INTERNATIONAL STANDARD ISO/IEC 14496-16:2011
TECHNICAL CORRIGENDUM 1

Published 2015-04-01

Information technology — Coding of audio-visual objects —

Part 16:
Animation Framework eXtension (AFX)

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage des objets audiovisuels —

Partie 16: Extension du cadre d'animation (AFX)

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 14496-16:2011 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of Audio, Picture, Multimedia and Hypermedia Information*.

Replace section 5.2.5.3.5.1 Syntax

"Class IntArrayDecoder (numberOfdata, dim)
{
*   ….*
}"
with

```
Class IntArrayDecoder(numberOfdata, dim)
{
    Bit(4) predictionMode;
    Bit(4) binarizationMode;
    If ((binarizationMode == 0) && (predictionMode == 0))          // FL
    {
```

ICS 35.040

Ref. No. ISO/IEC 14496-16:2011/Cor.1:2015(E)

Published in Switzerland

```
        unsinged int (32) streamSizeFL;
        Bit(8) QP;
        for(i=0;i< numberOfdata *dim;i++)
          bit (QP) nData; // simple QBCR
    }
    else if (binarizationMode == 1)        // BPC
    {
      unsinged int (32) streamSizeBPC;
      If (predictionMode==3)      bit(1-7) predictor;
      bit (5) prefixSize;
      for(i=0;i< numberOfdata *dim;i++)
        {BPDecoder(prefixSize) nDifData;
        If ((predictionMode==1,4,5)&&(nDifData != 0))      bit(1) nSign;
      }
    }
    else if  (binarizationMode == 2)        // 4C
    {
        unsinged int (32) streamSize4C;
        for(i=0;i< numberOfdata *dim;i++)
        {
              If (predictionMode==3)
              {
                bit(3) predictor;
                  bit (1) terminationBit;
                  while (terminationBit)
                  {
                      bit(3)      threeBitsFL;
                      bit(1)      terminationBit;
                  }
              }
              else
              {
                  Do
                  {
                      bit(3)      threeBitsFL;
                      bit(1)      terminationBit;
                  }
                  while (terminationBit)
              }
          If((predictionMode==1,4,5)&&(difValue!=0))      bit(1) signBit;
          }
    }
    else if  (binarizationMode == 3)        // AC
    {
          unsinged int (32) streamSizeAC;
          for(i=0;i< numberOfdata *dim;i++)
          {
              If (predictionMode==3)      ACDecoder(8) predictor
              ACDecoder(1<<QP) nValue
              ACDecoder(2) hasnext
              If (nValue!= 0)      ACDecoder(1) nSign
          }
    }
    else if  (binarizationMode == 4)        // AC/EGk
    {
          unsigned int (32) streamSizeACEGk;
          unsigned int (8) K
          unsigned int (8) M
          for(i=0;i< numberOfdata *dim;i++)
          {
```

```
                If (predictionMode==3)      ACDecoder(8) predictor
                ACDecoder(M+1) nDifData
          if (nDifData==M+1)   ACExpGolombDecode(K)   nDifDataEGk;
            }
        }
}
```

Add the following 4 semantics at the beginning of section 5.2.5.3.5.2 Semantics

streamSizeFL: A 32-bit unsigned integer indicating how many bytes are used for FL
streamSizeBPC: A 32-bit unsigned integer indicating how many bytes are used for BPC
streamSize4C: A 32-bit unsigned integer indicating how many bytes are used for 4C
nSign: A one bit syntax for indicating the positive or negative of nDifData

Replace Section 5.2.5.3.8.1 Syntax

"SVAIndexDecoder (numberOfIndex, numberOfData)
{
  ….
}" with

```
Class SVAIndexDecoder (numberOfIndex, numberOfData)
{
        if(entropytype == 1) // BPC case
        {
         bit (32) streamSizeNType
         bit(5) prefixSize_nType;
         for (i=1;i<numberOfIndex;i++){
         BPDecoder(prefixSize_nType) nType[i]
    }
    bit (32) streamSizeBPC
    bit (1) FDMode;
    if(FDMode == 0)
        bit (1) FaceDirection;
 bit(5) prefixSize_data;
 bit(5) prefixSize_nPosition
 bit(5) prefixSize_FaceDirection
 bit(5) prefixSize_nRotation
 for (i=0;i<3;i++)
 { // first face..
     BPDecoder(prefixSize) nData;
 }
 for (i=1;i<numberOfIndex;i++)
 {      // second to last face...
         switch(nType[i]){
           case 0:  // mode 0
                BPDecoder(prefixSize_nPosition) nPosition;
                if (FDMode == 1)
                    BPDecoder(prefixSize_FaceDirection) FaceDirection;
                BPDecoder(prefixSize_data) nDifIndex;
                If (nDifIndex!= 0)      bit (1) nSign;
                BPDecoder(prefixSize_nRotation) nRotation;
          break;
          case 1:   //mode 1
                for (j=0;j<3;j++){
```

```
                           BPDecoder(prefixSize_data) nDifIndex;
                           if (nDifIndex!= 0)    bit (1) nSign;
                   }
         break;
         case 2:  //mode 2
               BPDecoder(prefixSize_nPosition) nPosition;
               for(j = 0; j< 2; j++){
                   BPDecoder(prefixSize_data) nDifIndex;
                          if (nDifIndex!= 0)     bit (1) nSign;
                      }
                       BPDecoder(prefixSize_nRotation) nRotation;
               break;
               case 3:  //mode 3
                     for (j=0;j<3;j++){
                             BPDecoder(prefixSize_data) nDifIndex;
                             if (nDifIndex!= 0) bit (1) nSign;
                       }
               break;
               case 4:  //mode 4
                     if (FDMode == 1)
                          BPDecoder(prefixSize_FaceDirection) FaceDirection;
                     BPDecoder(prefixSize_nRotation) nRotation;
               break;
           }
}

else if (entropytype == 2) // AC case..
{
     bit (32) streamSizeNType;
     for (i=1;i<numberOfIndex;i++){       // second to last face...
        ACDecoder(mType) nType[i];
}
unsinged int (32) streamSizeAC;
for (i=1;i<numberOfIndex;i++){ // second to last face...
        switch(nType[i]){
                case 0:   // mode 0
                        ACDecoder(mPos) nPosition;
                        ACDecoder(mFD) faceDirection;
                        ACDecoder(mModel, mhasnext) nDifIndex;
                        if (nDifIndex!= 0)        ACDecoder(mSign) nSign;
                        ACDecoder(mRotaion) nRotation;
                     break;
                     case 1:  //mode 1
                        for (j=0;j<3;j++){
                                ACDecoder(mModel, mhasnext) nDifIndex;
                                if (nDifIndex!= 0)       ACDecoder(mSign) nSign;
                        }
                     break;
                     case 2:  //mode 2
                        ACDecoder(mPos) nPosition;
                        for(j = 0; j< 2; j++){
                                ACDecoder(mModel, mhasnext) nDifIndex;
                                if (nDifIndex!= 0)        ACDecoder(mSign) nSign;
                        }
                        ACDecoder(mRotaion) nRotation;
                     break;
                     case 3:  //mode 3
                        for (j=0;j<3;j++){
                                ACDecoder(mModel, mhasnext) nDifIndex;
                                if (nDifIndex!= 0)        ACDecoder(mSign) nSign;
```