. In this case, L1 data are protected by the L2 part of the first EPB data, and L'1 data are protected by the L'2 part of the second EPB data , using the default Tile-Part header error correction code. L4 data are protected using L3, with the error correction code specified in the Pepb parameter of the first EPB marker segment. L'4 data are protected using L'3, with the error correction code specified in the Pepb parameter of the second EPB marker segment. This structure allow to protect differently fundamental marker segments, such as QCD, whereas optional marker segments like PLT can be protected with less redundancy or even not protected at all.

5. Conclusions and perspectives

A fully JPEG 2000 Part 1 compliant backward compatible error protection scheme for headers was proposed in this paper. The description of a new normative embedded marker segment was provided and tested with Reed-Solomon codes. Simulation tests were shown, that show that the performance of the decoder can be significantly improved in terms of crash decoding, considering mono or multi tile-parts, with a reasonable decrease of compression efficiency.

As an immediate perspective of interest, using the particularly flexible structure of the EPB marker segment, embedded Unequal Protection schemes can be envisaged for joint or tandem Source-channel coding.

6. References

[1] *JPEG 2000 image coding system.* ISO/IEC 15444-1/ IUT-T T.800.

[2] L. Liang., B. Luc, A. Lie, J. Wus and F. Kossentini, "Error resilience Ad-hoc Sub-Group Report for the Tokyo Meeting ", *ISO/IEC JTC 1/SC 29/WG 1*, N1606, Tokyo, Japan,7th Mar. 2000.

[3] C. Pouillat and D. Nicholson, "Impact and efficiency of error resilience tools for mobile applications", *ISO/IEC JTC1 SC29 WG1* N2176, Stockholm, Sweden, July 2001.

[4] A. Natu and D. Taubman "Unequal Protection of JPEG 2000 Code-Streams in Wireless Channels", *Proceedings of IEEE GLOBECOM'02*, vol. 1, pp. 534-538, Taipei, China, 17-21 Nov. 2002.

[5] V. Sanchez and M.K. Mandal, "Robust transmission of JPEG 2000 images over noisy channels, " *Proceedings of IEEE ICCE'02*, pp. 80-81, 2002.

Annex A Proposed text for JPWL Working Draft

7. Markers, marker segments: codestream syntax

7.1. *Definitions of markers and marker segments*

It is proposed in this document to use marker segments to delimit and signal the characteristics of the codestream protected against errors. This proposed set of markers and marker segments is the minimal information needed to achieve the features of header protection in JPWL. Not that it does not impact any file format.

For backward compatibility, the JPWL markers and marker segments must be included in standard JPEG-2000 codestream headers, which can be of two types only:

1. the main header, found at the beginning of the codestream,
2. the tile-part headers, found at the beginning of each tile-part,

Main and tile-part headers are collections of markers and marker segments. The main header is found at the beginning of the codestream. The tile-part headers are found at the beginning of each tile-part.

As for every other standard marker defined in Part 1 of JPEG 2000, each marker defined in this proposal is two bytes long, and its first byte value is 0xFF. The second byte specifies the marker use and can take any value in the range 0x01 to 0xFE, apart from those already used by the ITU-T Rec. T.81 | ISO/IEC 10918-1 and ITU-T Rec. T.84 | ISO/IEC 10918-3 (recalled in Table 1).

A marker segment includes a marker and associated parameters, called marker parameters. By definition, the first two bytes of any marker segment immediately after the marker must correspond to an unsigned big endian integer value that denotes the length in bytes of the marker parameters (including two bytes of this length parameter but not including the two bytes of the marker itself). When the decoder finds a marker segment that is not specified in ITU-T Rec. T.800 | ISO/IEC 15444-1 in a codestream, it shall use the length parameter to discard the marker segment.

7.2. *Marker code range defined in this document*

Following the syntax used for each marker and marker segment defined in ITU-T Rec. T.81 | ISO/IEC 10918-1, we proposed to reserve the marker 0xFF96 for signalling one JPWL marker in. Table 1 recalls the various values of already existing or reserved markers.

Table 1 – Marker definitions

Marker value range	Standard definition
0xFF00, 0xFF01, 0xFFFE, 0xFFC0 – 0xFFDF	Defined in ITU-T Rec. T.81 | ISO/IEC 10918-1
0xFFF0 – 0xFFF6	Defined in ITU-T Rec. T.84 | ISO/IEC 10918-3
0xFFF7 – 0xFFF8	Defined in ITU-T Rec. T.87 | ISO/IEC 14495-1
0xFF4F – 0xFF6F, 0xFF90 – 0xFF93	ITU-T Rec. T.800 | ISO/IEC 15444-1
0xFF96	Defined in this proposal
0xFF30 – 0xFF3F	Reserved for definition as markers only (no marker segments)
	All other values reserved.

8. Information in the marker segments

As standardised in JPEG 2000 part 1, marker segments, and therefore the headers, are a multiple of 8 bits (one byte). They give elements and inform on what should be applied to the tile-part header or a start of packet header All markers and marker segments in a tile-part header or a start of packet header apply only to the tile or the packet to which it belongs.
If truncation, alteration, or editing of the codestream has been done, the marker segments shall be updated accordingly.

Table 2 lists the markers specified in this proposal and Table 3 lists the information provided by the syntax and indicates the marker segment containing that information.

Table 2 – List of marker segments

	Name	Code	Main header [a]	Tile-part header [a]
Fixed information marker segment				
Error Protection Block	EPB	0xFF96	required [b]	required [b]

a. Required means the marker segment shall be in this header, optional means it may be used.
b. The EPB marker is only required from the JPWL header protection perspective, in other words: only if the codestream is JPWL protected. For a classic, unprotected codestream, it is useless, therefore “optional”, even in the main header.

Table 3 – Information in the marker segments

Information	Marker segment
Presence of JPWL protected data in the header (information given by the existence of the marker itself). It includes: Set of error protection parameters used in the bit stream Error protection data generated from a block code	EPB

9. Construction of a JPWL protected header

The JPWL codestream construction process is very similar the construction described in JPEG-2000 Part 1 and just adds this proposal's specific marker segments to the codestream. However, the EPB marker and marker segment is specific in the sense where it is required to be in a specific location.

The first EPB is required to be placed:

- immediately after the SIZ marker segment when located in the Main Header,
- immediately after the SOT marker when located in a Tile Part header.

And to use for the error protection of its first part, the corresponding default error correction code.

10. Definition of the Header Protection Block (EPB) marker segment

The EPB marker segment contains information about the error protection parameters and data used to protect the headers against errors.

Function: The EPB marker contains necessary error correction data for the header where it is located.

Usage: Main header and Tile Part Headers. The marker must be placed after the SIZ marker segment.

Length: Variable depending the parameters used to protect the headers and the length of the headers to be protected.

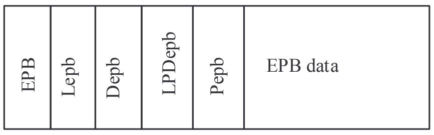

Figure 5 describes the syntax of the EPB marker segment.

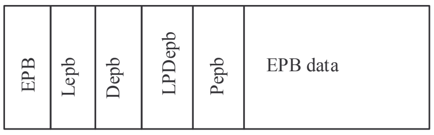

Figure 5 – Error Protection Block description syntax

EPB: Marker code. Table 4 indicates the size and parameter values of the marker symbol itself and for each parameters of the marker segment.

Lepb: Length of marker segment in bytes (not including the marker).

Depb: EPB style (for example defines if the current EPB is the latest in the current header) (Length 8 bits)

LDPepb: length of the data to be protected by the redundant information (EPB data) carried within the current EPB (Length 32 bits)

Pepb: EPB Parameters: to be defined in the future after Core Experiment results. It defines the next Error correction code to be used for protecting the remaining data. Fixed size information, current proposed size is 32 bits, but may be changed after Core Experiments.

EPB data: Contains the data enabling the correction (typically redundancy bits)

Table 4 – Error Protection Block parameter values

Parameter	Size (bits)	Values
EPB	16	0xFF96
Lepb	16	7 – (2^{16}-1)
Depb	8	See Table 5
LDPepb	32	0 – (2^{31}-1)
Pepb	32	Tbd

		Defines next Error correction code to be used in the codestream
EPB data	variable	

When the EPB is included in Main Header, the SOC marker, The SIZ marker segment, the EPB marker, the Lepb, Depb, LDPepb, Pepb data are protected with a fixed RS(N1,K1) code. The resulting error correction needed data are placed at the beginning of the EPB data.

When the EPB is included in a Tile Part Header, the SOT marker, the EPB marker, the Lepb, Zepb, Pepb data are protected with a fixed RS(N2,K2) code. The resulting error correction needed data are placed at the beginning of the EPB data..

The actual values for the fixed codes RS(N1,K1) and RS(N2,K2) are to be defined in the future after Core Experiment results.

Table 5 – Depb parameter values

Values (bits) MSB LSB	EPB configuration and index
x0xx xxxx	The EPB marker segment is not the latest in the current header
x1xx xxxx	The EPB marker segment is the latest in the current header
0xxx xxxx	EPBs marker segments are unpacked
1xxx xxxx	EPBs marker segments are packed
xx0x xxxx xxxx xxxx - xx1x xxxx xxxx xxxx	Reserved for future use

In the case where the Main or Tile Part Header is of large size, for example due to the inclusion of several PPM or PPT marker segments, the possibility of using more than one EPB marker segment is necessary. The Depb parameter, whose values allocation is proposed in Table 5, allows this functionality. This parameter allows also to indicate how the EPB information was placed in the header. There are two possibilities for chaining this information, while keeping the error protection feature among them:

1. One way is to interleave between them the marker segments corresponding of the redundant part included within the EPB marker segment. This structure is called "unpacked EPB marker segments".
2. The other way, which provides optimal redundant information length, called "packed EPB marker segments" consists in grouping together all the EPB marker segments before the remaining marker segments of the header.

In both cases, the "Last EPB marker" information allows to identify that the EPB marker segments is the latest in the header. It is particularly interesting when using packed EPB option, where it allows to locate the remaining header data, which is to be found just after the current EPB marker segment.

10.1. Default error correction codes

Two default fixed error correction code will have to be defined for protecting the beginning of both the Main Header and the Tile Part Headers. In order to fight efficiently harsh transmission conditions, these default codes will have to offer a large correction capacity. Their choice will be done thanks to the elements gathered within the Core Experiments studies, where it will in particular be looked at the size of the most frequent of error bursts in classical functioning conditions. As an example the RS(128,64) code could be considered for the Main Header, whereas the RS(44,22) code could be chosen for the Tile Part Headers.

10.2. EPB parameters

These parameters allow to select another error correction code than the default error correction code. The formatting of such parameter will be defined after Core experiments. One temporary definition is:

0x000000000 uses default error correction code (relevant in this header, either Main or Tile Part).
0xFFFFFFFF uses no error correction codes for the next data following the EPB marker segment (allow to stop the error protection from a certain location).

The specified error correcting code, has to be used for the correction of the codestream data following the EPB marker segment, for which redundant data are included within the EBP marker segments after the redundant data of the codestream data before and including the beginning of the marker segment itself.