INTERNATIONAL STANDARD

ISO/IEC 14496-1

Second edition
2001-10-01

AMENDMENT 1
2001-10-15

Information technology — Coding of audio-visual objects —

Part 1: Systems

AMENDMENT 1: Extended BIFS

Technologies de l'information — Codage des objets audiovisuels —

Partie 1: Systèmes

AMENDEMENT 1: BIFS étendus

Reference number
ISO/IEC 14496-1:2001/Amd.1:2001(E)

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

Printed in Switzerland

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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

Amendment 1 to International Standard ISO/IEC 14496-1:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects —

Part 1:
Systems

AMENDMENT 1: Extended BIFS

1) Add the following at the end of subclause 8.6.15.3.1:

"

```
class QoS_Qualifier_REBUFFERING_RATIO extends QoS_Qualifier : bit(8) tag=0x02 {
    bit(8) REBUFFERING_RATIO;
}
```

"

2) Add the following at the end of subclause 8.6.15.3.2:

"

REBUFFERING_RATIO – Ratio of the decoding buffer that should be filled in case of prebuffering or rebuffering. The ratio is expressed in percentage, with an integer value between 0 and 100. Values outside that range are reserved.

8.6.15.3.2.1 Rebuffering

In certain scenarios the System Decoder Model cannot be strictly observed. This is the case of e.g. file retrieval scenarios in which the data is pulled from a remote server over a network with unpredictable performances. In such a case prebuffering and/or rebuffering may be required in order to allow for a better quality in the user experience. Note that scenarios involving real time streaming servers do not fall in this category, since a streaming server presumably delivers content according to the appropriate timeline.

An elementary stream is prebuffered when the decoder waits until the decodingBuffer has been filled up to a certain threshold before starting fetching data from it.

An elementary stream is rebuffered when a decoder stops fetching data from the decodingBuffer and before resuming fetching data waits until that buffer has been filled again up to a certain threshold.

In order to inform a receiver whether a certain elementary stream requires prebuffering and/or rebuffering the QoS_Qualifier_REBUFFERING_RATIO qualifier can be included in the Elementary Stream Descriptor (see subclause 8.6.15.3.1). By default, in the absence of such qualifier, an elementary stream does not require pre-buffering or rebuffering.

"

3) Replace Table 31 (Compensation process) in subclause 9.3.4 by the following:

"

Table 31 — Compensation process for multiple fields and BIFS-Anim

<table>
<tr><th>quantType</th><th>animType</th><th colspan="2">Compensation Process</th></tr>
<tr><td>1,2,4,6,7,8, 9 (other than SFVec3fType), 10 (other than SFRotationType), 11,12,13</td><td>1,2,4,6,7,8 11,12,13</td><td colspan="2">The components of v_q^2 are:
vq2[i] = vq1[i] + vDelta[i]</td></tr>
<tr><td rowspan="6">9 (SFVec3fType), 10 (SFRotation)</td><td rowspan="6">9,10</td><td colspan="2">The addition is first performed component by component and stored in a temporary array:
vqTemp[i] = vq1[i] + vDelta[i].
Let scale = $2^{\max(0,nbBits-1)} - 1$.
Let N the number of reduced components (2 for normals, 3 for rotations)
There are then three cases are to be considered:</td></tr>
<tr><td>For every index I,
$|vqTemp[i]| \le scale$</td><td>v_q^2 is defined by,
vq2[i] = vqTemp[i]
orientation2= orientation1
direction2 = direction1 * inverse</td></tr>
<tr><td>There is one and only one index k such that
$|vqTemp[k]| > scale$</td><td>v_q^2 is rescaled as if gliding on the faces of the mapping cube.
Let inv = 1 if vqTemp[k]>=0 and –1 else
Let dOri = k+1
The components of vq2 are computed as follows
$0 \le i < N - dOri$: vq2[i] = inv*vqTemp[(i+dOri) mod N]
$i = N - dOri$: vq2[i] = inv*2*scale–vqTemp[dOri–1]
$N - dOri < i < N$: vq2[i] = inv*vqTemp[(i+dOri-1) mod N]
orientation2 = (orientation1 + dOri) mod (N+1)
direction2 = direction1 * inverse * inv</td></tr>
<tr><td>There are several indices k such that
$|vqTemp[k]| > scale$</td><td>The result is undefined</td></tr>
</table>

Note: The BIFS-Anim process is identical to the process applied for optimal encoding of BIFS multiple fields.

"

4) Replace the reserved bit in subclause 9.3.5.3.1 by the following:

"

bit(1) usePredictiveMFField;

"

5) *Insert the following in subclause 9.3.5.3.2 at the end of the second paragraph (on the use3DmeshCoding):*

"

The usePredictiveMFField flag is used to signal that the syntax for predictive MFField instead of the non-predictive mode is used to encode IndexedFaceSet nodes. This flag is used for terminals supporting this tool.

"

*6) Replace subclause 9.3.7.2.4 (**PROTOcode**) by the following subclauses:*

"

9.3.7.2.4 PROTOcode

9.3.7.2.4.1 Syntax

```
class PROTOcode(isedNodeData protoData) {
    bit(1) isExtern
    if (isExtern) {
        MFUrl locations;
    } else {
            PROTOlist subProtos;
    }
    do {
        SFNode node(SFWorldNodeType,protoData);
        bit(1) moreNodes;
    } while (moreNodes);
    bit(1) hasROUTEs;
    if (hasROUTEs) {
        ROUTEs routes();
    }
}
```


9.3.7.2.4.2 Semantics

First a flag signals whether the prototype is a PROTO, which then has his code included in the proto declaration, or if is an EXTERNPROTO, in which case only an external reference is provided. The EXTERNPROTO is an authoring facility that enables to distribute PROTOs in external libraries and be reused across scenes. The EXTERNPROTO opens a BIFSCommand stream that contains a ReplaceScene command with a BIFSScene with the PROTO definitions. The EXTERNPROTO code is found in the PROTO in this new scene with the same ID in this scene. The nodes that may be contained in this scene are ignored.

In case of a PROTO, the PROTOcode contains a (possibly empty) list of the sub-PROTOs of this PROTO in subProtos, followed by the code to execute the PROTO. The code is specified as a set of SFNodes, using a standard SFNode definition with the additional possibility to declare an IS field. Moreover, the PROTO body may contain ROUTEs if the hasROUTE flag is set to 1.

"

*7) Replace subclause 9.3.7.6 (**Field**) with the following subclauses:*

"

9.3.7.6 Field

9.3.7.6.1 Syntax

```
class Field(FieldData field) {
    if (isSF(field))
        SFField svalue(field);
    else {
        if (BIFSConfig.usePredictiveMFField == 1) {
            bit(1) usePredictive;
            if (usePredictive)
                PredictiveMFField mvalue(field);
            else
                MFField mvalue(field);
        }
        else {
            MFField mvalue(field);
        }
    }
}
```


9.3.7.6.2 Semantics

A field is encoded according to its type: single (SFField) or multiple (MFField). A multiple field is a collection of single fields.

"

*8) Add the following as new subclauses after subclause 9.3.7.9 (**MFVectorDescription**):*

"

9.3.7.10 PredictiveMFField

9.3.7.10.1 Syntax

```
class PredictiveMFField (FieldData field) {
    AnimFieldQP aqp = new AnimFieldQP();
    aqp.useDefault = FALSE;
    field.aqp = aqp;
    ArrayHeader   header(field);
    ArrayOfValues    values(field);
}
```

9.3.7.10.2 Semantic

The array of data is composed of a **Header**, and an **ArrayOfValues**. Note that the FieldData structure is filled as described in the BIFS-Scene quantization process (subclause 9.3.3.1 of ISO/IEC 14496-1:2001).

The process applied for optimal encoding of BIFS multiple fields is exactly identical to the BIFS-Anim process (See Table 31):

1 Compensation on the P values
2 Inverse Quantization into single field values

The compensation process uses the quant type as well as Pmin and PNbBits, defined in the ArrayQP and InitialArrayQP, and can be summarized in the following table.
The inverse quantization process uses the values of floatMax, floatMin, and NbBit as defined in the BIFS quantization process and as defined by the QuantizationParameter node.

9.3.7.11 ArrayHeader

9.3.7.11.1 Syntax

```
class ArrayHeader(FieldData field){
    uint(5)       NbBits;
    int(NbBits)      numberOfFields;
    bit(2)     intraMode;
InitialArrayQP     qp(intraMode,field);
}
```

9.3.7.11.2 Semantic

The array header contains first information to specify the number of fields (**NbBits** is the number of bits used to code the **numberOfFields**). Then the Intra/Predictive policy (**intraMode**) is specified as follows:

- 0 : Only one Intra value at the beginning and then only predictive coded values
- 1 : An Intra every given number of predictive values
- 2 : A bit for each value to determine whether the value is an Intra or predictive value

Lastly, the **InitialArrayQP** is coded.

9.3.7.12 InitialArrayQP

9.3.7.12.1 Syntax

```
class InitialArrayQP(int intraMode, FieldData field){
    switch (intraMode)
        case 1 :
            unsigned int(5)        NbBits;
            unsigned int(NbBits)    intraInterval;
            // no break
        case 0 :
        case 2 :
    int(5) CompNbBits;
    for (int i=0;i<getNbComp(field);i++) {
                        int(field.NbBits+1) vq;
                        field.aqp.Pmin[i] = vq-2^field.NbBits;
}


    }
    // no break
    case 3:
        break;
}
```


9.3.7.12.2 Semantic

If intraMode is 1, the size of the interval between two intras is first specified. Independent of the intraMode, the number of Bits used in Predictive mode CompNbBits and the CompMins are coded. The function getNbComp() is a function that returns the number of components of the quantizing bounds, and depends on the object. For instance it returns 3 for 3D positions, 2 for 2D positions, and 3 for rotations. See Table 17 (Return values of getNbComp) in ISO/IEC 14496-1:2001. CompNbBits and CompMin are stored in the field.aqp AnimationQP structure, and are used for the compensation process as defined in Table 31 (Compensation process for multiple fields and BIFS-Anim) and subclause 9.3.4 of ISO/IEC 14496-1:2001.

9.3.7.13 ArrayQP

9.3.7.13.1 Syntax

```
class ArrayQP(int intraMode, FieldData field){
    switch (intraMode)
        case 1 :
            int     NbBits;
            int(NbBits) intraInterval;
            // no break
        case 0 :
        case 2 :
            boolean(1) hasCompNbBits
            if (hasCompNbBits)
            int(5) CompNbBits;
            boolean(1) hasCompMin
            if (hasCompMin) {
            for (int i=0;i<NbComp(field)) {
                                int(field.NbBits+1) vq;
                                field.aqp.Pmin[i] = vq-2^field.NbBits;
}
            }
        case 3:
            break;
}
```


9.3.7.13.2 Semantic

ArrayQP fulfills the same purpose as InitialArrayQP, but in this case, the parameters are optionnaly set. If they are not set in the stream, they are set by default, in reference to the InitialArrayQP or the latest received value of the parameter.
If IntraMode is 1, the size of the interval between two intras is first specified. In any case, the number of Bits used in Predictive mode (**CompNbBits**) and the **CompMin**s are coded. The function getNbComp() is a function that returns the number of components of the quantizing bounds, and depends on the object. For instance it returns 3 for 3D positions, 2 for 2D positions, and 3 for rotations. See Table 17 in ISO/IEC 14496-1:2001. CompNbBits and CompMin are stored in the field.aqp AnimationQP structure, and are used for the compensation process as defined in Table 31 and subclause 9.3.4 of ISO/IEC 14496-1:2001.