INTERNATIONAL STANDARD

ISO/IEC 14496-1

Second edition
2001-10-01

AMENDMENT 4
2003-10-15

Information technology — Coding of audio-visual objects —

Part 1: **Systems**

AMENDMENT 4: SL extensions and AFX streams

Technologies de l'information — Codage des objets audiovisuels —

Partie 1: Systèmes

AMENDEMENT 4: Extensions SL et cours AFX

Reference number
ISO/IEC 14496-1:2001/Amd.4:2003(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

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

Published 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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

Amendment 4 to 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 4: SL extensions and AFX streams

In subclause 8.2.1, replace Table 1 - List of Class Tags for Descriptors, by the following:

Table 1 - List of Class Tags for Descriptors

Tag value	Tag name
0x00	Forbidden
0x01	ObjectDescrTag
0x02	InitialObjectDescrTag
0x03	ES_DescrTag
0x04	DecoderConfigDescrTag
0x05	DecSpecificInfoTag
0x06	SLConfigDescrTag
0x07	ContentIdentDescrTag
0x08	SupplContentIdentDescrTag
0x09	IPI_DescrPointerTag
0x0A	IPMP_DescrPointerTag
0x0B	IPMP_DescrTag
0x0C	QoS_DescrTag
0x0D	RegistrationDescrTag
0x0E	ES_ID_IncTag
0x0F	ES_ID_RefTag
0x10	MP4_IOD_Tag
0x11	MP4_OD_Tag
0x12	IPL_DescrPointerRefTag
0x13	ExtensionProfileLevelDescrTag
0x14	profileLevelIndicationIndexDescrTag
0x15-0x3F	Reserved for ISO use
0x40	ContentClassificationDescrTag
0x41	KeyWordDescrTag
0x42	RatingDescrTag
0x43	LanguageDescrTag
0x44	ShortTextualDescrTag
0x45	ExpandedTextualDescrTag
0x46	ContentCreatorNameDescrTag
0x47	ContentCreationDateDescrTag
0x48	OCICreatorNameDescrTag
0x49	OCICreationDateDescrTag

0x4A	SmpteCameraPositionDescrTag
0x4B	SegmentDescrTag
0x4C	MediaTimeDescrTag
0x4D-0x5F	Reserved for ISO use (OCI extensions)
0x60-0x61	Reserved for ISO use
0x62	FLEXmuxTimingDescrTag
0x63	FLEXmuxCodeTableDescrTag
0x64	ExtSLConfigDescrTag
0x65	FLEXmuxBufferSizeDescrTag
0x66	FLEXmuxIdentDescrTag
0x67	DependencyPointerTag
0x68	DependencyMarkerTag
0x69	FLEXmuxChannelDescrTag
0x6A-0xBF	Reserved for ISO use
0xC0-0xFE	User private
0xFF	Forbidden

In 8.6.4.2, replace Table 3 by:

Table 3 - ODProfileLevelIndication Values

Value	Profile	Level
0x00	Forbidden	-
0x01	Reserved for ISO use (no SL extension)	-
0x02-0x7F	Reserved for ISO use (SL extension)	-
0x03-0x7F	Reserved for ISO use	
0x80-0xFD	user private	-
0xFE	No OD profile specified	-
0xFF	No OD capability required	-

Note - 0x01 also indicates that the SL extension mechanism is not present ."

Replace 8.6.5.1 ESDescriptor syntax by:

```
class ES_Descriptor extends BaseDescriptor : bit(8) tag=ES_DescrTag {
   bit(16) ES_ID;
   bit(1) streamDependenceFlag;
   bit(1) URL_Flag;
   bit(1) OCRstreamFlag;
   bit(5) streamPriority;
   if (streamDependenceFlag)
      bit(16) dependsOn_ES_ID;
   if (URL_Flag) {
      bit(8) URLlength;
      bit(8) URLstring[URLlength];
   }
   if (OCRstreamFlag)
      bit(16) OCR_ES_Id;
   DecoderConfigDescriptor decConfigDescr;
   if (ODProfileLevelIndication==0x01)       //no SL extension.
   {
      SLConfigDescriptor slConfigDescr;
   }
   else                                      // SL extension is possible.
   {
      SLConfigDescriptor slConfigDescr;
   }
   IPI_DescrPointer ipiPtr[0 .. 1];
   IP_IdentificationDataSet ipIDS[0 .. 255];
   IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];
   LanguageDescriptor langDescr[0 .. 255];
   QoS_Descriptor qosDescr[0 .. 1];
   RegistrationDescriptor regDescr[0 .. 1];
   ExtensionDescriptor extDescr[0 .. 255];
}
```

In 8.6.5.2, replace:

`slConfigDescr - is an SLConfigDescriptor as specified in 8.6.8.`

by

`slConfigDescr` – is an SLConfigDescriptor as specified in 8.6.8. If `ODProfileLevelIndication` is different from 0x01, it may be an extension of SLConfigDescriptor (i.e. and extended class) as defined in 8.6.8.

In subclause 8.6.6.2: Semantics (of 8.6.6 DecoderConfigDescriptor), replace Table 8 - objectTypeIndication Values, with the following:

Table 8 - objectTypeIndication Values

Value	`ObjectTypeIndication` Description
0x00	Forbidden
0x01	Systems ISO/IEC 14496-1 [a]
0x02	Systems ISO/IEC 14496-1 [b]
0x03	Interaction Stream
0x04	Systems ISO/IEC 14496-1 Extended BIFS Configuration [c]
0x05	Systems ISO/IEC 14496-1 AFX [d]
0x06-0x1F	reserved for ISO use
0x20	Visual ISO/IEC 14496-2 [e]
0x21-0x3F	reserved for ISO use
0x40	Audio ISO/IEC 14496-3 [f]
0x41-0x5F	reserved for ISO use
0x60	Visual ISO/IEC 13818-2 Simple Profile
0x61	Visual ISO/IEC 13818-2 Main Profile
0x62	Visual ISO/IEC 13818-2 SNR Profile
0x63	Visual ISO/IEC 13818-2 Spatial Profile
0x64	Visual ISO/IEC 13818-2 High Profile
0x65	Visual ISO/IEC 13818-2 422 Profile
0x66	Audio ISO/IEC 13818-7 Main Profile
0x67	Audio ISO/IEC 13818-7 LowComplexity Profile
0x68	Audio ISO/IEC 13818-7 Scaleable Sampling Rate Profile
0x69	Audio ISO/IEC 13818-3
0x6A	Visual ISO/IEC 11172-2
0x6B	Audio ISO/IEC 11172-3
0x6C	Visual ISO/IEC 10918-1
0x6D - 0xBF	reserved for ISO use
0xC0 - 0xFE	user private
0xFF	no object type specified [g]

[a] This type is used for all 14496-1 streams unless specifically indicated to the contrary. Scene Description scenes, which are identified with StreamType=0x03 (see Table 7), using this object type value shall use the BIFSConfig specified in subclause 9.3.5.2.2 of this specification.

[b] This object type shall be used, with StreamType=0x03 (see Table 7), for Scene Description streams that use the BIFSv2Config specified in subclause 9.3.5.3.2 of this specification. Its use with other StreamTypes is reserved.

[c] This object type shall be used, with StreamType=0x03 (see Table 7), for Scene Description streams that use the BIFSConfigEx specified in subclause 8.7.6.1.2 of this specification. Its use with other StreamTypes is reserved.

[d] This object type shall be used, with StreamType=0x03 (see Table 7), for Scene Description streams that use the AFXConfig specified in subclause 8.7.6.1.2 of this specification. Its use with other StreamTypes is reserved.

[e] Includes associated Amendment(s) and Corrigendum(a). The actual object types are defined in ISO/IEC 14496-2 and are conveyed in the DecoderSpecificInfo as specified in ISO/IEC 14496-2, Annex K.

[f] Includes associated Amendment(s) and Corrigendum(a). The actual object types are defined in ISO/IEC 14496-3 and are conveyed in the DecoderSpecificInfo as specified in ISO/IEC 14496-3 subpart 1 subclause 6.2.1.

[g] Streams with this value with a StreamType indicating a systems stream (values 1,2,3, 6, 7, 8, 9) shall be treated as if the ObjectTypeIndication had been set to 0x01.

Append to 8.6.7.2:

For values of `DecoderConfigDescriptor.objectTypeIndication` that refers to extended BIFS configuration (0x04), the decoder specific information is:

```
class BIFSConfigEx extends DecoderSpecificInfo : bit (8) tag = DecSpecificInfoTag
{
   ExtendedBIFSConfig extBIFSConfig;
}

abstract aligned(8) expandable (..) class ExtendedBIFSConfig : bit (8) tag = 0x01..0xFF {
  //empty. To be filled by classes extending this class.
}
```

The class `BIFSConfigEx` contains an `ExtendedBIFSConfig`. `ExtendedBIFSConfig` is the base class for new classes meant to hold decoder specific information. With this approach, new BIFS streams will have streamType 3 and objectTypeIndication 3, but will use decoder configuration depending on the tag of the `ExtendedBIFSConfig`.

For values of `DecoderConfigDescriptor.objectTypeIndication` that refers to AFX streams (0x05), the decoder specific information is:

```
class AFXConfig extends DecoderSpecificInfo : bit(8) tag=DecSpecificInfoTag {
   AFXExtDescriptor afxext;
}

abstract class AFXExtDescriptor extends BaseDescriptor : bit(8) tag = 0..100
{
}
```

`AFXExtDescriptor` is an abstract class used as a placeholder for an optional DecoderSpecificInfo defined in this specification:

Table AMD4-1 — DecoderSpecificInfo for AFX streams

AFX stream	DecoderSpecificInfo
MeshGrid	See subclause 5.2.2.2 of ISO/IEC 14496-16.
WaveletSubdivisionSurface	See subclause 5.1.1.1 of ISO/IEC 14496-16.
Other AFX streams	None

and the tag refers to a specific node compression scheme as defined in Table AMD4-2.

Table AMD4-2 — AFX object code

AFX object code	object
0x00	3D Mesh Compression
0x01	WaveletSubdivisionSurface
0x02	MeshGrid
0x03	CoordinateInterpolator
0x04	OrientationInterpolator
0x05	PositionInterpolator
0x06	OctreeImage

In 10.2.3.1, append at the end of the subclause:

```
class ExtendedSLConfigDescriptor extends SLConfigDescriptor : bit(8)
tag=ExtSLConfigDescrTag {
   SLExtensionDescriptor slextDescr[1..255];
}
```

Append to 10.2.3.2:

`slextDescr` – is an array of ExtensionDescriptors defined for `ExtendedSLConfigDescriptor` as specified in 10.2.3.3.

Insert new subclause 10.2.3.3 SLExtensionDescriptor syntax:

```
abstract class SLExtensionDescriptor : bit(8) tag=0 {
}

class DependencyPointer extends SLExtensionDescriptor: bit(8) tag= DependencyPointerTag {
   bit(6) reserved;
   bit(1) mode;
   bit(1) hasESID;
   bit(8) dependencyLength;
   if (hasESID)
   {
      bit(16) ESID;
   }
}
class MarkerDescriptor extends SLExtensionDescriptor: bit(8) tag=DependencyMarkerTag {
   int(8) markerLength;
}
```

Insert new subclause 10.2.3.4 SLExtensionDescriptor semantics:

SLExtensionDescriptor is an abstract class specified so as to be the base class of sl extensions.

10.2.3.4.1 DependencyPointer semantics

DependencyPointer extends SLExtensionDescriptor and specifies that access units from this stream depend on another stream.

If mode equals 0, the latter stream can be identified through the ESID field or if no ESID is present, using the `dependsOn_ES_ID` ESID, and access units from this stream will point to the decodingTimeStamps of that stream.

If mode equals 1, access units from this stream will convey identifiers, for which the system (e.g. IPMP tools) are responsible to know whether dependent resources (e.g. keys) are available.

In both cases, the length of this pointer or identifier is `dependencyLength`.

If mode is 0 then `dependencyLength` shall be the length of the `decodingTimeStamp`.

10.2.3.4.2 MarkerDescriptor semantics

MarkerDescriptor extends SLExtensionDescriptor and allows to tag access units so as to be able to refer to them independently from their decodingTimeStamp.

`markerLength` – is the length for identifiers tagging access units.

Replace 10.2.4.1 with:

```
aligned(8) class SL_PacketHeader (SLConfigDescriptor SL) {
   if (SL.useAccessUnitStartFlag)
      bit(1) accessUnitStartFlag;
   if (SL.useAccessUnitEndFlag)
      bit(1) accessUnitEndFlag;
   if (SL.OCRLength>0)
      bit(1) OCRflag;
   if (SL.useIdleFlag)
      bit(1) idleFlag;
   if (SL.usePaddingFlag)
      bit(1) paddingFlag;
   if (paddingFlag)
      bit(3) paddingBits;

   if (!idleFlag && (!paddingFlag || paddingBits!=0)) {
      if (SL.packetSeqNumLength>0)
         bit(SL.packetSeqNumLength) packetSequenceNumber;
      if (SL.degradationPriorityLength>0)
         bit(1) DegPrioflag;
      if (DegPrioflag)
         bit(SL.degradationPriorityLength) degradationPriority;
      if (OCRflag)
         bit(SL.OCRLength) objectClockReference;

      if (accessUnitStartFlag) {
         if (SL.useRandomAccessPointFlag)
            bit(1) randomAccessPointFlag;
         if (SL.AU_seqNumLength >0)
            bit(SL.AU_seqNumLength) AU_sequenceNumber;
         if (SL.useTimeStampsFlag) {
            bit(1) decodingTimeStampFlag;
            bit(1) compositionTimeStampFlag;
         }
         if (SL.instantBitrateLength>0)
            bit(1) instantBitrateFlag;
         if (decodingTimeStampFlag)
            bit(SL.timeStampLength) decodingTimeStamp;
         if (compositionTimeStampFlag)
            bit(SL.timeStampLength) compositionTimeStamp;
         if (SL.AU_Length > 0)
            bit(SL.AU_Length)     accessUnitLength;
         if (instantBitrateFlag)
            bit(SL.instantBitrateLength)      instantBitrate;
      }
   if (SL.tag == ExtSLConfigDescrTag)
   {
      for (int i=0; i<SL.slextDescr.length;i++)
      {
         switch(SL.slextDescr[i].tag)

            case DependencyPointerTag:
               Marker value(SL.slextDescr[i].dependencyLength);
               break;
            case DependencyMarkerTag:
               Marker value(SL.slextDescr[i]. markerLength);
               break;
            default:
               break;
         }
      }
   }
   }
}

aligned expandable class Marker(int length) {
   bit(length) value;
}
```

Append to 10.2.4.2:

If the `SLConfigDescriptor` is an `ExtendedSLConfigDescriptor` (i.e. its tag is `ExtSLConfigDescrTag`), then descriptors associated with the array of `SLExtensionDescriptors` are appended to the end of the SLPacket Header.

Note – Since those descriptors conveying the extended SL information; carry their size, they can be skipped by a decoder.

DependencyPointerDescriptor and MarkerDescriptor define their associated descriptors as follows:

For DependencyPointerDescriptor a Marker of length dependencyLength will be encoded. It shall resolve either to an identifier or to a decodingTimeStamp as specified in 10.2.3.4.1.

For MarkerDescriptor a marker of length markerLength is encoded.

Replace subclause 12.2.4 with the following:

12.2.4 FlexMux packet specification

12.2.4.1 Syntax

```
class FlexMuxPacket (MuxCodeTableEntry mct[],
                     FlexMuxTimingDescriptor FM,
                     FlexMuxIDDescriptor mde) {
   unsigned int(8) index;
   if (mde == NULL | mde.Muxtype == 0) {
      bit(8) length;
   } else if (mde.Muxtype == 1) {
      length = 0;
      bit(1) nextByte;
      bit(7) length;
      while(nextByte) {
         bit(1) nextByte;
         bit(7) sizeByte;
         length = length<<7 | sizeByte;
      }
   }
      if (index<238) {
         if (length!=0) {
            SL_Packet sPayload;
            } else {
               bit(5) FMC_version_number;
               const bit(3) reserved=0b111;
            }
      } else if (index == 238) {
         bit(FM.FCR_Length) fmxClockReference;
         bit(FM.fmxRateLength) fmxRate;
            for (i=0; i<(length-FM.FCR_Length-FM.fmxRateLength); i++) {
               FlexMux_descriptor()
            }
         } else if (index == 239) {
         bit(8) stuffing[length];
      } else {
         bit(4) version;
         const bit(4) reserved=0b1111;
         multiple_SL_Packet mPayload(mct[index-240]);
      }
}
```

12.2.4.2 Semantics

`length` – the length of the FlexMux packet `payload` in bytes. This is equal to the length of the single encapsulated SL packet in Simple Mode and to the total length of the multiple encapsulated SL packets in MuxCode Mode. If the FlexMuxIDDescriptor is not used, or if it is used and if the Muxtype is designing the first FlexMux tool, the length field is on one byte. If the FlexMuxIDDescriptor is used and if the Muxtype is designing the second FlexMux tool, the length calculation relies on the combination of the `nextByte` and `sizeByte` fields that can be spread over several bytes. In Simple Mode, when this length is equal to zero, the FlexMux packet carries one byte that contains the `FMC_version_number` field. In Simple Mode, FlexMux packets with a length equal to zero (each carrying a `FMC_version_number`) can be duplicated.

`FMC_version_number` – This 5 bit field indicates the current version of the FLEXmuxChannelDescriptor that is applicable. `FMC_version_number` is used for error resilience purposes. If this version number does not match the version of the referenced FLEXmuxChannelDescriptor that has most recently been received, the following FlexMux packets belonging to the same FlexMux Channel cannot be parsed. The implementation is free to either wait until the required version of FLEXmuxChannelDescriptor becomes available or to discard the following FlexMux packets belonging to the same FlexMux Channel. In Simple Mode, the value given to the `FMC_version_number` field is identical in subsequent duplicated FlexMux packets with a length equal to zero.

Replace subclause 12.2.6 with the following:

12.2.6 Configuration and usage of FlexMux clock references

12.2.6.1 Syntax

```
aligned(8) class FlexMuxTimingDescriptor {
   bit(16) FCR_ES_ID;
   bit(32) FCRResolution;
   bit(8) FCRLength;
   bit(8) FmxRateLength;
}
```

12.2.6.2 Semantics

The sequence of `fmxClockReference` time stamps in a FlexMux stream constitutes a clock reference stream, albeit with a different syntax as specified in clause 10. Elementary streams shall be associated to the time base established by this clock reference by referencing the `FCR_ES_ID` as their `OCR_ES_ID` in the `SLConfigDescriptor`. The transport of the `FlexMuxTimingDescriptor` shall be defined during the design of the transport protocol stack that makes use of the FlexMux tool.

Insert subclause 12.2.10:

12.2.10 FlexMuxID Descriptor

12.2.10.1 Syntax

```
aligned(8) class FlexMuxIDDescriptor {
   bit(8) MuxID;
   bit(4) Muxtype;
    bit(4) Muxmanagement;
}
```

12.2.10.2 Semantics

`MuxID` – the ID of the FlexMux stream.

`Muxtype` – the type of the Multiplexing tool used to generate the FlexMux stream.

Indicated type value shall comply with the following AMD4-3 — Multiplexing type table.

`Muxmanagement` – the mode of management used by the Multiplexing tool, to generate the Flexmux stream. Indicated mode value shall comply with the AMD4-4 — Multiplexing management mode table.

Table AMD4-3 — Multiplexing type table

Type	Multiplexing tool
0	FlexMux tool
1	FlexMux_2 tool
2-7	ISO/IEC 14496-1 Reserved
8-15	User Private

Table AMD4-4 — Multiplexing management mode table

Type	management mode
0	Static
1	Dynamic
2-7	ISO/IEC 14496-1 Reserved
8-15	User Private

Add subclause 12.3:

12.3 FLEXmux Descriptors

Directly derived from the FlexMux descriptor classes, hereafter are defined the FLEXMux descriptors pointed to by the “List of Class Tags for Descriptors” table.

12.3.1 FLEXmuxChannel Descriptor

12.3.1.1 Syntax

```
class FLEXmuxChannelDescriptor extends BaseDescriptor
          : bit(8) tag= FLEXmuxChannelDescrTag {
     bit(5) version_number;
     bit(1) current_next_indicator;
     const bit(2) reserved=0b11;
     for (i=0; i<( sizeOfInstance-2); i += 3) {
     bit(16) ES_ID;
     bit(8)  FlexMuxChannel;
     }
}
```