INTERNATIONAL STANDARD

**ISO/IEC 16500-6**

First edition
1999-12-15

# Information technology — Generic digital audio-visual systems —

## Part 6: Information representation

*Technologies de l'information — Systèmes audiovisuels numériques génériques —*

*Partie 6: Représentation des informations*



Reference number
ISO/IEC 16500-6:1999(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 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

## Contents Page

**Foreword** ..... vii

**Introduction** ..... viii

**1. Scope** ..... 1

**2. Normative references** ..... 2

**3. Definitions** ..... 4

**4. Acronyms and abbreviations** ..... 6

**5. Conventions** ..... 8

**6. Monomedia Components** ..... 9

**6.1. Character Information** ..... 9

**6.2. Text Information** ..... 10
6.2.1 Decoding requirements on HTML support ..... 10
6.2.2 HTML encoding ..... 11
6.2.3 HTML mapping to MHEG elements ..... 11
6.2.4 Informative list of supported HTML Tags ..... 11

**6.3. Outline Font Format** ..... 13
6.3.1 Requirements ..... 13
6.3.2 Font Format Specification ..... 14
6.3.3 Font rendering ..... 14

**6.4. Language Information** ..... 16

**6.5. Service Information** ..... 16

**6.6. Telephone Numbers** ..... 16

**6.7. Compressed Audio** ..... 17
6.7.1 Compressed audio coding using MPEG-1 Audio ..... 17
6.7.2 Compressed audio coding using ATSC A/52 Audio ..... 17

**6.8. Scaleable Audio** ..... 18

**6.9. Linear Audio** ..... 18

**6.10. Compressed Video** ..... 19
6.10.1 Coding constraints for video with a resolution up to ITU-R 601 ..... 19
6.10.2 Coding constraints for video with a resolution beyond ITU-R 601 ..... 22
6.10.3 Decoding tool requirements ..... 24

**6.11. Still Pictures** ..... 25
6.11.1 Normal resolution still pictures ..... 25
6.11.2 Higher resolution still pictures ..... 25

**6.12. Compressed Graphics** ..... 26
6.12.1 Requirements ..... 26
6.12.2 Format for Compressed Graphics ..... 26

**6.13. Compressed Character Data** ..... 28



**6.14. Network Graphics ..... 28**

**7. Monomedia Streams ..... 29**

**7.1. Types of Monomedia Components ..... 29**

**7.2. Real-time and Stored Monomedia Streams ..... 30**

**7.3. Carriage of Monomedia Streams in PES Packets ..... 31**
7.3.1 Packetization of MPEG- and ATSC-defined Components ..... 31
7.3.2 Packetization of DVB-defined Components ..... 31
7.3.3 Packetization of DAVIC-defined Components ..... 31

**7.4. Packetization of Compressed Character Data ..... 31**

**8. Transport of Monomedia Streams and Components ..... 32**

**8.1. Transport of Real Time Streams ..... 32**

**8.2. Transport of Stored Streams ..... 32**

**8.3. Transport of Stand-alone Monomedia Components ..... 32**

**9. Application Format ..... 33**

**9.1. Application Interchange Format ..... 33**

**9.2. MHEG-5 Profile for the DAVIC Application Domain ..... 34**
9.2.1 Object Interchange Format ..... 34
9.2.2 Set of Classes ..... 34
9.2.3 Set of Features ..... 34
9.2.4 Content Data Encoding ..... 34
9.2.5 Attribute Encoding ..... 35
9.2.6 UserInput Registers ..... 36
9.2.7 Constraints on the Use of Variables ..... 37
9.2.8 Semantic Constraints on the MHEG-5 Applications ..... 37
9.2.9 EngineEvent ..... 37
9.2.10 GetEngineSupport ..... 37
9.2.11 TransitionEffect Parameter of the TransitionTo Elementary Action ..... 38
9.2.12 MHEG-5 Resident Programs ..... 40
9.2.13 Protocol Mapping and External Interaction ..... 45

**9.3. Mapping of MHEG-5 Elements to DSM-CC U-U ..... 47**
9.3.1 Stream Events and Normal Play Time Mapping ..... 47
9.3.2 Namespace Mapping ..... 47
9.3.3 MHEG-5 Object References ..... 48
9.3.4 MHEG-5 Content References ..... 49
9.3.5 Mapping of MHEG-5 ComponentTag to DSM-CC and DVB-SI ..... 49
9.3.6 Java class File Names ..... 49

**9.4. Core set of Java APIs ..... 51**
9.4.1 java.lang ..... 51
9.4.2 java.util ..... 51
9.4.3 java.io ..... 51
9.4.4 iso.mheg5 ..... 51
9.4.5 The DSM-CC User to User API ..... 52
9.4.6 The Service Information (SI) API ..... 52
9.4.7 The MPEG-2 Section Filter API ..... 52
9.4.8 The Resource Notification API ..... 52
9.4.9 The MPEG Component API ..... 52



**9.5. URL Format for Access to Broadcast Services ..... 53**
9.5.1 Introduction ..... 53
9.5.2 Numerical format ..... 53

**9.6. Run-time execution environment ..... 54**
9.6.1 Application execution ..... 54
9.6.2 User Input Events ..... 54
9.6.3 IDL definition for RTE run remote call ..... 54
9.6.4 Mapping of High-Level API Actions on DSM-CC Primitives ..... 55

**10. DAVIC Reference Model for Contents Decoding ..... 56**

**10.1. Scope ..... 56**

**10.2. Reference Decoder Model ..... 56**

**10.3. DAVIC Application Resource Descriptor ..... 59**

**10.4. Minimum ISO/IEC 16500 STU requirements ..... 61**

**10.5. Support for Graphics in STUs ..... 61**

**10.6. Persistent Memory ..... 61**

**11. Content Packaging and Metadata ..... 62**

**11.1. Content package structure ..... 62**
11.1.1 Content Item Elements ..... 62
11.1.2 Content Item ..... 62
11.1.3 Content package ..... 63

**11.2. Content Metadata ..... 64**
11.2.1 Types ..... 64
11.2.2 Semantics of the Loading Commands ..... 65
11.2.3 Metadata Mapping ..... 66

**11.3. Content Packaging Format ..... 72**
11.3.1 Bento ..... 72
11.3.2 Bento definition of DAVIC Objects ..... 72

**11.4. Content Loading Toolset ..... 79**
11.4.1 Scope ..... 79
11.4.2 Toolset requirements ..... 79
11.4.3 Toolset features ..... 80
11.4.4 Toolset Specifications ..... 80
11.4.5 Content Loading by CMSL ..... 82

**Annex A** (normative) **Coding of Outline Fonts ..... 84**

**Annex B** (normative) **Coding of Linear Audio ..... 112**

**Annex C** (normative) **Default CLUT for single bitmaps with compressed graphics ..... 116**

**Annex D** (normative) **PES Packetization of DAVIC defined Monomedia Components ..... 120**

**Annex E** (normative) **MPEG-2 Section Filter API ..... 122**

**Annex F** (normative) **Resource Notification API ..... 152**

**Annex G** (normative) **MPEG Component API ..... 156**



**Annex H** (informative) **Carriage of Private Data** ..... **166**

**Annex I** (informative) **Video Input Formats** ..... **168**

**Annex J** (not used) ..... **170**

**Annex K** (informative) **STU Video Display Capabilities** ..... **172**

**Annex L** (informative) **Coding and Carriage of A/52 Audio in ATSC Systems** ..... **177**

**Annex M** (informative) **Transition Effects for Still Picture Compositions** ..... **179**

**Annex N** (informative) **Example of an OSI NSAP Address Format** ..... **185**

**Annex O** (informative) **Content Metadata Specification Language** ..... **187**

**Annex P** (informative) **Example of Simple Movie Content Item** ..... **193**

**Bibliography** ..... **195**



# 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 part of ISO/IEC 16500 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 16500-6 was prepared by DAVIC (Digital Audio-Visual Council) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

ISO/IEC 16500 consists of the following parts, under the general title *Information technology — Generic digital audio-visual systems*:

— *Part 1: System reference models and scenarios*

— *Part 2: System dynamics, scenarios and protocol requirements*

— *Part 3: Contours: Technology domain*

— *Part 4: Lower-layer protocols and physical interfaces*

— *Part 5: High and mid-layer protocols*

— *Part 6: Information representation*

— *Part 7: Basic security tools*

— *Part 8: Management architecture and protocols*

— *Part 9: Usage information protocols*

Annexes A to G form a normative part of this part of ISO/IEC 16500. Annexes H to P are for information only.

# Introduction

ISO/IEC 16500 defines the minimum tools and dynamic behavior required by digital audio-visual systems for end-to-end interoperability across countries, applications and services. To achieve this interoperability, it defines the technologies and information flows to be used within and between the major components of generic digital audio-visual systems. Interoperability between these components and between individual sub-systems is assured through specification of tools and specification of dynamic systems behavior at defined reference points. A reference point can comprise one or more logical (non-physical) information-transfer interfaces, and one or more physical signal-transfer interfaces. A logical interface is defined by a set of information flows and associated protocol stacks. A physical interface is an external interface and is fully defined by its physical and electrical characteristics. Accessible reference points are used to determine and demonstrate compliance of a digital audio-visual subsystem with this international standard.

A summary of each part follows.

ISO/IEC 16500-1 (DAVIC 1.3.1a Part 2) defines the normative digital audio-visual systems technical framework. It provides a vocabulary and a Systems Reference Model, which identifies specific functional blocks and information flows, interfaces and reference points.

ISO/IEC 16500-2 (DAVIC 1.3.1a Part 12) defines system dynamic behavior and physical scenarios. It details the locations of the control functional entities along with the normative protocols needed to support the systems behavior. It is structured as a set of protocol walk-throughs, or *"Application Notes"*, that rehearse both the steady state and dynamic operation of the system at relevant reference points using specified protocols. Detailed dynamics are given for the following scenarios: video on demand, switched video broadcast, interactive broadcast, and internet access.

ISO/IEC 16500-3 (DAVIC 1.3.1a Part 14) provides the normative definition of DAVIC Technology Contours. These are strict sets of Applications, Functionalities and Technologies which allow compliance and conformance criteria to be easily specified and assessed. This part of ISO/IEC 16500 contains the full details of two contours. These are the Enhanced Digital Broadcast (EDB) and Interactive Digital Broadcast (IDB). ISO/IEC 16500-3 specifies required technologies and is a mandatory compliance document for contour implementations.

ISO/IEC 16500-4 (DAVIC 1.3.1a Part 8) defines the toolbox of technologies used for lower layer protocols and physical interfaces. The tools specified are those required to digitize signals and information in the Core Network and in the Access Network. Each tool is applicable at one or more of the reference points specified within the Delivery System. In addition a detailed specification is provided of the physical interfaces between the Network Interface Unit and the Set Top Unit and of the physical interfaces used to connect Set Top Boxes to various peripheral devices (digital video recorder, PC, printer). The physical Delivery System mechanisms included are copper pairs, coaxial cable, fiber, HFC, MMDS, LMDS, satellite and terrestrial broadcasting.

ISO/IEC 16500-5 (DAVIC 1.3.1a Part 7) defines the technologies used for high and mid-layer protocols for ISO/IEC 16500 digital audio-visual systems. In particular, this part defines the specific protocol stacks and requirements on protocols at specific interfaces for the content, control and management information flows.

ISO/IEC 16500-6 (DAVIC 1.3.1a Part 9) defines what the user will eventually see and hear and with what quality. It specifies the way in which monomedia and multimedia information types are coded and exchanged. This includes the definition of a virtual machine and a set of APIs to support interoperable exchange of program code. Interoperability of applications is achieved, without specifying the internal design of a set top unit, by a normative Reference Decoder Model which defines specific memory and behavior constraints for content decoding. Separate profiles are defined for different sets of multimedia components.

ISO/IEC 16500-7 (DAVIC 1.3.1a Part 10) defines the interfaces and the security tools required for an ISO/IEC 16500 system implementing security profiles. These tools include security protocols which operate across one or both of the defined conditional access interfaces CA0 and CA1. The interface CA0 is to all security and conditional access functions, including the high speed descrambling functions. The interface CA1 is to a tamper resistant device used for low speed cryptographic processing. This cryptographic processing function is implemented in a smart card.

ISO/IEC 16500-8 (DAVIC 1.3.1a Part 6) specifies the information model used for managing ISO/IEC 16500 systems. In particular, this part defines the managed object classes and their associated characteristics for managing the access network and service-related data in the Delivery System. Where these definitions are taken from existing standards, full reference to the required standards is provided. Otherwise a full description is integrated in the text of this part. Usage-related information model is defined in ISO/IEC 16500-9.



ISO/IEC 16500-9 (DAVIC 1.3.1a Part 11) specifies the interface requirements and defines the formats for the collection of usage data used for billing, and other business-related operations such as customer profile maintenance. It also specifies the protocols for the transfer of Usage Information into and out of the ISO/IEC 16500 digital audio-visual system. In summary, flows of audio, video and audio-visual works are monitored at defined usage data collection elements (e.g., servers, elements of the Delivery System, set-top boxes). Information concerning these flows is then collected, processed and passed to external systems such as billing or a rights administration society via a standardised usage data transfer interface.

### Additional Information

ISO/IEC TR 16501 is an accompanying Technical Report. Further architectural and conformance information is provided in other non-normative parts of DAVIC 1.3.1a (1999). A summary of these documents is included here for information.

ISO/IEC TR 16501 (DAVIC 1.3.1a Part 1) provides a detailed listing of the functionalities required by users and providers of digital audio-visual applications and systems. It introduces the concept of a contour and defines the IDB (Interactive Digital Broadcast) and EDB (Enhanced Digital Broadcast) functionality requirements which are used to define the normative contour technology toolsets provided in ISO/IEC 16500-3.

DAVIC 1.3.1a Parts 3, 4 and 5 are DAVIC technical reports. They provide additional architectural and other information for the server, the delivery-system, and the Service Consumer systems respectively. Part 3 defines how to load an application, once created, onto a server and gives information and guidance on the protocols transmitted from the set-top user to the server, and those used to control the set-up and execution of a selected application. Part 4 provides an overview of Delivery Systems and describes instances of specific DAVIC networked service architectures. These include physical and wireless networks. Non-networked delivery (e.g., local storage physical media like discs, tapes and CD-ROMs) are not specified. Part 5 provides a Service Consumer systems architecture and a description of the DAVIC Set Top reference points defined elsewhere in the normative parts of the specification.

DAVIC 1.3.1a Part 13 is a DAVIC technical report, which provides guidelines on how to validate the systems, technology tools and protocols through conformance and / or interoperability testing.

# Information technology — Generic digital audio-visual systems — Part 6: Information representation

## 1. Scope

This part of ISO/IEC 16500 takes a practical approach to the specification of Information Representation. Just the information types that cannot be dispensed with in producing the set of DAVIC applications (viz. broadcast, movies on demand, home shopping, etc.) are specified. The approach taken in this part of ISO/IEC 16500 starts by defining the various monomedia information types. They include character, text, fonts, service information, audio, video, and graphics. Consistent with DAVIC principles, one tool is selected for the encoding of each information type. Multimedia components comprise one or more monomedia components. This part of ISO/IEC 16500 defines the way in which multimedia information is coded and exchanged. This includes the definition of a virtual machine and a set of APIs to support interoperable exchange of program code. Finally, this part of ISO/IEC 16500 defines a Reference Decoder Model for contents decoding which provides constraints on content. The major problem addressed by the model is to ensure interoperability of applications by specifying memory and behaviour constraints for contents decoding by a hypothetical STU, without specifying the internal design of an STU. An application built according to the reference decoder model will be an "ISO/IEC 16500 conforming application" and will successfully execute on a STU that is compliant to ISO/IEC 16500.

For each monomedia and multimedia component the coding format is specified, as well as applicable constraints for coding of the components. Three types of monomedia components are distinguished. Monomedia components which are included within other monomedia components, such as characters within text, are of type implied. Non-implied monomedia components that do not require synchronization with a time base at play back, are of type stand-alone. Finally, non-implied monomedia components of which the presentation may require synchronization with a time base are of type stream. This part of ISO/IEC 16500 defines which type each DAVIC defined monomedia component may take, and specifies that the coded representation of monomedia components of type stream are packetized in PES packets (for definition of PES packets refer to ISO/IEC 13818-1). PES packets permit (1) to include time stamps to support mutual synchronisation of multiple monomedia components in reference to a common time base and (2) to define timing and buffer behaviour in a common reference model for contents decoding. While there are various ways to deliver the monomedia and multimedia components to the STU, This part of ISO/IEC 16500 defines how the components are carried in an MPEG-2 Transport Stream.

DAVIC specifies a number of different profiles. In a specific profile there may be support of a subset of the monomedia components. Each STU that complies to a specific profile of DAVIC shall be capable of decoding and presenting each monomedia and multimedia component permitted within that profile.

This part of ISO/IEC 16500 also specifies methods for packaging of contents and metadata. The way in which content is packaged for delivery is independent of the way in which content data is delivered to the SPS (it may be delivered to a Service Provider either on physical media or over a transmission system). All programming content is represented in the DAVIC system as multimedia components. Multimedia components comprise one or more monomedia components coupled with the logical relationships between the monomedia components. The multimedia components will be created by content providers for input to the servers.



# 2. Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 16500. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 16500 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau (TSB) maintains a list of currently valid ITU-T Recommendations.

## 2.1. ISO, ISO/IEC and ITU Normative References

1. ISO 639, *Codes for the representation of names of languages*.
2. ISO 3166, *Codes for the representation of names of countries.*
3. ISO/IEC 8859-1:1987, *Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1*.
4. ISO/IEC 10646-1, *Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane"* (also known as Unicode).
5. ISO/IEC 11172–2:1993, *Information technology—Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s—Part 2: Video* (Note: known as MPEG-1 Video).
6. ISO/IEC 11172–3:1993, *Information technology—Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s—Part 3: Audio* (Note: known as MPEG-1 Audio).
7. ISO/IEC 13522–5:1997, *Information technology—Coding of multimedia and hypermedia information—Part 5: Support for base-level interactive applications* (Note: known as MHEG-5).
8. ISO/IEC 13522-6, *Information technology - Coding of multimedia and hypermedia information - Part 6: Support for enhanced interactive applications*.
9. ISO/IEC 13818-3:1998, *Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio* (Note: known as MPEG-2 Audio).
10. ISO/IEC 13818–6, *Information technology—Generic coding of moving pictures and associated audio information—Part 6: Extensions for DSM-CC*.
11. ITU-T (CCITT) Recommendation X.208 (1988), *Specification of Abstract Syntax Notation One (ASN.1)* | ISO/IEC 8824: 1990, *Information Technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1).*
12. ITU-T (CCITT) Recommendation X.209 (1988) *Specification of Basic Encoding rules for abstract syntax notation one (ASN.1)* | ISO/IEC 8825: 1990, *Information technology—Open Systems Interconnection—ASN.1 encoding rules—Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*
13. ITU-T Recommendation H.222.0 (1995) | ISO/IEC 13818–1: 1996, *Information technology—Generic coding of moving pictures and associated audio information: Systems* (Note: known as MPEG-2 Systems).

    ISO/IEC 13818-1/Amendment 1: 1997, *Registration procedure for "copyright identifier"*.
    ISO/IEC 13818-1/Amendment 2: 1997, *Registration procedure for "format identifier"*.
    ISO/IEC 13818-1/Amendment 3: 1998, *Private data identifier*.
14. ITU-T Recommendation H.262 | ISO/IEC 13818–2, *Information technology—Generic coding of moving pictures and associated audio information: Video* (Note: known as MPEG-2 Video).

    ISO/IEC 13818–2 /Amendment 1: *Registration procedure for "copyright identifier"*.

## 2.2. Other Normative References

### 2.2.1 ATSC (Advanced Television Systems Committee)

1. ATSC A/52: *Digital audio compression standard* (AC-3).
   available at ftp://ftp.atsc.org/pub/Standards/A52.
2. ATSC A/53: *Digital television standard for HDTV transmission*.
   available at ftp://ftp.atsc.org/pub/Standards/A53.



### 2.2.2 ANSI (American National Standards Institute)

1. ANSI SMPTE 274M-1995, *Television - 1920x1080 Scanning and interface*.
2. ANSI SMPTE 296M-1997, *Television - 1280x720 Scanning, analog and digital representation and analog interface.*

### 2.2.3 Apple Corporation Inc.

1. *AIFF-C Audio Interchange File Format, version C, allowing for Compression.*
2. *Bento Specification, Revision 1.0d5*, July 15, 1993.

### 2.2.4 ETSI (European Telecommunications Standards Institute)

1. ETR 162 (October 1995): *Digital broadcasting systems for television, sound and data services: Allocation of Service Information (SI) codes for Digital Broadcasting (DVB) systems.*
2. ETR 211: *Digital broadcasting systems for television, sound, and data services; Guidelines for the usage of Service Information (SI) in Digital Video Broadcasting (DVB) systems.*
3. ETS 300 468 (January 1997*): Specification for Service Information (SI) in DVB Systems Informative Annex C: Conversion Between Time and Date Conventions.*
4. ETS 300 472, *Digital broadcasting systems for television, sound, and data services; Specification for conveying ITU-R System B Teletext in Digital Video Broadcasting (DVB) bitstreams*.
5. ETS 300 743, *Digital Video Broadcasting (DVB), DVB subtitling.*
6. ETS 300 777-2, *Use of Digital Storage Media Command and Control (DSM-CC) for basic multimedia applications.*
7. ETSI DI / MTA-01074, *Multimedia Terminals and Applications, Application Programming Interface (API) for DAVIC Service Information.*

### 2.2.5 SCTE (Society of Cable Telecommunications Engineers, Inc)

1. SCTE DVS/026 - *Digital Video : Subtitling methods for Broadcast Cable.*

### 2.2.6 SMPTE (Society of Motion Picture and Television Engineers)

See ANSI (American National Standards Institute)

### 2.2.7 W3C

1. CSS-1, *Cascading Style Sheets, level 1*; by Håkon Wium Lie and Bert Bos, 17-December-96. available at http://www.w3.org/TR/REC-CSS1-961217.
2. HTML 3.2, *HyperText Mark-up Language reference specification*, by Dave Raggett, 14-Jan-1997. available at http://www.w3.org/TR/REC-html32.html.
3. PNG *Portable Network Graphics version 1*, 01-October-1996. available at http://www.w3.org/TR/REC-png.html.

# 3. Definitions

This clause defines new terms, and the intended meaning of certain common terms, used in this part of ISO/IEC 16500. Annex A of ISO/IEC 16500-1 defines additional terms and, in some cases, alternative interpretations that are appropriate in other contexts. For convenience, the normative definitions below are included in the annex.

**3.1. access control:** Provides means to access services and protection against the unauthorized interception of the services.

**3.2. anchor:** one of two ends of a hyperlink

**3.3. application:** a set of objects that provides an environment for processing Application Service Layer information flows.

**3.4. Application Programming Interface (API):** set of inter-layer service request and service response messages, message formats, and the rules for message exchange between hierarchical clients and servers. API messages may be executed locally by the server, or the server may rely on remote resources to provide a response to the client.

**3.5. assets:** Things that a user sees or hears, e.g., bitmap, audio, text.

**3.6. character:** an atom of textual information, for example a letter or a digit

**3.7. conditional access:** A means of allowing system users to access only those services that are authorized to them.

**3.8. Content Item:** A collection of content items / Content Item Elements that will form a complete application or a complete programme.

**3.9. Content Item Element: t**he smallest (and indivisible) content component.

**3.10. Content package:** A set of content Item Elements and/or content items for transfer across an A10 interface between Content Provider and Service Provider Systems.

**3.11. Content Provider:** one who owns or is licensed to sell content.

**3.12. Control Word:** the secret key used for a scrambling algorithm.

**3.13. Delivery System (DS):** The portion of the DAVIC System that enables the transfer of information between DS-users.

**3.14. element:** a component of the hierarchical structure defined by a document type definition; it is identified in a document instance by descriptive markup, usually a start-tag and end-tag.

**3.15. end-tag:** descriptive markup that identifies the end of an element

**3.16. encryption:** a mathematical technique used to ensure the confidentiality of security management information.

**3.17. Entitlement Control Message (ECM):** conditional access messages carrying an encrypted form of the control words or a means to recover the control words, together with access parameters, i.e., an identification of the service and of the conditions required for accessing this service.

**3.18. Entitlement Management Message (EMM):** conditional access messages used to convey entitlements or keys to users, or to invalidate or delete entitlements or keys.

**3.19. key management:** The generation, storage, distribution archiving, deletion, revocation, registration, and deregistration of cryptographic keys.

**3.20. hyperlink:** a relationship between two anchors

**3.21. joint stereo:** a coding option in MPEG-1 audio that exploits the redundancy between the left and right audio channels

**3.22. logical interface:** an interface where the semantic, syntactic, and symbolic attributes of information flows is defined. Logical interfaces do not define the physical properties of signals used to represent the information. A logical interface can be an internal or external interface. It is defined by a set of information flows and associated protocol stacks.

**3.23. monomedia component:** a collection of data representing a single type of audiovisual information.



**3.24. monospace format:** a presentation format of characters in which each character utilizes a character matrix of the same size, independent of the width and height of the character.

**3.25. multimedia component:** a collection of data comprising one or more multimedia components

**3.26. navigation:** the process of reaching a service objective by means of making successive choices; the term may be applied to the selection of a service category, a service provider or an offer within a particular service.

**3.27. protocol:** set of message formats (semantic, syntactic, and symbolic rules) and the rules for message exchange between peer layer entities (which messages are valid when).

**3.28. real-time stream:** an MPEG-2 transport stream containing monomedia components of which the timing of the decoding and presentation in an STU is controlled by the characteristics of the stream during the delivery of the stream to the STU.

**3.29. rendering:** the process in the STU to combine one or more monomedia components such as characters, text, and graphical objects into one presentation on a screen.

**3.30. scrambling:** The process of making a signal unintelligible at the transmission point in order that it can only be received if an appropriate descrambling system is in place at the point of reception. Scrambling can be applied to audio, video or data signals

**3.31. server:** any service providing system.

**3.32. Service Information (SI):** Digital data describing the delivery system, content and scheduling/timing of MPEG-2 Transport Streams. It includes MPEG-2 PSI together with independently defined extensions.

**3.33. Service Provider:** an entity that provides a service to a client.

**3.34. session:** an interval during which a logical, mutually agreed correspondence between two objects exists for the transfer of related information. A session defines a relationship between the participating users in a service instance.

**3.35. Set Top Box (STB):** a module that comprises both Set Top Unit (STU) and Network Interface Unit (NIU) functional elements. The STB may be either “integrated” or “modular”. An integrated STB is designed for connection to a single DAVIC A1 or equivalent interface. A modular STB may be equipped with a DAVIC A0 or equivalent interface to enable connection of a range of NIUs.

**3.36. Set Top Unit (STU):** a module that contains the “network independent” functionalities of a Set Top Box (STB). The following functionalities are contained in a typical STU:- Processing & Memory Functions; MPEG2 Demux & AV Decoders; Graphics Display; Modulator Output for TV; Peripheral Interfaces.

**3.37. start-tag:** descriptive markup that identifies the start of an element

**3.38. tag:** markup that delimits an element

**3.39. virtual machine (VM):** An abstract specification of a micro-processor and its behaviour

NOTE: A VM may be implemented on different hardware processors. A VM therefore implements the mechanism for all these processors to execute the same instruction set. It is also possible for a micro-processor to be designed so that its instruction set is identical to that of a VM. VM code can be used to make software portable. In the context of DAVIC, the VM is used to extend interoperability by allowing program code produced once to be delivered to and executed on any compliant STU.



# 4. Acronyms and abbreviations

This clause defines the acronyms and abbreviations used in this part of ISO/IEC 16500. Annex B of ISO/IEC 16500-1 defines acronyms and abbreviations used within ISO/IEC 16500.

| | |
|---|---|
| AC-3 | ATSC A52 Audio |
| AIFF | Audio Interchange File Format |
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Exchange |
| ASN.1 | Abstract Syntax Notation 1 |
| ATSC | Advanced Television Systems Committee |
| BNF | Backus-Naur Form |
| bslbf | Bit string left bit first |
| CA | Conditional Access |
| CIE | Content Item Element |
| CI | Content Item |
| CLUT | Color LookUp Table |
| CMSL | Content Metadata Specification Language |
| CORBA | Common Object Request Broker Architecture |
| CPS | Content Provider System |
| CRC | Cyclic Redundancy Check |
| CW | Control Word |
| DIS | Draft International Standard |
| DSM-CC | Digital Storage Media - Command and Control |
| DSM-CC U-N | DSM-CC User to Network |
| DSM-CC U-U | DSM-CC User to User |
| DTS | Decoding Time Stamp |
| DVB | Digital Video Broadcasting |
| DVB-SI | DVB - Service Information |
| ECM | Entitlement Control Message |
| EMM | Entitlement Management Message |
| EPG | Electronic Program Guide |
| ESC | End Service Consumer |
| ESCS | End-Service Consumer System |
| ESP | End Service Provider |
| ESPS | End-Service Provider System |
| ETS | European Telecommunications Standard |
| ETSI | European Telecommunications Standards Institute |
| ETR | European Telecommunications Recommendation |
| FIFO | First In First Out |
| fpvsbf | Floating point value sign bit first |



| | |
|---|---|
| HDTV | High Definition Television |
| HTML | HyperText Markup Language |
| ID | Identification |
| IDL | Interface Definition Language |
| IEC | International Electrotechnical Commission |
| IETF | Internet Engineering Task Force |
| IP | Internet Protocol |
| IPR | Intellectual Property Rights |
| ISO | International Organization for Standardization |
| ITU | International Telecommunications Union |
| Mbps | Megabits per second |
| LSB | Least Significant Bit |
| MHEG | Multimedia and Hypermedia information coding Experts Group |
| MPEG | Moving Picture Experts Group |
| MPEG-TS | MPEG-2 Transport Stream |
| MSB | Most Significant Bit |
| MUX | Multiplex |
| NIU | Network Interface Unit |
| NPT | Normal Play Time |
| NTSC | National Television Systems Committee |
| OMG | Object Management Group |
| OMG-UNO | Object Management Group - Universal Networked Object |
| OS | Operating System |
| OSI | Open Systems Interconnection (Reference Model) |
| PC | Personal Computer |
| PCR | Program Clock Reference |
| PES | Packetized Elementary Stream |
| PID | Packet Identifier, or Program Identification |
| PMT | Program Map Table |
| PN | Program Number (MPEG-2) |
| PNG | Portable Network Graphics (specified by W3C) |
| PSI | Program Specific Information (MPEG-2) |
| PTS | Presentation Time Stamp |
| QoS | Quality of Service |
| RDM | Reference Decoder Model |
| RGB | Red Green Blue |
| RPC | Remote Procedure Call |
| RTE | RunTime Engine |
| SCS | Service Consumer System |
| SCTE | Society of Cable Telecommunications Engineers, Inc |

| | |
|---|---|
| SGML | Standard Generalized Markup Language |
| SI | Service Information |
| SPS | Service Provider System |
| SPV | Service Provider |
| SRC | Service Related Control |
| STU | Set-Top Unit |
| TBD | To be defined |
| TCP | Transmission Control Protocol |
| tcimsbf | Two's complement integer, msb (sign) bit first |
| TS | Transport Stream |
| T-STD | Transport System Target Decoder |
| TV | Television |
| UDP | User Datagram Protocol |
| uimsbf | Unsigned integer most significant bit first |
| UNO | Universal Networked Object |
| UTC | Universal Coordinated Time |
| VCR | Video Cassette Recorder |
| VM | Virtual Machine |
| W3C | World Wide Web Consortium |

# 5. Conventions

The style of this Part of ISO/IEC 16500 follows the *Guide for ITU-T and ISO/IEC JTC 1 cooperation. Appendix II: Rules for presentation of ITU-T | ISO/IEC common text (March 1993).*



# 6. Monomedia Components

The basic monomedia components in ISO/IEC 16500 include the following elements with their associated coding formats. Various options within the coding standards and other file and packaging formats are allowed as detailed in the following clauses. If the content is encrypted the following descriptions pertain to the decrypted data.

**Table 6-1. — Coding options of monomedia components**

| *Monomedia component* | *Coding Options* |
|---|---|
| Characters | ISO 10646-1 |
| Text | HTML 3.2 |
| Outline Fonts | Defined in this specification |
| Language Information | ISO 639, part 2 |
| Service Information | ETS 300-468 |
| Telephone Numbers | ETS 300 468 |
| Compressed Audio | MPEG-1 Audio, AC-3 Audio |
| Scaleable Audio | MPEG-2 BC Audio |
| Linear Audio | AIFF-C |
| Compressed Video | MPEG-2 Video (note that this includes constrained MPEG-1 video) |
| Still Picture | MPEG-2 Intra Frame and MPEG-2 Systems |
| Compressed graphics | ETS 300 743 |
| Compressed Character Data | SCTE DVS/026 |
| Network graphics | PNG (Portable Network Graphics) |
| *Note : Network Graphics replaces the monomedia component 'uncompressed graphics' from DAVIC 1.0 up to DAVIC 1.2.* | |



## 6.1. Character Information

In ISO/IEC 16500 coding of characters is based on Unicode, ISO 10646-1, to support multilingual text. Rather than extending the mandatory Latin characterset in each DAVIC compliant STU, as defined in DAVIC 1.0, ISO/IEC 16500 provides a mechanism to download character images that are not residently available in a STU. To this effect character images can be downloaded. ISO/IEC 16500 does not specify a coding format for character images. Future versions of DAVIC specifications are expected to adopt the result of the ongoing work in the World Wide Web Consortium on an outline format for character coding.

Character images for the Latin-1 characters as defined in ISO 8859-1 are mandatorily supported in ISO/IEC 16500 STUs. The STU shall at least be capable to display 24 lines per screen with 40 monospaced characters per line with respect to ITU-R 601.

## 6.2. Text Information

The following requirements on coding of text information have been identified :

- the syntax provides a tag for hyperlinks consistent with MHEG-5 Hypertext class anchor
- the syntax provides tags for rendering control: new line (left, right, center, justify), italics, underline, bold, emphasis, strong emphasis, font selection, colour selection, size selection
- the syntax and semantics are suitable for verification in the Reference Decoder Model
- the syntax is extensible in order to accommodate the expected evolution of text coding requirements
- the syntax is efficient with respect to coding and parsing
- the syntax allows for error resilience
- the character encoding encompasses the full Latin character set (ISO 8859-1)
- the character encoding is multilingual and allow expansion (ISO 10646-1)
- the syntax allows control over the starting corner, text flow direction and rotation
- the syntax provides additional tags for sub-script, super-script, embedding bitmap objects and language specifiers

ISO/IEC 16500 is based on the HTML 3.2 Reference Specification (W3C Recommendation 14-Jan-1997), including the support of RFC 2070 Internationalization of the Hypertext Markup Language, as well as the support of Cascading Style Sheets (CSS). The specification supports the marking of fragments of text as being in a certain language or writing direction (left to right or right to left), so that proper formatting can be applied to them.

Language specifiers are included in the HTML 3.2 specification. The language is identified with attributes that are formed according to RFC 1766, which in turn is based on ISO 639 and ISO 3166. Also superscripts and subscripts are supported in HTML 3.2, as well as alignment attributes on most elements and selection of colour and size using the FONT tag. Increased presentational control and greater separation of structural and presentation information is provided by (Cascading) Style Sheets. Style Sheets also allow targeting for different types of display, for example, there might be one stylesheet for typical PC/workstation display, another for display on an ISO/IEC 16500 compliant STU, another for handheld mobile devices, and a fourth for a laser printer.

ISO/IEC 16500 text is coded as either a stand-alone <BODY> element or a full <HTML> element, both as defined in HTML 3.2. ISO 10646-1 as specified in HTML 3.2 is used both as the document character set as well as the character encoding scheme.

### 6.2.1 Decoding requirements on HTML support

DAVIC compliant decoders shall act upon all HTML 3.2 tags, except the following elements which may be optionally acted upon:

1. all HTML 3.2 deprecated elements; these include: LISTING, XMP.
2. all table-based elements; these include: TABLE, TD, TH, TR.
3. all form-based elements; these include: ISINDEX, FORM, INPUT, SELECT, TEXTAREA.
4. all applet or script-based elements (such as JavaScript), which should be parsed, but may not be displayed (except any alternate text (ALT); these include: APPLET, SCRIPT.
5. all client-side image-map elements; these include MAP.
6. all URL modification tags; these include: BASE.

Tags specified in RFC2070 shall also be parsed, and it is recommended that these should be acted upon (in addition to the HTML 3.2 reference specification); these include: LANG, DIR, ALIGN, Q, SPAN plus additional character entities.

Tags specified in Cascading Style Sheets, level 1 (W3C Recommendation 17 Dec 1996) shall also be parsed, and it is recommended that these should be acted upon (in addition to the HTML 3.2 reference specification); these include: STYLE, STYLESHEET, CLASS, ID.

Additional tags shall be parsed and may either be discarded if no alternative text (ALT) is available, or display the alternative text if it is available, or may act upon the tag if it is known.



### 6.2.2 HTML encoding

ISO/IEC 16500 specifies that HTML 3.2 documents are encoded using a variant of UTF-8 (Amendment 1 to ISO/IEC 10646-1) so that Unicode characters can be supported directly within HyperText objects. This variant of UTF-8 specifies the following :

- the null byte, (byte) 0, is encoded using the two-byte format, rather than the one-byte format — this ensures that there are no embedded null's in a UTF-8 string.
- Only one-, two- and three-byte formats are used; longer variants are not recognised.

This variant of UTF-8 encoding is also used in the Sun Java virtual machine.

The UTF-8 encoding ensures that the ASCII subset of Unicode is transmitted in a single byte, whilst other Unicode characters are transmitted in two or three bytes. This approach offers a significant advantage when most characters lie within the ASCII portion of Unicode.

See Clause 6.2.4 for an informative list of DAVIC supported HTML tags.

### 6.2.3 HTML mapping to MHEG elements

Various elements within an HTML document require a mapping to MHEG elements; these include: font-based elements and colour-based elements.

- Fonts: HTML supports a relative sizing model for fonts, ranging from 1–7. A mapping between HTML font sizes and MHEG fonts is to be provided.
- Colours: HTML specifies colours as a 24-bit RGB hex triple, e.g. #FF0000. Such colours may be mapped to MHEG absolute colours by considering the most significant 4 bits of each of the HTML RGB components to be mapped directly to each of the RGB components of an MHEG RGBα16 absolute colour. By default, the α component of the MHEG RGBα16 absolute colour may be considered to be opaque, i.e. 15.

### 6.2.4 Informative list of supported HTML Tags

In addition to standard SGML entities, such as document type and comments, the support of tags from HTML3.2 by ISO/IEC 16500 includes :

**Table 6-2. — Examples of HTML 3.2 tags supported by ISO/IEC 16500**

| *HTML Tag* | *Meaning* |
|---|---|
| A | Anchor: create a hyperlink |
| ADDRESS | Address: the enclosed text is an address |
| B | Bold: format the enclosed text as bold |
| BASEFONT | Base font: Specify the font size for subsequent text |
| BIG | Big: format the enclosed text using a bigger font |
| BLOCKQUOTE | Block quotation: the enclosed text is a block quotation |
| BODY | Body: delimit the body of the document text |
| BR | Break: break the current text flow |
| CAPTION | Caption: specify a caption for a table |
| CENTER | Centre: center the enclosed text |
| CITE | Citation: the enclosed text is a citation |
| CODE | Code: the enclosed text is a code sample |
| DD | Definition: define the definition portion of an element in a definition list |
| DFN | Definition: the enclosed text is a definition of a term |

| | |
|---|---|
| DIR | Directory: create a “directory” list |
| DIV | Division: create a division within a document |
| DL | Definition list: create a list of definitions and their terms |
| DT | Definition term: define the term portion of an element in a definition list |
| EM | Emphasis: the enclosed text should be emphasised |
| FONT | Font: set the size / colour of the font |
| H(1-6) | Heading: the enclosed text is a level 1–6 heading |
| HEAD | Head: delimit the head of a document |
| HR | Horizontal rule: draw a horizontal rule |
| HTML | HTML: delimit the extent of the whole HTML document |
| I | Italic: format the enclosed text as italic |
| IMG | Image: insert an image into the current text flow |
| KBD | Keyboard: the enclosed text is keyboard-like input |
| LI | List item: an item in an ordered or unordered list |
| LINK | Link: define a link between the current document and another document specified in the head — this can be used for CSS |
| MENU | Menu: define a menu list |
| META | Meta information: provide additional information about a document |
| OL | Ordered list: delimit an ordered list |
| P | Paragraph: define the start (and end) of a paragraph |
| PLAINTEXT | Plain text: format the remainder of the document as pre-formatted plain text |
| PRE | Pre-formatted text: format the enclosed text in its original pre-formatted version |
| SAMP | Sample: the enclosed text is a sample |
| SMALL | Small: format the enclosed text using a smaller font |
| STRIKE | Struck-though: format the enclosed text as struck through with a horizontal line |
| STRONG | Strong: strongly emphasize the enclosed text |
| STYLE | Style: used for cascading style sheets (CSS) |
| SUB | Subscript: format the enclosed text as a subscript |
| SUP | Superscript: format the enclosed text as a superscript |
| TITLE | Title: specify the title of an HTML document |
| TT | Typewriter text: format the enclosed text in a typewrite (mono-spaced) font |
| U | Underlined text: format the enclosed text as underlined |
| UL | Unordered list: delimit an unordered list |
| VAR | Variable: the enclosed text is a variable |



## 6.3. Outline Font Format

### 6.3.1 Requirements

General requirements for a font format to support DAVIC are listed below. The DAVIC font format may be implemented in a static or dynamic process. For example, a set of fonts may exist in ROM of a set-top unit, and are rendered on demand by applications executing within the set-top.

The DAVIC font format can be used by content providers in several ways. First, the content provider may render or rasterize the content and produce a MPEG, or PNG data that is then broadcast down to the set-top unit. Secondly for down loaded applications that require a variety of fonts, servers can dynamically create small DAVIC font objects that contain only the character data required to image the content, and they are downloaded to the set-top. Once the application has no further need, the small font objects can be deleted.

Specific details about dynamic generation of this format and rendering this format are outside the scope and purpose of this document. Those details are available from DAVIC member, Bitstream Inc. The DAVIC font format is intended to be used by members of DAVIC or developers of DAVIC technology. The requirements for the font format which extends the current bitmap DAVIC technology is as follows.

- **Platform Independent font format:** DAVIC requires a byte stream format that can be transferred to any platform within a DAVIC network. Network nodes may be running a variety of operating systems on a variety of processors so it is necessary the format is reduced to a byte stream of known format. Implementation of the format across processors and operating systems must not favor or require a particular processor or operating system.
- **Scaleable outline font format**: The request for a scaleable cubic outline format is required to resolve the differing resolutions or aspect ratios of the target devices. The format must be capable of supporting multiple fonts (logical fonts) with a variable number of characters. The format must be capable of supporting any font style or treatment for the purpose of imaging within a DAVIC device. In the event that static output devices such as printers are connected to a DAVIC device, the format must be suitably rendered in the resolution of the output device.
- **Character Encoding:** Digital content that references characters within DAVIC environments are based on UNICODE and the font format must fully support UNICODE.
- **Default Character Set for Resident Font(s) within DAVIC devices**: DAVIC devices may contain the ISO-Latin 8859-1 character set in a Bitmap or Outline format. It is desirable to extend the current text and UNICODE character access mechanism to support outline font technology. Augmenting the character set dynamically is highly desirable so that multi-lingual digital content can be imaged and displayed. Manufactures may implement any extra character sets to support target markets within their devices statically or by dynamic downloading. The default ISO-Latin 8859-1 character set is always assumed to be within the DAVIC device.
- **Dynamic Merging of character sets:** Character access via UNICODE implies multi-lingual support and can be accomplished by downloading either bitmap or outline font fragments. The selected font format must be easy to combine font fragments so that font fragments appear as a single font with a single unified character set.
- **Memory, Font Format Size and Compression:** Size of the outline font format must not burden the DAVIC devices with unnecessary data (ROM or RAM) or require specialized decompression logic. The selected format must be dense or tightly coded, and not require the commercial licensing of data compression logic.
- **Floating Point Processing:** DAVIC compliant network nodes, may not contain a FPP so the selected font format must not require floating point processing.
- **Font Effects and Rendering Considerations:** The selected scaleable cubic outline font format must be able to be rendered as follows:
  - Scaled independently in X and Y to accommodate asymmetric resolutions. Scaling is also required to produce bitmaps for devices which have higher resolutions, such as printers which may be connected to the set-top boxes in the future.
  - Scale to any arbitrary device context (in case static printouts are desired in the future)
  - Rotation of characters or text to any angle. This is important for down-loadable applications that wish to provide these effects. Orthogonal rotations as well as any increment in 10ths of a degree are required.



- Slanting or Obliqueing or shearing by using a transform matrix rather than requiring an Italic font must be optionally callable from any application. Usually a shear of 12 degrees is used to obtain an Italic effect. Shearing must be specified to any angle in 10ths of degree increments.
- Anti-aliased to arbitrary levels
- Filtered, gamma corrected, colored or blended
- Be made transparent or opaque

- **Format and technology availability:** DAVIC requires that the selected format be available to DAVIC members at no charge. DAVIC also requires commercially available technology to dynamically generate the format and render the format.
- **Industry synergy:** DAVIC devices may be connected to the Internet in the future, and therefore are expected to be capable of viewing Internet content. Synergy with the Internet industry and other standards is desirable.

### 6.3.2 Font Format Specification

The DAVIC specification of the format for outline fonts is contained in Annex A of this specification.

### 6.3.3 Font rendering

To ensure consistent text composition of rendering engines, the use of the following parameters in the rendering process are recommended :

- A default anti-alias bit depth of four.
- A default sub-pixel character placement of one quarter pixel accuracy.
- The use of a method for calculation of character string and algorithm for rounding to the nearest sub-pixel boundary with the same result as from the method given in the following example :

  Consider the word "DAVIC" to be written at a specified position (x0, y0) in device coordinates. Device coordinates should be stored in fractional pixels. DAVIC recommends to use a 16.16 representation of each coordinate in a 32-bit word. In other words, to describe x and y coordinates each in units of 1/65536 pixels.

  The first character 'D' is rendered with its character origin at the nearest sub-pixel boundary relative to (x0, y0). With a sub-pixel accuracy of the recommended 1/4 pixel, the rounded character origin (x, y) is calculated by:

  x = (x0 + 0x00002000) & 0xFFFFC000;

  y = (y0 + 0x00002000) & 0xFFFFC000;

  The set width of the 'D' is stored in the PFR in metricsResolution units. This is transformed into device coordinates by multiplying it by the current transformation matrix (CTM). The CTM is the product of the outputMatrix and the fontMatrix. The fontMatrix defines the size of the font in pixels per user unit. The outputMatrix defines the transformation from user units to pixels. Both matrices are expressed in 16.16 units. Multiplication of one 16.16 unit by another preserves the full precision except, of course, for the ultimate rounding to the nearest 1/65536 pixel.

  The formal expression for the transformation of the setWidth is:

$$\begin{vmatrix} \text{setWidth.x} \\ \text{setWidth.y} \end{vmatrix} = \begin{vmatrix} \text{outputMatrix} \end{vmatrix} * \begin{vmatrix} \text{fontMatrix} \end{vmatrix} * \begin{vmatrix} \text{charWidth} \\ 0 \end{vmatrix}$$

  where the value of charWidth is equal to the PFR-defined set width of the 'D' in metrics resolution units, multiplied by 65536 and then divided by its metrics resolution.

  The current position is then updated by adding (setWidth.x, setWidth.y) to it. Note that the rounding required to render the 'D' is not incuded in the update to the current position. This avoids any accumulation of rounding errors across a line and ensures that each character is positioned with the best possible accuracy.



The updated current position is then used to position the letter 'A'. Like the first character, it is rendered at the nearest sub-pixel boundary to its current position.

If pair kerning is enabled, the character pair 'D' and 'A' are looked up in the list of kerning pairs for the currently selected font. If a matching kerning pair is found, the adjustment is transformed into device coordinates by multiplying it with the CTM and the result added to the current position prior to rendering the letter 'A'.

If track kerning is enabled, the track kerning adjustment associated with the currently selecter font and the current nominal point size is also transformed into device coordinates by multiplying it with the CTM and the result added to the current position prior to rendering the letter 'A'.

If both pair kerning and track kerning apply to an inter-character space, the track and pair kerning adjustments are added together before transforming the composite adjustment into device coordinates. This improves performance by saving a transformation operation. It also improves precision by eliminating a rounding error.

The process of applying intercharacter kerning adjustments and rendering characters continues until the 'C' has been rendered. No kerning adjustment is applied after the last character. If additional characters are appended to the string, the appropriate kerning adjustment should be applied prior to the first character. This is unlike the first character in a new string.

## 6.4. Language Information

In ISO/IEC 16500 coding of language information shall be based on ISO 639-2. If an elementary stream in a program represents a specific language, then the ISO 639 descriptor shall be used in the Program Map Table to identify that language. In the case of a compressed audio stream with two independent channels, each representing a different language, both languages shall be identified by first including the ISO 639 descriptor for channel 1, immediately followed by the same descriptor for channel 2.

## 6.5. Service Information

This clause describes the service information (SI) data which forms a part of compliant bitstreams, in order to provide the user with information to assist in the selection of services and/or events within the bitstream. In addition it provides physical transmission information to enable the set-top unit to access a service.

In ISO/IEC 16500 the Service Information format shall comply to the ETS 300 468 specification.

In addition, the use of service information in ISO/IEC 16500 shall adhere to the SI implementation guidelines, specified in ETR 211.

## 6.6. Telephone Numbers

In ISO/IEC 16500 the coding of telephone numbers shall be according to the telephone number segmentation format specified for the ETSI SI telephony descriptor in ETSI ETS 300-468. This permits a common telephone number segmentation to be referenced by application objects in the STU. This would allow, for example the telephone number provided as part of a televote service to be automatically modified according to the users wishes before the number was dialed.



## 6.7. Compressed Audio

ISO/IEC 16500 compressed audio includes two methods of compression that can be used to realise two sets of functionalities. These methods are described in the following subsections.

### 6.7.1 Compressed audio coding using MPEG-1 Audio

MPEG-1 Audio (ISO/IEC 11172-3) shall be coded with the following constraints:

- Compressed audio shall use MPEG-1 Layers I and II coding (layer = '11' or '10')
- Compressed audio shall be in single channel, dual channel, joint stereo or stereo.
- Compressed audio shall use a sampling rate of 32 kHz, 44.1 kHz, or 48 kHz.
- For Layer I, compressed audio may have each bit rate permitted in the range between 32 and 448 kbits/sec (32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416 and 448) , with the exception of the free format bit rate.
- For Layer II, compressed audio may have each bit rate permitted in the range between 32 and 384 kbits/sec (32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320 and 384), with the exception of the free format bit rate.
- Compressed audio shall not apply the free format bit rate (bitrate_index = '0000' is forbidden).
- Compressed audio shall have no emphasis (emphasis = '00').
- Compressed audio shall include the parity word check (crc_check) in each audio frame.

Note that the bit rate may be switched in compressed audio streams on audio frame boundaries.

### 6.7.2 Compressed audio coding using ATSC A/52 Audio

ATSC A/52 Audio, including support for multichannel surround sound, shall be coded with the following constraints:

- ATSC A/52 compressed audio shall be constrained to a maximum bit rate of 448 kb/s.

Note that the bit rate may be switched in compressed audio streams on audio frame boundaries.

Further information on coding and carriage of ATSC A/52 Audio in ATSC systems is described in Annex L.

## 6.8. Scaleable Audio

This subclause describes the information representation of audio content to be used in environments where scaleability is needed ( i.e. a level of performance beyond the capability of the compressed audio tools in subclause 6.7 ‘Compressed Audio’), such as delivery of audio over bandwidth limited media or over media with contention (e.g. the internet or intranets with packet loss and variable delay, or over mobile channels), and non-guaranteed rates.

For scaleable audio, ISO/IEC 13818-3 (Second Edition, 1997) shall be used with the following constraints:

- Scaleable audio shall use MPEG-2 Layer II coding (layer = ‘10’)
- Scaleable audio shall not apply the free format bit rate (bitrate_index = ‘0000’ is forbidden).
- Scaleable audio shall have no emphasis (emphasis = ‘00’).
- Scaleable audio shall include the parity word check (crc_check) in each audio base_frame.
- Scaleable audio shall not apply prediction (mc_prediction_on = ‘0’)
- Scaleable audio shall not apply multilingual (no_of_multi_lingual_ch = ‘000’)
- Scaleable audio decoders shall support a switch in base bit rate and extension bit rate on audio frame boundaries.

## 6.9. Linear Audio

Linear audio shall be coded using AIFF-C. The specification of AIFF-C is contained in Annex B of this specification. AIFF-C describes a very versatile audio coding format. In the AIFF-C specification, the audio sample is broken into ‘chunks’, four of which must be present in all audio samples: the Form chunk, the Format Version chunk, the Extended Common chunk, and the Sound Data chunk. The Form chunk must be present at the beginning of the audio sample. All other chunks including user defined chunks, referred to herein as Private chunks, are allowed to exist in any order after the Form chunk. Multiple instantiations of all chunks except the Format Version chunk and the Sound Data chunk are allowed within a sample.

*Note : Support of Linear Audio as a real-time stream in future versions of the DAVIC specifications may require concatenation of objects containing linear audio with a play back duration of up to 0.7 second each.*



## 6.10. Compressed Video

Video information shall be coded using MPEG-1 Video (ISO/IEC 11172-2) or MPEG-2 Video (ISO/IEC 13818-2). For MPEG-1 Video, the data must follow the constrained parameter set, i.e. in the MPEG-1 Video data the constrained_parameter_flag shall be set to '1'. For MPEG-2 Video, the data shall conform to Main Profile syntax. Next to coding of video up to the standard video resolution as specified in ITU-R 601, DAVIC specifies coding of video at higher resolutions. Informative Annex L provides examples of video input formats that may be used.

Constraints for the coding of video with a resolution up to ITU-R 601 are specified in Clause 6.10.1. Clause 6.10.2 specifies constraints for the coding of video with resolutions higher than ITU-R 601.

### 6.10.1 Coding constraints for video with a resolution up to ITU-R 601

#### 6.10.1.1 Main Profile

For the coding of video with a resolution up to ITU-R 601, the constraints defined by MPEG for the Main Profile at Main Level (MP@ML) shall apply.

#### 6.10.1.2 Aspect Ratio of 4:3 and Pan Vectors

Compressed video shall have a source aspect ratio of 1:1, 4:3 or 16:9. Specifically, the aspect_ratio_information shall have the value '0001', '0010' or '0011'. It is recommended that pan vectors for a 4:3 window are included in the video bitstream when the source aspect ratio is 16:9. The vertical component of each pan vector shall be zero. If pan vectors are included, then the sequence_display_extension shall be present in the bitstream and the aspect_ratio_information shall be set to '0010' (4:3 display). The display_vertical_size shall be equal to the vertical_size_value. The display_horizontal_size shall contain the resolution of the target 4:3 display. The value of the display_horizontal_size field may be calculated by the following formula :

display_horizontal_size = (3/4) * (horizontal_size_value)

The table below gives some typical examples.

**Table 6-3. — Some typical values for display_horizontal_size for 4:3 window in 16:9 picture**

| *horizontal_size_value* | *display_horizontal_size* |
|---|---|
| 720 | 540 |
| 704 | 528 |
| 544 | 408 |
| 528 | 396 |
| 480 | 360 |
| 352 | 264 |



#### 6.10.1.3 Full Screen

If no sequence _display_extension is present, then the coded picture size shall be any of the values from Table 6-4, with the required upsampling ratios for full screen display on 4:3 and 16:9 monitors with 720 pixels per line. Note that DAVIC does not constrain the horizontal display resolution of the STU to 720 pixels; the applied display resolution is fully at the discretion of the implementation.

**Table 6-4. — Horizontal Upsampling Ratios for Full Screen Picture Sizes**

| *coded_picture_size* [6] | *source_aspect_ratio* | *horizontal upsampling ratios to 720 active pixels per line* | |
|---|---|---|---|
| | | *4:3 monitor* | *16:9 monitor* |
| 720 x 576 | 4:3 | x 1 | x (3/4) [1] |
| | 16:9 | x (4/3) [7] | x 1 |
| 704 x 576 | 4:3 | x 1 [3] | x (3/4) [1] |
| | 16:9 | x (4/3) [3] [7] | x 1 |
| 544 x 576 | 4:3 | x (4/3) | x 1 [1] |
| | 16:9 | x (16/9) [7] | x (4/3) |
| 528 x 576 | 4:3 | x (4/3) | x 1 [1] |
| | 16:9 | x (16/9) [7] | x (4/3) |
| 480 x 576 | 4:3 | x (3/2) | x (9/8) [1] |
| | 16:9 | x 2 [7] | x (3/2) |
| 352 x 576 | 4:3 | x 2 | x (3/2) [1] |
| | 16:9 | x (8/3) [7] | x 2 |
| 352 x 288 | 4:3 | x 2 [2] | x (3/2) [1] [2] |
| | 16:9 | x (8/3) [2] [7] | x 2 [2] |
| 720 x 480 | 4:3 | x 1 | x (3/4) [1] |
| | 16:9 | x (4/3) [7] | x 1 |
| 704 x 480 | 4:3 | x 1 | x (3/4) [1] |
| | 16:9 | x (4/3) [7] | x 1 |
| 640 x 480 | 4:3 [4] | x (9/8) [5] | x (27/32) [1] |
| 544 x 480 | 4:3 | x (4/3) | x 1 [1] |
| | 16:9 | x (16/9) [7] | x (4/3) |
| 528 x 480 | 4:3 | x (4/3) | x 1 [1] |
| | 16:9 | x (16/9) [7] | x (4/3) |
| 480 x 480 | 4:3 | x (3/2) | x (9/8) [1] |
| | 16:9 | x 2 [7] | x (3/2) |
| 352 x 480 | 4:3 | x 2 | x (3/2) [1] |
| | 16:9 | x (8/3) [7] | x 2 |
| 352 x 240 | 4:3 | x 2 [2] | x (3/2) [1] [2] |
| | 16:9 | x (8/3) [2] [7] | x 2 [2] |

*note 1 : this upsampling may be optional as 16:9 monitors can (in general) be switched to operate in 4:3 mode.*
*note 2 : also vertical upsampling x 2*
*note 3: upsampling to a window with a width of 704 pixels within an active area with a width of 720 pixels*
*note 4 : in this case the aspect_ratio_information field may be coded with '0001', indicating an aspect ratio of 1*
*note 5 : x (11/10) in case of 704 active pixels per line*
*note 6 : the coded_picture_size is defined to be equal to the horizontal_size_value by the vertical_size_value*
*note 7 : this upsampling is only applied to the (3/4)x(horizontal_size_value) pixels from the 16:9 picture to be displayed on the 4:3 display*



### 6.10.1.4 Non-Full Screen Pictures

If the sequence_display_extension is present, and the coded picture size is smaller than or equal to the display picture size, then the display picture size shall be any of the values from Table 6-5, with the required upsampling ratios for display on 4:3 and 16:9 monitors with 720 active pixels per line. For the purpose of this clause, the coded picture size is smaller than or equal to the display picture size if both the horizontal size and the vertical size of the coded picture are smaller than or equal to the horizontal size and the vertical size of the displayed picture respectively.

**Table 6-5. — Horizontal Upsampling Ratios For Non-Full Screen Pictures**

| *displayed_picture_size* [6] | *permitted coded_picture_size* | *source_aspect_ratio* | *horizontal upsampling ratios* 4:3 monitor | 16:9 monitor |
|---|---|---|---|---|
| 720 x 576 | <=720 x <=576 | 4:3 | x 1 | x (3/4) [1] |
| | | 16:9 | x (4/3) [7] | x 1 |
| 704 x 576 | <=704 x <=576 | 4:3 | x 1 [3] | x (3/4) [1] |
| | | 16:9 | x (4/3) [7] | x 1 |
| 544 x 576 | <=544 x <=576 | 4:3 | x (4/3) | x 1 [1] |
| | | 16:9 | x (16/9) [7] | x (4/3) |
| 528 x 576 | <=528 x <=576 | 4:3 | x (4/3) | x 1 [1] |
| | | 16:9 | x (16/9) [7] | x (4/3) |
| 480 x 576 | <=480 x <=576 | 4:3 | x (3/2) | x (9/8) [1] |
| | | 16:9 | x 2 [7] | x (3/2) |
| 352 x 576 | <=352 x <=576 | 4:3 | x 2 | x (3/2) [1] |
| | | 16:9 | x (8/3) [7] | x 2 |
| 352 x 288 | <=352 x <=288 | 4:3 | x 2 [2] | x (3/2) [1] [2] |
| | | 16:9 | x (8/3) [2] [7] | x 2 [2] |
| 720 x 480 | <=720 x <=480 | 4:3 | x 1 | x (3/4) [1] |
| | | 16:9 | x (4/3) [7] | x 1 |
| 704 x 480 | <=704 x <=480 | 4:3 | x 1 | x (3/4) [1] |
| | | 16:9 | x (4/3) [7] | x 1 |
| 640 x 480 | <=640 x <=480 | 4:3 [4] | x (9/8) [5] | x (27/32) [1] |
| 544 x 480 | <=544 x <=480 | 4:3 | x (4/3) | x 1 [1] |
| | | 16:9 | x (16/9) [7] | x (4/3) |
| 528 x 480 | <=528 x <=480 | 4:3 | x (4/3) | x 1 [1] |
| | | 16:9 | x (16/9) [7] | x (4/3) |
| 480 x 480 | <=480 x <=480 | 4:3 | x (3/2) | x (9/8) [1] |
| | | 16:9 | x 2 [7] | x (3/2) |
| 352 x 480 | <=352 x <=480 | 4:3 | x 2 | x (3/2) [1] |
| | | 16:9 | x (8/3) [7] | x 2 |
| 352 x 240 | <=352 x <=240 | 4:3 | x 2 [2] | x (3/2) [1] [2] |
| | | 16:9 | x (8/3) [2] [7] | x 2 [2] |

*note 1 : this upsampling is optional as 16:9 monitors can (in general) be switched to operate in 4:3 mode.*
*note 2 : also vertical upsampling x 2*
*note 3: upsampling to a window with a width of 704 pixels within an active area with a width of 720 pixels*
*note 4 : in this case the aspect_ratio_information field may be coded with '0001', indicating an aspect ratio of 1*
*note 5 : x (11/10) in case of 704 active pixels per line*
*note 6 : displayed_picture_size is defined to be equal to the display_horizontal_size by the display_vertical_size*
*note 7 : this upsampling is only applied to the (3/4)x(horizontal_size_value) pixels from the 16:9 picture to be displayed on the 4:3 display*



In the absence of control by the high-level API over the position of non-full screen video, the position of the coded picture video on the display (as defined by the displayed picture size), is defined by the pan vectors, when included. If no pan vectors are included, a default position shall be assumed in the center of the displayed picture. If the high-level API takes control over the position, the pan vectors, when included, may be disregarded.

DAVIC supports trick mode for compressed video by indicating the trick mode in the PES packet header in which the MPEG video data is contained, fully compliant to ISO/IEC 13818-1.

## 6.10.2 Coding constraints for video with a resolution beyond ITU-R 601

### 6.10.2.1 High profile

The allowable parameters shall be bounded by the upper limits specified for the Main Profile at High Level.[1] Additionally, the MPEG-2 video data shall meet the constraints and specifications described in the following sections[2].

### 6.10.2.2 Syntactical Constraints

The following tables list the allowed values for each of the ISO/IEC 13818-2 syntactic elements which are restricted beyond the limits imposed by MP@HL. In these tables conventional numbers denote decimal values, numbers preceded by **0x** are to be interpreted as hexadecimal values and numbers within single quotes (e.g., '10010100') are to be interpreted as a string of binary digits. The following table identifies parameters in the sequence header of a bit stream that shall be constrained and lists the allowed values for each.

**Table 6-6. — Sequence Header Constraints**

| *Sequence header syntactic element* | *Allowed value* |
|---|---|
| horizontal_size_value | see Table 6-7 |
| vertical_size_value | see Table 6-7 |
| aspect_ratio_information | see Table 6-7 |
| frame_rate_code | see Table 6-7 |
| bit_rate_value (≤ 38.8 Mbps) | ≤ 97000 |
| vbv_buffer_size_value | ≤ 488 |

### 6.10.2.3 Compression Format Constraints

The following table lists the allowed compression formats.

**Table 6-7. — Compression Format Constraints[3]**

| *vertical_size_ value* | *horizontal_size_ value* | *aspect_ratio_ information* | *frame_rate_ code* | *progressive_ sequence* |
|---|---|---|---|---|
| 1080[4] | 1920 | 1, 3 | 1,2,3,4,5 | 1 |
| | | | 3,4,5 | 0 |
| 1035[4] | 1920 | 3 | 4,5 | 0 |
| 720 | 1280 | 1,3 | 1,2,3,4,5,6,7,8 | 1 |
| 576 | 720 | 2, 3 | 6 | 1 |
| | | | 3 | 1 |
| | | | 3 | 0 |
| | 704 | 2, 3 | 6 | 1 |
| | | | 3 | 1 |



[1] See ISO/IEC 13818-2, Section 8 for more information regarding profiles and levels.

[2] The additional constraints and specifications are based on ATSC A/53 and, in part, on ITU-R BT.1208.

[3] Shaded areas describe formats up to ITU-R resolution. This information is provided for convenience and does not constrain in any manner the functionality for coding of video up to ITU-R 601 resolution.

[4] Note that 1088 lines are actually coded in order to satisfy the MPEG-2 requirement that the coded vertical size be a multiple of 16 (progressive scan) or 32 (interlaced scan) and to provide a common coding format for 1035 and 1080.

| *vertical_size_ value* | *horizontal_size_ value* | *aspect_ratio_ information* | *frame_rate_ code* | *progressive_ sequence* |
|---|---|---|---|---|
| | | | 3 | 0 |
| | 544 | 2, 3 | 3 | 1 |
| | | | 3 | 0 |
| | 528 | 2, 3 | 3 | 1 |
| | | | 3 | 0 |
| | 480 | 2, 3 | 3 | 1 |
| | | | 3 | 0 |
| | 352 | 2, 3 | 3 | 1 |
| | | | 3 | 0 |
| 480 | 720 | 2, 3 | 2,5,7,8 | 1 |
| | | | 1, 4 | 1 |
| | | | 5 | 0 |
| | | | 4 | 0 |
| | 704 | 2, 3 | 2,5,7,8 | 1 |
| | | | 1,4 | 1 |
| | | | 5 | 0 |
| | | | 4 | 0 |
| | 640 | 1, 2, 3 | 2,5,7,8 | 1 |
| | | | 1,4 | 1 |
| | | | 5 | 0 |
| | | | 4 | 0 |
| | 544 | 2, 3 | 1,4 | 1 |
| | | | 4 | 0 |
| | 528 | 2, 3 | 1,4 | 1 |
| | | | 4 | 0 |
| | 480 | 2, 3 | 1,4 | 1 |
| | | | 4 | 0 |
| | 352 | 2, 3 | 1,4 | 1 |
| | | | 4 | 0 |
| 288 | 352 | 2, 3 | 3 | 1 |
| 240 | 352 | 2, 3 | 1,4 | 1 |

| *Legend for MPEG-2 coded values* | | | | | |
|---|---|---|---|---|---|
| aspect_ratio_information | 1 = square samples | 2 = 4:3 display aspect ratio | 3 = 16:9 display aspect ratio | | |
| frame_rate_code | 1 = 23.976 Hz | 2 = 24 Hz | 3 = 25 Hz | 4 = 29.97 Hz | 5 = 30 Hz | 6 = 50 Hz |
| | 7 = 59.94 Hz | 8 = 60 Hz | | | | |
| progressive_sequence | 0 = interlaced scan | 1 = progressive scan | | | | |



### 6.10.2.4 Sequence Extension Constraints

The following table identifies parameters in the sequence extension part of a bit stream that shall be constrained by the video subsystem and lists the allowed values for each. A sequence_extension structure is required to be present after every sequence_header structure.

**Table 6-8. — Sequence Extension Constraints**

| *Sequence extension syntactic element* | *Allowed values* |
|---|---|
| progressive_sequence | see Table 6-7 |
| profile_and_level_indication | see Note |
| chroma_format | ‘01’ |

| *Sequence extension syntactic element* | *Allowed values* |
|---|---|
| horizontal_size_extension | '00' |
| vertical_size_extension | '00' |
| bit_rate_extension | '0000 0000 0000' |
| vbv_buffer_size_extension | '0000 0000' |
| frame_rate_extension_n | '00' |
| frame_rate_extension_d | '0000 0' |

Note: The profile_and_level_indication field shall indicate the lowest profile and level defined in ISO/IEC 13818-2, Subclause 8, that is consistent with the parameters of the video elementary stream.

#### 6.10.2.5 Sequence Display Extension Constraints

The following table identifies parameters in the sequence display extension part of a bit stream that shall be constrained by the video subsystem and lists the allowed values for each.

**Table 6-9. — Sequence Display Extension Constraints**

| *Sequence display extension syntactic element* | *Allowed values* |
|---|---|
| video_format | '000' |

The preferred and default values for color_primaries, transfer_characteristics, and matrix_coefficients are defined to be SMPTE 274M[5] (value 0x01 in all three cases). While all values described by MPEG-2 are allowed in the transmitted bit stream, it is noted that SMPTE 170M values (0x06 in all three cases) will be the most likely alternate in common use.

#### 6.10.2.6 Picture Header Constraints

In all cases other than when vbv_delay has the value 0xFFFF, the value of vbv_delay shall be constrained as follows:

vbv_delay • 45000

### 6.10.3 Decoding tool requirements

#### 6.10.3.1 Decoding tools for video with a resolution up to ITU-R 601

DAVIC tools for decoding video with resolutions up to ITU-R 601 shall decode bitstreams of all of the specified resolutions into the output for which the specific STU was designed. When video encoded with 525 lines at a framerate of 29.97 Hz is delivered to the STU, the STU shall produce a baseband NTSC output signal or some other video output signal for which the STU was designed. Similarly, when video encoded with 625 lines at a framerate of 25 Hz is delivered to the STU, the STU shall produce a baseband PAL output signal or some other video output signal for which the STU was designed.

#### 6.10.3.2 Decoding tools for video with a resolution beyond ITU-R 601

DAVIC tools for decoding video with resolutions higher than ITU-R 601 shall decode bitstreams of all of the specified resolutions into the output for which the specific STU was designed. The informative Annex K explains some implementation examples for providing compatibility with each video format such that each decoder can decode and display each DAVIC input bitstream resolution.



[5] At some point in the future, the color gamut may be extended by allowing negative values of RGB and defining the transfer characteristics for negative RGB values.

## 6.11. Still Pictures

Each still picture shall be encoded as an MPEG-2 Intra Frame; see ISO/IEC 13818-2. In the case of a series of still pictures with a presentation schedule associated to a time base (e.g. a series representing a slide show), then the still picture feature of MPEG-2 Systems (ISO/IEC 13818-1) shall be used. For the definition of still pictures in terms of MPEG Systems refer to Clause 2.1.52 of ISO/IEC 13818-1. In the case of a single still picture (bitmap), only ISO/IEC 13818-2 applies.

### 6.11.1 Normal resolution still pictures

For still pictures with coded picture size smaller than or equal to 720 pixels by 576 lines, the picture size constraints specified in Clause 6.10.1 apply.

### 6.11.2 Higher resolution still pictures

For still pictures with coded picture size larger than 720 pixels by 576 lines, the picture size constraints specified in Clause 6.10.2 apply.

# 6.12. Compressed Graphics

## 6.12.1 Requirements

The following prioritized list of criteria and requirements for compression of graphics have been identified. A larger value indicates a lower priority.

1. Ease and speed of software implementation in a STU
2. Minimum use of memory in the STU
3. Visual appearance while decoding is in progress in the STU
4. Lossless
5. Efficiency of compression
6. Technology available without IPR royalty

## 6.12.2 Format for Compressed Graphics

In ISO/IEC 16500 compressed graphics shall be coded using ETS 300 743, also known as the DVB subtitling system.

### 6.12.2.1 Real-Time Compressed Graphic streams

If the presentation of the graphics is associated to the decoder system time clock (STC as defined in ISO/IEC 13818-1) then ETS 300 743 is fully applicable.

### 6.12.2.2 Single bitmaps with Compressed Graphics

In the case of single bitmaps with compressed graphics of which the presentation is not associated to a time base then each bitmap is represented by a “Region Composition Segment” followed by each “Object Data Segment” referred to; for the definition of these segments refer to Clauses 1.8.2 and 1.8.4 of ETS 300 743. For decoding a single bitmap with compressed graphics a separate colour palette may be available that fully complies to the “CLUT definition segment” as specified in Clause 1.8.3 of ETS 300 743. For single bitmaps with compressed graphics the encoded value of the following fields of Region Composition Segments, Object Data Segments and CLUT Definition Segments may be disregarded at decoding:

- page_id
- region_id
- region_version_number
- object_version_number
- CLUT-version_number

### 6.12.2.3 Default CLUT for single bitmaps with Compressed Graphics

ISO/IEC 16500 specifies a default CLUT for single bitmaps with compressed graphics. The default CLUT is applicable when the use of no other CLUT is specified in this bitmap. A Visible Object, when referring to a Palette Object, will interpret specified IndexColours as entries in the default CLUT, except for Bitmap objects that use any specific Palette Object.

#### *6.12.2.3.1 Requirements*

The default CLUT for use by single bitmaps with compressed graphics is to meet the following requirements :

- the default CLUT shall define a CLUT with 256 entries
- the default CLUT shall support transparency values of 0 %, 25 %, 50 %, 75 % and 100 %
- the colours defined by default CLUT shall have a perceptually uniform distribution over the colour space



- the default CLUT shall leave some colours available for private use
- the colours defined by the default CLUT, when used in an STU implementation as the basic colours for the presentation of graphic images, are suitable for rendering any graphic image
- the R, G and B components of the colours defined by the default CLUT are orthogonal, i.e. quantization of a random colour towards a colour defined by the default CLUT is possible for each R, G or B component independently of the value and quantization of the other components.

#### *6.12.2.3.2 Default CLUT characteristics*

To meet the above requirements, a default CLUT is defined with properties as given in the following table, where for each transparency level the total number of CLUT entries is given, as well as the quantization levels for the Red, Green and Blue component of the colour associated to the entry. Each possible combination of component quantization levels is allowed for. As a consequence the total number of entries per transparency level is found by multiplying the number of quantization levels of the Red, Green and Blue components for that transparency level. The default CLUT reserves 12 entries for private use.

**Table 6-9. — Default CLUT characteristics**

| *Transparency level* | *Total number of CLUT entries* | *Quantization levels for Red* | *Quantization levels for Green* | *Quantization levels for Blue* |
|---|---|---|---|---|
| 0 %<br>(fully opaque) | 135 | 5 levels :<br>0, 63, 127, 191, 255 | 9 levels :<br>0, 31, 63, 95, 127, 159, 191, 223, 255 | 3 levels :<br>0, 127, 255 |
| 25 % | 56 | 4 levels :<br>0, 85, 170, 255 | 7 levels :<br>0, 42, 85, 127, 170, 212, 255 | 2 levels :<br>0, 255 |
| 50 % | 36 | 3 levels :<br>0, 127, 255 | 6 levels :<br>0, 51, 102, 153, 204, 255 | 2 levels :<br>0, 255 |
| 75 % | 16 | 2 levels :<br>0, 255 | 4 levels :<br>0, 85, 170, 255 | 2 levels :<br>0, 255 |
| 100 %<br>(fully transparent) | 1 | - | - | - |
| privately defined | 12 | privately defined | privately defined | privately defined |

#### *6.12.2.3.3 Default CLUT specification*

The colours of the default CLUT are defined in Annex C of this specification.

## 6.13. Compressed Character Data

Compressed character data shall be coded using SCTE DVS/026, defined by the Society of Cable Telecommunications Engineers (SCTE).

## 6.14. Network Graphics

As format for Network Graphics DAVIC supports the PNG format as defined by the World Wide Web Consortium (W3C Recommendation 01 October 1996, available at http://www.w3.org/TR/REC-png-multi.html).



# 7. Monomedia Streams

## 7.1. Types of Monomedia Components

DAVIC distinguishes three types of monomedia components. The first type are implied monomedia components; implied components are either included within other monomedia components (e.g. characters within text) or are implied by some other mechanism defined by DAVIC, such as the use of Service Information within an MPEG-2 Transport stream. The monomedia components which are not of the implied type are one of the two following types :

- Monomedia streams. The monomedia components of which the presentation requires synchronization with a time base are represented as monomedia streams. For such synchronization the mechanisms provided by MPEG Systems, ISO/IEC 13818-1, are used. To enable the use of time stamps as defined in ISO/IEC 13818-1, each monomedia stream is packetized in PES packets.
- Stand-alone monomedia components. Monomedia components that do not require synchronization with a time base at play back, are referred to as stand-alone monomedia components. Stand-alone monomedia components are not packetized in PES packets.

Table 7-1 defines which type each DAVIC defined monomedia component may take. Some monomedia components can be of more than one type, specifically they can be a monomedia stream and a stand-alone component. Specifically a graphical object may either be a single bitmap (such as a button for user control) or a graphical representation of text used within a graphical stream for subtitling. Furthermore a still picture may be a single bitmap, e.g. representing a background for a downloaded application, or may be part of a series of still pictures representing a slide show.

**Table 7-1. — Types of monomedia components**

| *Monomedia component* | *Representation* |
|---|---|
| Characters | implied component, e.g. in Text and Service Information |
| Text | stand-alone component |
| Outline Fonts | stand-alone component |
| Language Information | implied component, e.g. in Text and Service Information |
| Service Information | implied component in MPEG-2 Transport Stream |
| Telephone Numbers | implied component, e.g. in downloaded Application |
| Compressed Audio | stream |
| Scaleable Audio | stream |
| Linear Audio | stream |
| Compressed Video | stream |
| Still Picture series | stream |
| Still Picture bitmap | stand-alone component |
| Compressed graphic real-time stream | stream |
| Compressed graphic bitmap | stand-alone component |
| Compressed character data stream | stream |
| Network graphic bitmap | stand-alone component |

## 7.2. Real-time and Stored Monomedia Streams

For monomedia streams ISO/IEC 16500 distinguishes real-time streams and stored streams, which differ in the process which drives their play back.

- The playback of real-time streams is real time driven; real-time streams are played back in reference to a time base which is controlled by the PCR fields from the MPEG-2 Transport Stream that contains the real-time stream. With real-time streams the timing of the decoding and presentation in an STU is controlled by the characteristics of the stream during the delivery of the stream to the STU.
- The timing of the playback of stored streams is controlled by an application. The stream is stored in STU memory, providing the functionality of a local server that can be controlled from an application that also resides in the STU. As an example, in a certain application it may be possible for the user to play back a stored stream by pushing a specific button on a Remote Control.

Also transmission requirements differ for real-time streams and stored streams. In the case of real-time streams, constraints for real-time delivery apply to avoid problems such as buffer underflow and overflow. For synchronization of real-time streams and stored streams during playback, the mechanisms provided by ISO/IEC 13818-1 are applied by ISO/IEC 16500. Therefore monomedia streams are stored in PES packets.

While the data structure used to carry real-time streams and stored streams in PES packets is the same, there are important consequences for the encoding and interpretation of PTSs and DTSs. In the case of real-time streams the time base of the program is used, as specified in ISO/IEC 13818-1 for MPEG audio and MPEG video. For real-time AC-3 audio streams the time base of the program specified in ISO/IEC 13818-1 is also used. In the case of stored streams an independent time base may be used, but the phase of the time base for playback of the stored streams is determined by the application. Each time a stored stream is played back, the time base (system time clock) starts with a value of zero.

Table 7-2 summarizes for each ISO/IEC 16500 non-implied monomedia component whether it can be defined as a real-time stream, as a stored stream object and whether it is contained in a PES packet.

**Table 7-2. — Real-time stream, stored stream and use of PES packets per monomedia component**

| *Monomedia component (non-implied)* | *Real-time stream* | *Stored stream* | *contained in PES packets* |
|---|---|---|---|
| Text | no | no | no |
| Outline Fonts | no | no | no |
| Compressed Audio | yes | yes | yes |
| Scaleable Audio | yes | yes | yes |
| Linear Audio | no | yes | yes |
| Compressed Video | yes | yes | yes |
| Still Picture series | yes | yes | yes |
| Still Picture bitmap | no | no | no |
| Compressed graphic real-time stream | yes | yes | yes |
| Compressed graphic bitmap | no | no | no |
| Compressed character data stream | yes | no | no |
| Network graphic bitmap | no | no | no |



## 7.3. Carriage of Monomedia Streams in PES Packets

Monomedia streams are carried in PES packets, with the only exception of Compressed Character Data. One of the advantages of applying PES packets is that they provide a general structure with the following major features :

- identification of monomedia components.
- association of Time Stamps with monomedia streams. Time Stamps (PTSs and DTSs) are essential to ensure synchronisation and to ensure correct buffer behavior in the T-STD defined in ISO/IEC 13818-1 and in the Reference Decoder Model for contents decoding defined in ISO/IEC 16500.
- association of trick mode operation with monomedia streams, specifically compressed video.
- association of copyright information with monomedia streams.

### 7.3.1 Packetization of MPEG- and ATSC-defined Components

The packetization of MPEG- and ATSC-defined monomedia streams in PES packets shall fully comply to ISO13818-1. In addition the following constraints shall apply :

- In the case of a stored stream containing compressed audio, the first byte of the first PES packet with data from the stored stream shall be the first byte of an MPEG or AC-3 audio frame, i.e. the first byte of a sync word. In addition the last byte of the last PES packet containing data from the stored stream shall be the last byte from an audio frame. Consequently, each stored audio stream must have an integer number of audio frames.
- In the case of a stored stream containing compressed video or a series of still pictures, the first byte of the first PES packet with data from the stored stream shall be the first byte of a sequence_start_code. In addition, the PES_packet_length field shall not be encoded with the value of zero. The sequence header associated with this start code shall be followed immediately by an I-picture, optionally preceded by a GOP header. The last byte of the last PES packet containing data from the stored stream shall be the last byte of a sequence_end_code. Consequently, each stored video or still picture stream must have an integer number of video frames.

### 7.3.2 Packetization of DVB-defined Components

The packetization of DVB subtitling in PES packets shall fully comply to ETS 300 743.

### 7.3.3 Packetization of DAVIC-defined Components

The packetization in PES packets of monomedia streams with a coding format defined by ISO/IEC 16500 is defined in Annex D of this specification.

## 7.4. Packetization of Compressed Character Data

Compressed character data are carried in MPEG private sections, defined in ISO/IEC 13818-1. The packetization of Compressed Character Data within private sections shall fully comply to SCTE DVS/026.

# 8. Transport of Monomedia Streams and Components

## 8.1. Transport of Real Time Streams

Real time streams are mapped directly to the MPEG-2 Transport Stream, without using DSM-CC. For real-time streams containing compressed audio, compressed video, or a series of still pictures, the mapping shall comply to ISO/IEC 13818-1. In the case of ISO/IEC 16500 defined compressed graphics, the mapping shall comply to ETS 300 743. In the case of Compressed character data the mapping shall comply to SCTE DVS/026.

## 8.2. Transport of Stored Streams

In ISO/IEC 16500 the transport method of stored streams depends on whether upstream information flow is used by the delivery system. For applications which utilize upstream information flow, the stored streams may be transmitted in reply to the DSM-CC File Read function (refer to clause 7.3.8.1; file access) with the UNO remote procedure call, data representation and transport mechanism (refer to clause 7.2.2, user-to-user interaction; 7.3.1 DSM-CC option choices summary; and 7.3.2, remote procedure call) as specified in ISO/IEC 16500-5. Alternatively, the stored streams are transmitted in “User-to-User-Object Carousels” within MPEG-2 Transport Streams, using private MPEG-2 sections, as specified by DSM-CC.

## 8.3. Transport of Stand-alone Monomedia Components

To transport stand-alone monomedia components the same mechanisms are used as for stored streams; see Clause 8.2.



# 9. Application Format

## 9.1. Application Interchange Format

To deliver multimedia information to STUs in an interoperable way, applications shall use the MHEG-5 final form interchange format, as defined by ISO/IEC 13522-5. The ASN.1 notation and encoding, as defined by Annex A of ISO/IEC 13522-5, shall be used to interchange MHEG-5 objects. This format defines the semantics and the encoding of the multimedia and hypermedia objects. Subclause 9.2 below specifies the MHEG-5 options whose support by the DAVIC platform is mandatory or optional, as well as the semantic extensions to MHEG-5 related to the DAVIC application domain.

To deliver program code to STUs in an interoperable way, applications shall use the MHEG-5 InterchangedProgram class to encapsulate Java[6] VM code, according to the semantics and encoding defined by ISO/IEC 13522-6. Java VM classes are called from MHEG-5 objects using the MHEG-5 Call and Fork actions.

The Java VM code interchange unit is a Java VM class. Java VM classes shall be encoded as defined by the *Class File Format* section of the *Java Virtual machine specification*. A Java class encapsulates data and methods that consist of sequences of instructions. The instruction set is defined by the *Java Virtual machine instruction set* section of the *Java Virtual machine specification*.



[6] Java is a trademark or a registered trademark of Sun Microsystems, Inc.

## 9.2. MHEG-5 Profile for the DAVIC Application Domain

This subclause specifies the features of ISO/IEC 13522-5 that shall be supported by the DAVIC application domain, according to the application domain definition principles set by Annex D of ISO/IEC 13522-5.

### 9.2.1 Object Interchange Format

The ASN.1 notation defined in Annex A of ISO/IEC 13522-5 shall be used as the application interchange format of ISO/IEC 13522-5, as well as the Distinguished Encoding Rules (DER) for the encoding of the interchanged objects.

### 9.2.2 Set of Classes

The following set of MHEG-5 classes shall be mandatory:

> Action, Application, Audio, Bitmap, BooleanVariable, ContentRefVariable, DynamicLineArt, EntryField, HotSpot, HyperText, IntegerVariable, InterchangedProgram, Link, ListGroup, ObjectRefVariable, OctetStringVariable, Palette, PushButton, Rectangle, RemoteProgram, ResidentProgram, RTGraphics, Scene, Slider, Stream, SwitchButton, Text, TokenGroup, Video.

### 9.2.3 Set of Features

The set of mandatory and optional features shall be as defined in Table 9-1.

**Table 9-1. — Feature requirements**

| *Feature* | *Requirement* |
|---|---|
| Caching | Optional |
| Cloning | Mandatory |
| Free moving cursor | Optional |
| Scaling (Video and Bitmap) | Optional |
| Stacking of Applications | Optional |
| Trick modes | Optional |
| Ancillary connections | Optional |

### 9.2.4 Content Data Encoding

DAVIC specifies the content data encoding as defined in Table 9-2.

**Table 9-2. — Content Encoding**

| *Type of content* | *Specification (Data Types)* | *Hook values* | *Clause* |
|---|---|---|---|
| Font encoding format | ISO/IEC 16500 defined Outline Font Format | 1 | 6.3 |
| Palette encoding format | CLUT Definition Segment (ETS 300 743) [1)] | 1 | 6.12.2.2 |
| Bitmap encoding format | reserved | 1 | |
| | MPEG-2 Intra frame | 2 | 6.11 |
| | Region Definition Segment (ETS 300 743) | 3 | 6.12.2.2 |
| | PNG bitmap | 4 | 6.14 |
| Text encoding format | Subset of HTML 3.2 | 1 | 6.2 |



| | | | |
|---|---|---|---|
| EntryField encoding format | Characters encoded according to ISO 10646-1 (Unicode) or according to ISO 8859-1, depending on the value of the CharacterSet Attribute of the Entryfield | 1 | 6.1 |
| HyperText encoding format | Subset of HTML 3.2 | 1 | 6.2 |
| Stream encoding format | video: MPEG-1 Video (ISO/IEC 11172-2)<br>MPEG-2 Video (ISO/IEC 13818-2)<br>audio: MPEG-1 Audio (ISO/IEC 11172-3)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 1 [2)] | 6.10<br>6.10<br>6.7.1<br>6.12.2.1 |
| | video: MPEG-2 Still (ISO/IEC 13818-2)<br>audio: MPEG-1 Audio (ISO/IEC 11172-3)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 2 [2)] | 6.10<br>6.7.1<br>6.12.2.1 |
| | video: -<br>audio: DAVIC Linear Audio (AIFF-C)<br>rtgraphics: - | 3 [3)] | <br>6.9<br> |
| | video: MPEG-1 Video (ISO/IEC 11172-2)<br>MPEG-2 Video (ISO/IEC 13818-2)<br>audio: AC-3 Audio (ATSC A52)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 4 [2)] | 6.10<br>6.10<br>6.7.2<br>6.12.2.1 |
| | video: MPEG-2 Still (ISO/IEC 13818-2)<br>audio: AC-3 Audio (ATSC A52)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 5 [2)] | 6.10<br>6.7.2<br>6.12.2.1 |
| | video: MPEG-1 Video (ISO/IEC 11172-2)<br>MPEG-2 Video (ISO/IEC 13818-2)<br>scaleable audio: MPEG-2 Audio (ISO/IEC 13818-3, layer II)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 6 [2)] | 6.10<br>6.10<br>6.8<br>6.12.2.1 |
| | video: MPEG-2 Still (ISO/IEC 13818-2)<br>scaleable audio: MPEG-2 Audio (ISO/IEC 13818-3, layer II)<br>rtgraphics: DVB Subtitling (ETS 300 743) | 7 [2)] | 6.10<br>6.8<br>6.12.2.1 |
| LineArt encoding format | (None specified) | | |
| CursorShape encoding format | (None specified) | | |
| InterchangedProgram encoding format | MHEG-6 (ISO/IEC 13522-6) | 1 | 9.1 |

*Note 1 : Only for use by bitmaps coded in the Region Definition Segment format*

*Note 2 : For Stream objects with ContentHook 1, 2, 4, 5, 6 and 7, the value of the attribute Storage shall be "stream" or "memory".*

*Note 3 : For Stream objects with the ContentHook 3, the value of the attribute Storage shall be "memory".*



### 9.2.5 Attribute Encoding

DAVIC specifies the attribute encoding as defined in Table 9-3.

**Table 9-3. — Attribute Encoding**

| *Attribute* | *Specification (Data Types)* |
|---|---|
| Permissible FontAttributes | "<style>.<size>" with <style> being "plain", "bold", "italic", "bold-italic", "emphasis" or "strong", and <size> being an integer.<br>e.g. "bold-italic.20" |
| Permissible FontNames | "fixed" specifying a font with a fixed spacing<br>"proportional" specifying a font with proportional spacing |
| TransitionEffects | See Clause 9.2.11 |
| CharacterSet | 1: ISO 8859-1 (Latin)<br>2: ISO 10646-1 (Unicode) |
| AbsoluteColour | RGBα32, that is coding of graphics in 32 bits per pixel, allocating 8 bits to the Red, Green and Blue components, as well as 8 bits for the translucency component. The first three bytes are unsigned integers, providing the value of the red, green and blue components of the pixel respectively. The value 0 indicates minimum value, while the value 255 indicates maximum value. The last byte specifies the transparency level of the pixel. The value zero indicates an opaque pixel, while the value 255 indicates full transparency. The value shall be coded as the big-endian octet string representation of the integer value. |

### 9.2.6 UserInput Registers

DAVIC specifies the following InputEventRegisters defined in Table 9-4.

**Table 9-4. — InputEventRegisters**

| *Register #* | *UserInputEventTag* | *Name* |
|---|---|---|
| 1 | 1 | Up |
| | 2 | Down |
| | 3 | Left |
| | 4 | Right |
| | 5 - 14 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, respectively |
| | 15 | Select |
| | 16 | Exit |
| | 17 | Help |
| | 18-99 | Reserved for future specification. |
| | >99 | Vendor specific |
| 2 | 1 | Up |
| | 2 | Down |
| | 3 | Left |
| | 4 | Right |



| | |
|---|---|
| 5-14 | Forbidden |
| 15 | Select |
| 16 | Exit |
| 17 | Help |
| 18-99 | Reserved for future specification. |
| >99 | Vendor specific |

### 9.2.7 Constraints on the Use of Variables

The use of IntegerVariables shall be restricted as defined in Table 9-5.

Table 9-5. — Constraints to IntegerVariables

| *type of value* | *default value* | *Size* | *min value* | *max. value* |
|---|---|---|---|---|
| signed integer | 0 | 32 bits | -214783648 | 214783647 |

### 9.2.8 Semantic Constraints on the MHEG-5 Applications

Table 9-6 provides a list of constraints that each MHEG-5 applications shall apply to features of ISO/IEC 13522-5. STUs may support the features to a higher or lesser degree.

**Table 9-6. — Feature Constraints**

| *Feature* | *Constraint* |
|---|---|
| SceneCoordinateSystem(X,Y) | The following coordinate systems are supported:. 720x576, 720x480, 640x480 and 1080x1920. |
| SceneAspectRatio(W,H) | The following aspect ratios are supported: 1/1, 4/3 and 16/9. |
| MultipleRTGraphicsStreams(N) | One active RTGraphic stream at a time. |
| MultipleAudioStreams(N) | One active MPEG-1 audio stream at a time and one active AIFF-C audio stream at a time. |
| MultipleVideoStreams(N) | One active video stream at a time. |
| OverlappingVisibles(N) | No constraint. |
| Cloning | Mandatory |

### 9.2.9 EngineEvent

The DAVIC application domain reserves no particular value of EngineEvent. This is left to the application developer.

### 9.2.10 GetEngineSupport

DAVIC specifies no other strings for the GetEngineSupport action than the ones listed in ISO/IEC 13522-5.

### 9.2.11 TransitionEffect Parameter of the TransitionTo Elementary Action

To support transition effects between Still Picture Compositions in ISO/IEC 16500, the TransitionEffect parameter of the TransitionTo elementary action can be used, as specified in this Clause. For a graphical description of the transition effects see Annex M.

The TransmissionEffect parameter consists of 32 bits. The structure of the parameter is as follows :

Transition Effect Parameter: [parameter_1] [parameter_2 ] [parameter_3] [parameter_4] [parameter_5]

[parameter_1] : Parameter_1 is 7 bits field of type uimsbf which defines transition effect type

[parameter_2] : Parameter_2 is 3 bits field of type uimsbf which defines number of splits

[parameter_3] : Parameter_3 is 6 bits field of type uimsbf which defines duration of effect time

[parameter_4] : Parameter_4 is 8 bits field of type uimsbf which defines horizontal position of effect start

[parameter_5] : Parameter_5 is 8 bits field of type uimsbf which defines vertical position of effect start

**Table 9-7. — TransitionEffect parameter of "TransitionTo" action**

| *No.* | *Transition Effect type* | *Abbrevia-tion* | *Para-meter_1* | *Para-meter_2* | *Para-meter_3* | *Para-meter_4* | *Para-meter_5* |
|---|---|---|---|---|---|---|---|
| 0 | Reserved | | 00 | | | | |
| 1 | Simple cut | CUT | 01 | | | | |
| 2 | Dissolve | DIS | 02 | | T | | |
| 3 | Vertical wipe (top to bottom) | WVD | 03 | | T | | |
| 4 | Vertical wipe (bottom to top) | WVU | 04 | | T | | |
| 5 | Horizontal wipe (left to right) | WHR | 05 | | T | | |
| 6 | Horizontal wipe (right to left) | WHL | 06 | | T | | |
| 7 | Wipe that vertically opens from the center | WVO | 07 | | T | | Y |
| 8 | Wipe that vertically closes from both top and bottom | WVC | 08 | | T | | Y |
| 9 | Wipe that horizontally opens from the center | WHO | 09 | | T | X | |
| 10 | Wipe that horizontally closes from both left and right | WHC | 0A | | T | X | |
| 11 | Square wipe (open) | WRO | 0B | | T | X | Y |
| 12 | Square wipe (close) | WRC | 0C | | T | X | Y |
| 13 | Vertical split wipe (top to bottom) | WDD | 0D | N | T | | |
| 14 | Vertical split wipe (bottom to top) | WDU | 0E | N | T | | |
| 15 | Horizontal split wipe (left to right) | WDR | 0F | N | T | | |
| 16 | Horizontal split wipe (right to left) | WDL | 10 | N | T | | |
| 17 | Vertical slide-out (top to bottom) | SOD | 11 | | T | | |
| 18 | Vertical slide-out (bottom to top) | SOU | 12 | | T | | |
| 19 | Horizontal slide-out (left to right) | SOR | 13 | | T | | |
| 20 | Horizontal slide-out (right to left) | SOL | 14 | | T | | |
| 21 | Vertical slide-in (top to bottom) | SID | 15 | | T | | |
| 22 | Vertical slide-in (bottom to top) | SIU | 16 | | T | | |



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 23 | Horizontal slide-in (left to right) | SIR | 17 | | T | | |
| 24 | Horizontal slide-in (right to left) | SIL | 18 | | T | | |
| 25 | Vertical roll (top to bottom) | RVD | 19 | | T | | |
| 26 | Vertical roll (bottom to top) | RVU | 1A | | T | | |
| 27 | Horizontal roll (left to right) | RHR | 1B | | T | | |
| 28 | Horizontal roll (right to left) | RHL | 1C | | T | | |
| 29 | Reserved | | 1D | | | | |
| 30 | Reserved | | 1E | | | | |
| 31 | Reserved | | 1F | | | | |
| 32 | Reserved | | 20 | | | | |
| 33 | Reserved | | 21 | | | | |
| 34 | Reserved | | 22 | | | | |
| 35 | Reserved | | 23 | | | | |
| 36 | Half vertical roll (top to bottom) | RPD | 24 | | T | | Y |
| 37 | Half vertical roll (bottom to top) | RPU | 25 | | T | | Y |
| 38 | Half horizontal roll (left to right) | RPR | 26 | | T | X | |
| 39 | Half horizontal roll(right to left) | RPL | 27 | | T | X | |
| 40 | Reserved | | 28-7F | | | | |

The symbols used in Table 9-7 indicate the following:

N: An unsigned integer, specifying the number of splits as defined in Table 9-8. For the coordinate systems supported by DAVIC the start positions of the split-effects are defined in Table 9-9.

**Table 9-8. — Number of splits**

| *Parameter Value* | *Number of splits* |
|---|---|
| 0x0 | 0 |
| 0x1 | 2 |
| 0x2 | 3 |
| 0x3 | 4 |
| 0x4 | 5 |
| 0x5 | 8 |
| 0x6 | 10 |
| 0x7 | Reserved |

**Table 9-9. — Start positions of split-effects with n splits per Coordinate System**

| *Transition Effect number* | *Transition Effect type with Split Effect* | *Start position in Coordinate System (x * y)*<br>*Supported coordinate systems by DAVIC : (720 * 576), (720 * 480), (640 * 480), and (1920 * 1080). Note : the most left pixel and the top line have index value zero.* |
|---|---|---|

| | | |
|---|---|---|
| 13 | Vertical split wipe (top to bottom) | lines (y*(n-k)/n), where k=n, n-1, ..., 1 and with truncation to zero after division. |
| 14 | Vertical split wipe (bottom to top) | lines ((y-1) - (y*(n-k)/n)), where k=n, n-1, ..., 1 and with truncation to zero after division. |
| 15 | Horizontal split wipe (right to left) | pixels (x*(n-k)/n), where k=n, n-1, ..., 1 and with truncation to zero after division. |
| 16 | Horizontal split wipe (left to right) | pixels ((x-1) - (x*(n-k)/n)), where k=n, n-1, ..., 1 and with truncation to zero after division. |

T: An unsigned integer, specifying the duration time of the effect, with an inclusive range of 0 – 63 indicating the duration time of the transition effect in units of 0.5 seconds.

X: An unsigned 8 bit integer, specifying the horizontal start position of the effect. The horizontal start position indicates the horizontal index of the pixel of the coordinate system where the effect starts. The most left pixel of the coordinate system has index zero. For the four coordinate systems supported by DAVIC, 720 x 576, 720 x 480, 640 x 480 and 1920 x 1080 (see Table 9-6) the horizontal index of the pixel where the effect starts is defined by the formula in Table 9-10. If an index value larger than the index of the most right pixel of the coordinate system is indicated, then the effect is indicated to start at the most right pixel. Table 9-10 also specifies the range permitted for X.

**Table 9-10. — Coding of horizontal index of pixel where the effect starts**

| *Horizontal resolution of Coordinate System* | *Horizontal index of pixel where effect starts* | *Permitted range of X (inclusive)* |
|---|---|---|
| 640 | 4*X | 0 - 160 |
| 720 | 4*X | 0 - 180 |
| 1920 | 8*X | 0 - 240 |

Y: An unsigned 8 bit integer, specifying the vertical start position of the effect. The vertical start position indicates the vertical index of the line of the coordinate system where the effect starts. The top line of the coordinate system has index zero. For the four coordinate systems supported by DAVIC, 720 x 576, 720 x 480, 640 x 480 and 1920 x 1080 (see Table 9-6) the vertical index of the line where the effect starts is defined by the formula in Table 9-11. The formula is defined such that the effect can be defined to start in the exact center position of the coordinate system. If an index value larger than the index of the bottom line of the coordinate system is indicated, then the effect is indicated to start at the bottom line. Table 9-11 also specifies the range permitted for Y.

**Table 9-11. — Coding of vertical index of line where the effect starts.**

| *Vertical resolution of Coordinate System* | *Vertical index of line where effect starts* | *Permitted range of Y (inclusive)* |
|---|---|---|
| 480 | 2*Y | 0 - 240 |
| 576 | 4*Y | 0 - 144 |
| 1080 | 8*Y + 4 | 0 - 134 |



## 9.2.12 MHEG-5 Resident Programs

In order to improve the functionality of MHEG-5 applications, DAVIC has defined a set of MHEG-5 resident programs. These MHEG-5 resident programs offer the following facilities in a platform independent fashion :

- The ability to retrieve and manipulate date information
- The ability to request a random number

- The ability to manipulate strings
- The ability to convert an OctetString to a ContentReference or an object reference.

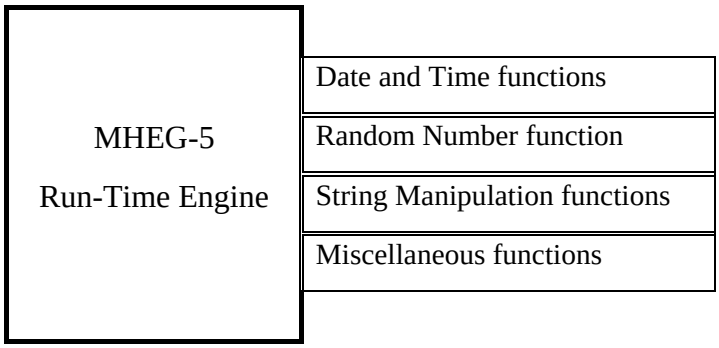



**Figure 9-12. — MHEG-5 resident programs**

#### 9.2.12.1 Date and Time functions

- `GetCurrentDate()` - Retrieves the current date and time

  Synopsis:

  ```
  GetCurrentDate(date, time)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | output | IntegerVariable | date |
  | output | IntegerVariable | time |

  Description:

  Retrieves the current date and time. The argument date is the Modified Julian date which is encoded as the number of days since Midnight on November 17, 1858 and the argument time is the current time encoded as the number of seconds since midnight. The date and time values represent ***Local Time***.

- `FormatDate()` - Format a string representing a date according to a specifiable format

  Synopsis:

  ```
  FormatDate(dateFormat, date, dateTime, datestring)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | dateFormat |
  | input | GenericInteger | date |
  | input | Genericinteger | dateTime |
  | output | OctetStringVariable | dateString |

  Description:

  Returns a string in the argument string dateString formatted according to the string argument dateFormat. The function also takes the argument date as a date encoded as the Modified Julian date (number of days since Midnight on November 17 1858) and the argument dateTime as the number of seconds since midnight. The argument string dateString is the same as the dateFormat string except for the field codes (defined below) which start with a '%' character. Where encountered in the dateFormat string, the field codes must be replaced by the indicated part of the date. The following field codes have been defined:

  '%Y' Year, 4 digits

  '%y' Year, last 2 digits

  '%X' Month, with padding zero (01-12)

‘%x’ Month, without padding zero (1-12)

‘%D’ Day, with padding zero (01-31)

‘%d’ Day, without padding zero (1-31)

‘%H’ Hour, with padding zero (00-23)

‘%h’ Hour, without padding zero (0-23)

‘%I’ Hour, with padding zero (01-12)

‘%i’ Hour, without padding zero (1-12)

‘%M’ Minutes, with padding zero (00-59)

‘%m’ Minutes, without padding zero (0-59)

‘%S’ Seconds, with padding zero (00-59)

‘%s’ Seconds, without padding zero (0-59)

‘%A’ AM/PM indication

‘%a’ am/pm indication

‘%%’ single“%” character

The following is an example of the use of this function at June 4, 1995, at 16:56, when the input argument dateFormat “%Y-%x-%d %I:%M %a” results an output argument dateString of “1995-6-4 4:56 pm”.

- `GetDayOfWeek()` - Returns the day of the week

Synopsis:

```
GetDayOfWeek(date, dayOfWeek)
```

Arguments:

| | | |
|---|---|---|
| input | Genericinteger | date |
| output | IntegerVariable | dayOfWeek |

Description:

Returns the day of the week. From the input argument date in the Modified Julian form, the argument dayOfWeek returns an integer starting from 0 representing Sunday, 1 Monday, 2 Tuesday … until 6 representing Saturday.

#### 9.2.12.2 Random Number function

- `Random()` - Returns a random number

Synopsis:

```
Random(num, random)
```

Arguments:

| | | |
|---|---|---|
| input | GenericInteger | num |
| output | IntegerVariable | random |

Description:

Returns a random integer number between 1 and num (inclusive).

#### 9.2.12.3 String Manipulation functions

In this clause a number of character string manipulation functions are defined; strings are either octet or double octet based. The semantics of a string are transparent to the string manipulation functions; that is the manipulation on a string is independent of the representation of the string. For example, when a string includes escape sequences, the octets representing such sequence are treated in the same way as any other octets.



- `GetStringLength()` - Returns the number of octets within the string

  Synopsis:

  ```
  GetStringLength(string, length)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | string |
  | output | IntegerVariable | length |

  Description:

  Returns in the output argument length the number of octets within the input argument string.

- `GetSubString()` - Extracts a sub-string from a string

  Synopsis:

  ```
  GetSubString(string, octetLengthSearch, beginExtract, endExtract,
  stringResult)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | string |
  | input | GenericInteger | octetLengthSearch |
  | input | GenericInteger | beginExtract |
  | input | GenericInteger | endExtract |
  | output | OctetStringVariable | stringResult |

  Description:

  Extracts part of a string from an octet specified by argument beginExtract upto the octet specified by argument endExtract. The octets specified by the beginExtract and endExtract arguments are included. The substring is returned in the argument stringResult. The first octet in the string starts at index 1. A string can be octet based or double octet based. The argument octetLengthSearch specifies whether the extracted string is an octet or a double octet string. A value of 1 identifies an octet string and a value of 2 a double octet string.

- `CompareString()` - Compares two strings

  Synopsis:

  ```
  CompareString(string1, string2, octetLengthSearch, result)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | string1 |
  | input | GenericOctetString | string2 |
  | input | GenericInteger | octetLengthSearch |
  | output | IntegerVariable | result |

  Description:

  Compares two octet strings using lexicographical order, the result is -1 if string1<string2, 0 if string1=string2, + 1 if string1>string2. A string can be octet based or double octet based. The argument octetLengthSearch specifies whether the search is on an octet basis or a double octet basis. A value of 1 identifies an octet string and a value of 2 a double octet string.

- `SearchSubString()` - Searches for a sub-string within a string

  Synopsis:

  ```
  SearchSubString(string, octetLengthSearch, startIndex,
                                  searchedString, stringPosition)
  ```

  Arguments:

| | | |
|---|---|---|
| input | GenericOctetString | string |
| input | GenericInteger | octetLengthSearch |
| input | GenericInteger | startIndex |
| input | GenericOctetString | searchedString |
| output | IntegerVariable | stringPosition |

Description:

Searches for a sub-string within a string from a specified starting index. The string is specified by the first argument string. The octet to start the search from is specified by the startIndex argument and the sub-string is specified by the searchedString argument. The argument stringPosition returns the index within the string of the first octet of the sub-string , -1 is returned if the string is not found. The first octet of the string is at index 1. A string can be octet based or double octet based. The argument octetLengthSearch specifies whether the comparison is on an octet basis or a double octet basis. A value of 1 identifies an octet based comparison and a value of 2 a double octet based comparison.

- `SearchAndExtractSubString()` - Searches and extracts a sub-string within a string

Synopsis:

```
SearchAndExtractSubString(string, octetLengthSearch, startIndex,
                          searchedString,          stringResult,
                    stringPosition)
```

Arguments:

| | | |
|---|---|---|
| input | GenericOctetString | string |
| input | GenericInteger | octetLengthSearch |
| input | GenericInteger | startIndex |
| input | GenericOctetString | searchedString |
| output | OctetStringVariable | stringResult |
| output | IntegerVariable | stringPosition |

Description:

Searches and extracts a sub-string within a string from a specified starting index. The string is specified by the first argument string. The octet to start from is specified by the startIndex argument and the sub-string is specified by the searchedString argument.

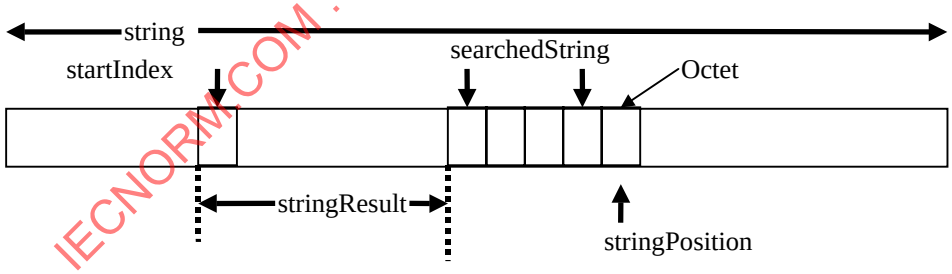



**Figure 9-13. — Searching and extracting a sub-string within a string.**

The argument stringPosition returns the index of the octet immediately after the searchedString within the string, -1 if the string is not found. The index of thefirst octet of string equals 1. The stringPosition may if the substring is the last element within the string have an index beyond the end of the string i.e. the index = the number of octets within the string + 1. The argument stringResult returns the string itself between the startIndex upto not including the searched string. The first octet of the string is at index 1. A string can be octet based or double octet based. The argument octetLengthSearch specifies whether the search is on an octet basis or a double octet basis. A value of 1 identifies an octet search and a value of 2 a double octet search.



#### 9.2.12.4 Miscellaneous functions

- `CastToContentRef()` - Casts an OctetString to a ContentReference.

  Synopsis:

  ```
  CastToContentRef(inString, outString)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | inString |
  | output | ContentReferenceVariable | outString |

  Description:

  Casts the OctetString variable inString to the ContentReference variable outString.

- `CastToObjectRef()` - Casts an OctetString and Object Identifier to an Object Reference

  Synopsis:

  ```
  CastToObjectRef(inString, ObjectId, outObjectRef)
  ```

  Arguments:

  | | | |
  |---|---|---|
  | input | GenericOctetString | inString |
  | input | GenericInteger | ObjectId |
  | output | ObjectRefVariable | outObjectRef |

  Description:

  Casts the combination of the OctetString variable inString and the Integer variable ObjectId to the Object Reference variable outObjectRef.

### 9.2.13 Protocol Mapping and External Interaction

Table 9-14 defines the mapping to the DAVIC external environment:

**Table 9-14. — Protocol Mapping**

| *MHEG-5 entity* | *Mapping needed* | *Semantics* |
|---|---|---|
| OpenConnection, CloseConnection | Mapping to connection management | • In OpenConnection:<br>  • Protocol: one of the two strings: “PSTN” or “ISDN”.<br>  • Address: E.164 NSAP. This is the address of the network service access point *and* the internet address of the service gateway to attach to. For the encapsulation of the internet address, the ATM forum specification is used. For an example see Annex N of this specification. |
| RemoteProgram objects | Mapping to RemoteProgram call protocol in the application domain (see clause 9.6) | • In Call and Fork:<br>  • Name<br>  • Parameters<br>  • ProgramConnectionTag |

| | | |
|---|---|---|
| Application name space | Mapping to name space of the application domain | • ObjectReference (see 9.9.3)<br>• ContentReference (see 9.9.4) |
| Application name space in case a TransitionTo action uses the ConnectionTag parameter | Mapping to the name space of the application domain | • ObjectReference (see 9.9.3)<br>• ContentReference (see 9.9.4) |
| Persistent storage name space | Mapping to the name space of the persistent storage | • In StorePersistent and ReadPersistent:<br>  • InFileName, OutFileName<br>Flat name space. Names may be anything from one to 64 characters long. |
| Stream actions | Mapping to the stream interface of the application domain | • In Stream<br>  • Speed<br>  • CounterPosition (see 9.9.1)<br>• In Audio, Video, RTGraphics<br>  • ComponentTag |
| Stream events | Mapping to stream states and stream events in the application domain | • In Stream<br>  • StreamPlaying, StreamStopped (mapping to application-domain stream state machine)<br>  • CounterPosition<br>  • StreamEventTag (see 9.9.1) |



## 9.3. Mapping of MHEG-5 Elements to DSM-CC U-U

### 9.3.1 Stream Events and Normal Play Time Mapping

The DSM-CC StreamEvent interface provides the possibility to carry private data in the data structure for the event, in the form of the PrivateDataByte field. These bytes shall be mapped one-to-one on the StreamEventTag of the MHEG-5 event StreamEvent.

The MHEG-5 internal attribute CounterPosition of the Stream class shall also be mapped on the value of the DSM-CC Normal Play Time of the corresponding stream. The counter position shall represent a millisecond precision. Mapping from Normal Play Time to CounterPosition shall take place by rounding the value to the nearest millisecond.

### 9.3.2 Namespace Mapping

In figure 9-15, an example of a DSM-CC file structure is given, showing a diagram of a logical DSM-CC file structure along with the object references for an application object file, a scene object file and content data files. Below is a code fragment for accessing the different objects depicted in figure 9-15.

```
{:application
      ("App1/startup" 0)}
{:scene
      ("App1/Scene1/scene1.mheg" 0)
      :Items (
      {:bitmap 1
            (:content "App1/Scene1/bitmap.graphics") ...}
      {:audio 2
            (:content "App1/Scene1/audio1.aiff") ...}
      {:video 3
            (:content "App1/Scene1/video1.mpeg") ...}
...)
```

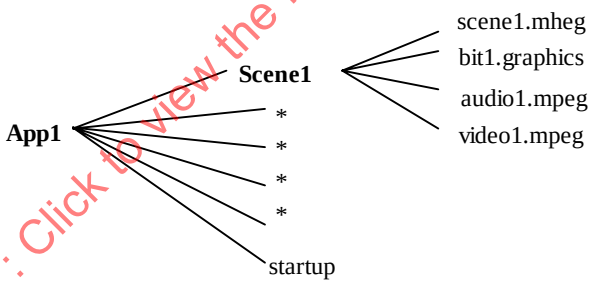



**Figure 9-15. — Searching and extracting a sub-string within a string.**

When an application starts, it is assumed that a service gateway has been located and attached to, so that there is exactly one name space within which the application objects are located. Within that name space, a service has also been located. That service is a DSM-CC directory; within it, there can be other directories, files, and streams.

Furthermore, it is assumed that each object belongs to exactly one application. This assumption is necessary to allow for unambiguous object references.

The high-level API differentiates between three types of retrieved data:

- objects that comply to the high-level API definition,
- the content (such as bitmaps or text) of those objects, and
- streams (such as video and audio).

For accessing the various objects of an application on the server side, the DSM-CC Directory, File and Stream objects are used. Note that the server, in this context, does not have to be a physical server, but could be implemented, for example, as a broadcast carousel in a pure-broadcast topology.



Each file is either a Scene object, an Application object, or the content data of an Ingredient object. Each Scene object, Application object and content data is stored in a separate file.

### 9.3.3 MHEG-5 Object References

MHEG-5 objects can be exchanged in two ways: either the object is exchanged as a DSM-CC file object or within another high-level API object. The former method is used for Applications and Scenes, the latter for all other objects contained in a scene or application object.

MHEG-5 references objects by an `ObjectReference`, consisting of an optional byte string `GroupIdentifier`, followed by an integer, the `ObjectNumber`.

For the mapping on DSM-CC, the following additional rules are defined:

1. Each Application and Scene object shall have in its `GroupIdentifier` a byte string which maps on the name of the DSM-CC file which contains that object. These objects shall have their ObjectNumber set to 0.
2. Each application shall have exactly one Application object. That object shall be contained in a DSM-CC File object with the name ‘startup’. Only one Application object shall be contained in each such file.
3. Ingredient Objects other than Application and Scene objects may
   - either leave out the `GroupIdentifier`, in which case it is assumed to be a string which maps on the name of a DSM-CC file which contains the object (Application or Scene) of which this object is a part,
   - or fill in the `GroupIdentifier` with such a string.
4. Such objects shall have their ObjectNumber set to a value which is unique within that Scene.
5. For the `GroupIdentifier`, the mapping rules defined in the following Clause apply.

#### 9.3.3.1 Mapping Rules for GroupIdentifier

All `GroupIdentifiers` are ASCII strings. They are composed of four components:

| Source | Path Origin | Path | Filename |
|---|---|---|---|

The Source component is optional, and specifies the data source to be used to retrieve the data. Each source identification is terminated with a “:”. The default source identification is “DSM:” (in upper case). ISO/IEC 16500 does not specify any further data sources, but the use of other source identifications is permitted.

##### *9.3.3.1.1 Explicitly specified data source*

The semantics defined in this clause apply in the case that the data source is defined explicitly. If the source is not explicitly defined, the semantics as described in clause 9.3.3.1.2 apply.

The Path Origin is either ‘//’ or ‘/’. If the path origin is ‘//’ then the following path and filename are to be interpreted as an absolute path starting from the root of the current service gateway to which the runtime is attached. If the path origin is ‘/’ then the following path and filename is to be interpreted as a relative path, starting from the directory that contains the current `Application` object.

The Path component is a (possibly empty) sequence of directory names, each followed by a ‘/’ character. Each directory name is to be used to delimit directory references (of the depth type).

The Filename component identifies the DSMCC object within the directory that is identified by the preceding Source, Path Origin and Path components.

For instance, if the `GroupIdentifier` is ‘DSM://apps/otherAppl’, it is mapped on the DSM-CC name ‘otherAppl’ in the directory ‘apps’ of the service gateway to which the runtime has attached. If the GroupIdentifier is ‘DSM:/scenes/myScene’, it is mapped on the DSM-CC name ‘myScene’ in the directory ‘scenes’ of the directory where the `Application` object was found.

##### *9.3.3.1.2 Shorthand Notation when data source is not explicitly defined*

In cases where the data source is not explicitly defined, the following shorthand notations may be used for references to objects accessible from the default Data Source. In ISO/IEC 16500 the default Data Source is a DSM-CC service



gateway. The rationale behind the shorthand notation is compatibility with previous versions of the DAVIC specification.

There are two abbreviations allowed. The first abbreviation is '~/' (standard ASCII tilde + slash), which may be used to specify the default Data Source and a relative Path Origin; use in ISO/IEC 16500 means 'DSM:' for the Data Source and '/' for Path Origin. This abbreviation can therefore be used to refer to objects relative to the current application.

The other abbreviation is '/' (standard ASCII slash), which may be used to specify the default Data Source and an absolute Path Origin; use in ISO/IEC 16500 means 'DSM:' for the Source and '//' for Path Origin. This abbreviation can therefore be used to refer to objects with an absolute path from the root of the current service gateway to which the runtime is attached.

For instance, an object, referred to as 'DSM:/scenes/myScene' may be referred to as '~/scenes/myScene', while the object referred to as 'DSM://apps/otherAppl' may be referred to as '/apps/otherAppl'. Thus if no source is explicitly specified, then '/' represents an absolute path name and not a relative one as would be the case if the source was defined explicitly (see clause 9.3.3.1.1).

#### *9.3.3.1.3 Other Service Gateways*

It is possible to refer to objects in another service gateway, as DSMCC objects have built-in indirection to the actual location of the data. An object with the path "DSM://Mheg5/Banking/Welcome" can refer to a DSMCC object whose actual data is only accessible from another service gateway. Normal access to such an object will result in a 'service transfer exception', which will be followed by an attempt to attach to that service gateway.

### 9.3.4 MHEG-5 Content References

MHEG-5 has a separate way of referencing the actual content of objects belonging to the Ingredient class. This is done by way of a `ContentReference`. The `ContentReference` consists of an optional `PublicIdentifier` followed by a `SystemIdentifier`, which is a byte string. The following rule shall apply.

For the `SystemIdentifier`, the exact same mapping shall be used as for the `GroupIdentifier` above. The `PublicIdentifier` shall not be used.

#### 9.3.4.1 DSMCC Stream Objects

Note that the mapping of `ContentReference` and `GroupIdentifier` allow references to both DSMCC File and DSMCC Stream objects. Although this is possible, applications shall not refer to DSMCC Stream objects using a `GroupIdentifier`. `GroupIdentifiers` are always expected to refer to DSMCC File objects.

`ContentReferences` of content of MHEG-5 Stream objects may refer to both DSMCC File and DSMCC Stream objects. If such a `ContentReference` refers to a DSMCC File object, the MHEG-5 Stream object shall have its `Storage` attribute set to 'memory'. If the `ContentReference` refers to a DSMCC Stream object, the MHEG-5 Stream object shall have its `Storage` attribute set to 'stream'.

`ContentReferences` of content of MHEG-5 Ingredients, other than Stream objects must always refer to a DSMCC File object.

### 9.3.5 Mapping of MHEG-5 ComponentTag to DSM-CC and DVB-SI

The MHEG-5 ComponentTag is mapped to the least significant byte of the association_tag as defined by ISO/IEC 13818-6. The most significant byte of the association_tag may take any value (0xXX). The value encoded in the lower significant byte (LSB) of the association_tag shall be equal to the MHEG-5 ComponentTag value. In addition, the value of the component_tag defined in the stream_identifier_descriptor specified by DVB shall also take the same value as the MHEG-5 ComponentTag. A stream_identifier_descriptor in the descriptor loop of a PID is equivalent with an association_tag_descriptor for that PID with an association_tag value of MSB=0xXX and LSB=<component_tag> and a use value of 0x0100.

### 9.3.6 Java class File Names

In addition to the interworking provisions specified by ISO/IEC 13522-6, the DSM-CC to MHEG-5 name mapping rules specified in the above subclauses shall apply to the class file names interchanged in the ContentData attribute of

an MHEG-5 InterchangedProgram object, whenever this ContentData attribute is of the ReferencedContent type, and therefore used to identify a set of files that contain the referenced Java classes.



## 9.4. Core set of Java APIs

The following set of APIs shall be used by Java VM code to express access to basic functions of the STU in an interoperable way:

- the `java.lang` package;
- the `java.util` package;
- the `java.io` package;
- the `iso.mheg5` package;
- the `org.davic.net.dsmcc.uu` package;
- the `org.davic.net.dvb.si` package;
- the `org.davic.mpeg.sections` package;
- the `org.davic.resources` package;
- the `org.davic.mpeg` package.

### 9.4.1 java.lang

The `java.lang` package, as defined by the *Java API documentation*, consists of the minimal set of Java VM classes needed to run Java VM code, supporting the following functionality: basic data types, object, mathematic operations, security, thread management, string manipulation, exception handling.

The methods load(), loadLibrary(), exec(), traceInstructions() and traceMethodCalls() in the java.lang.Runtime class and the java.lang.Process class must be implemented to some extent but their use is not standardised in a DAVIC system and hence applications using them will not be fully inter-operable.

The core java class java.lang.System includes three references to java.io being java.lang.System.in, java.lang.System.out, java.lang.System.err. For java.lang.System.in, if there is no default system input device then attempts to read from this stream shall return '-1' to indicate 'end of stream' as defined in the Java reference documents. For java.lang.System.out and java.lang.System.err, if there is no default system output device then the method call shall return immediately without any negative results. It must be possible for DAVIC applications to make method calls such as "System.out.println()" without any negative effects (e.g. unforeseen exceptions, blocking or terminating) on STUs where there is no suitable output device for that call to use.

### 9.4.2 java.util

The `java.util` package, as defined by the *Java API documentation*, consists of Java VM classes supporting a number of utility features common to all Java VM programs.

### 9.4.3 java.io

The java.io package, as defined by the Java API documentation, consists of Java VM classes supporting a basic model for input and output of non real time streams of data.

All interfaces, classes and methods defined in the reference documents for java.io shall be implemented in a STU. The classes File, FileInputStream, FileOutputStream, RandomAccessFile and FileDescriptor are required to support access to files that are defined within the DSM.CC object domain using the DSM.CC File and Directory interfaces. The name space used to reference these shall be the same name space as specified for use with MHEG-5/6 in subclause 9.5.

### 9.4.4 iso.mheg5

The `iso.mheg5` package, as defined by ISO/IEC 13522-6, provides Java VM code with access to and manipulation of the MHEG-5 multimedia presentation and interaction objects, i.e. access to the dynamic attributes of MHEG-5 objects and invocation of elementary actions on MHEG-5 objects.

### 9.4.5 The DSM-CC User to User API

DAVIC supports the `org.davic.net.dsmcc.uu` package, as defined in ETS 300 777-2. This package enables Java VM code to use the DSM-CC U-U interface objects for network data access.

The `org.davic.net.dsmcc.uu` package implements a subset of the DSM-CC U-U API defined by ISO/IEC 13818-6. Access to the following Core consumer services is provided:

- interface Base: operations Close and Destroy;
- interface File: operations Read and Write;
- interface Directory: operations Open, Close and Get;
- interface ServiceGateway: operations Attach and Detach;
- interface stream: operations Resume, Pause, Status, Reset, Play and Jump;
- interface CosNaming::NamingContext: operations List and Resolve;
- interface CosNaming::BindingIterator: operations Next_One and Next_N.

### 9.4.6 The Service Information (SI) API

The objective of this API is to allow inter-operable applications to access service information data from MPEG-2 streams. One example of such applications would be electronic program guides. This API is a relatively high level API allowing applications to access information from the SI tables in a clean and efficient way. The specification of this API is defined by ETSI DI /MTA-01074, entitled Application Program Interface (API) for DAVIC Service Information.

### 9.4.7 The MPEG-2 Section Filter API

The objective of this API (`org.davic.mpeg.sections`) is to provide a general mechanism allowing access to data held in MPEG-2 private sections. This will provide a mechanism for inter-operable access to data which is too specialized to be supported by the high level DVB-SI API or which is not actually related to service information.

The definition of the MPEG-2 section filter API is in Annex E of this specification. The API definition does not specify the lengths of the section filtering patterns. For those methods which do not specify an offset, the length of the section filtering pattern arrays shall be 8 with their mapping on to the section header as described in the last subclause of Annex E. For those methods which include an offset, the length of the section filtering pattern arrays shall be 7.

The API definition below does not specify the efficiency or effectiveness of the section filtering process. If filtering is happening with filters set beyond the 10th byte of the total section, filtering throughputs must be supported as in ISO/IEC 16500-7 subclause 12.5.3 with the restriction that support for filtered throughputs of more than 2 Mbits/second is not mandatory.

### 9.4.8 The Resource Notification API

The section filter API uses a resource notification API in the `org.davic.resources` package. This API provides a standard mechanism for applications to register interest in scarce resources and to be notified of changes in those resources or removal of those resources by the environment. The description of this API is in Annex F of this specification.

### 9.4.9 The MPEG Component API

Various MPEG related APIs use an MPEG component API in the `org.davic.mpeg.sections` package. This API provides a standard way of referring to standard MPEG features. The definition of the MPEG component API is in Annex G of this specification.



## 9.5. URL Format for Access to Broadcast Services

### 9.5.1 Introduction

DAVIC defines a specific Uniform Resource Locator (URL) format to access broadcast services. This URL format provides a general addressing mechanism intended to access broadcast services from e.g. JAVA and HTML. Note that URLs are commonly used in JAVA for addressing resources and other objects.

DAVIC broadcast networks carry the Service Information (SI) which contains globally unique parameters for locating the services in the broadcast networks. The URL format defined by DAVIC to access such services is based on these parameters as they provide an addressing mechanism in a physical network independent way. The same services may be carried simultaneously in many physical networks, but the parameters in the SI will remain the same and thus they can be used by the clients to locate the services regardless of the actual physical network.

DAVIC defined the following general format of the URLs :

<protocol>://<“server”>/<dir1>/.../<file>

The protocol (scene) part of the URL identifies that it is a broadcast service. The “server” part of the URL points to the service as the services are the basic element that is carried in the broadcast networks. The rest of the URL specifies the individual component inside a service. The format of the last part is dependent on the type of the service (this part is not needed if the URL points to the whole service).

### 9.5.2 Numerical format

The combination of original_network_id, transport_stream_id and service_id identifies a service in a DVB / DAVIC broadcast network globally uniquely. Thus, the first part of the URL in a numerical form can be following:

dvb://<original_network_id>.<transport_stream_id>[.<service_id>[.<component_tag>][;<event_id>]][/<...>]

The <original_network_id>, <transport_stream_id>, <service_id>, <event_id> and <component_tag> are the values represented as hexadecimal strings without any “0x” prefix (for example, “4d2e”).

Depending on the service the following part of the URL (“/<...>“) may contain a more detailed specifier that identifies the desired part of the service. For example, if the service contains a data carousel, the last part of the URL may contain the file name.

## 9.6. Run-time execution environment

### 9.6.1 Application execution

The STU run-time environment shall include the following component:

- the MHEG-5 run-time engine, as defined by ISO/IEC 13522-5; the MHEG-5 run-time engine may be extended by supporting MHEG-5/Java VM interworking provisions as defined by ISO/IEC 13522-6.

In addition, the STU run-time environment may include the following components:

- implementation of the Java virtual machine, as specified by the *Java Virtual machine specification*;
- implementation of the core set of Java API packages as defined in Clause 9.4 of this Specification.

The MHEG-5 run-time engine shall fully support the MHEG-5 instantiation defined by this part of ISO/IEC 16500.

### 9.6.2 User Input Events

The use of InputEventRegisters is defined in Clause 9.2.6 of this Specification. Each register has a number, which is exchanged as one of the parameters of a Scene object. The contents of a UserInputEventRegister (which is not exchanged) is a set of numbers (representing UserInputEventTags) and a name. The name/number pairs bind a specific UserInputEventTag to a logical input event. It is the task of the engine implementor to bind the logical input event to one or more physical input events. Table 9-4 in Clause 9.2.6 specifies the mandatory user input events that have to be generated by the user device for ISO/IEC 16500 STUs.

### 9.6.3 IDL definition for RTE run remote call

MHEG-5 RemoteProgram objects invoke a stub at the client side. That stub then sends a DAVIC-defined message to the server, using the remote procedure call protocol defined in Part 7 of this Specification. The IDL definition of the stubs is as follows:

```
module DAV
{
      enum parType {boolPar, intPar, octStringPar, objRefPar, contRefPar};
      typedef sequence<octet> octString;
      typedef struct oR {
            octString groupIdentifier;
            long objectNumber;
      } objRef;
      typedef octString contRef;
      union par switch (parType)
      {
            case boolPar: boolean aBoolean;
            case intPar: long anInt;
            case octStringPar: octString aString;
            case objRefPar: objRef anObjRef;
            case contRefPar: contRef aContRef;
      };
      typedef sequence<par> pars;
      interface MHEG {
            void call(
                  in octString programName,
                  inout pars somePars
            );
            void fork(
                  in octString programName,
                  inout pars somePars
            );
      };
};
```



The mapping of the parameters of the MHEG-5 actions Call and Fork is intuitive. It is the responsibility of the MHEG engine to set the MHEG-5 parameters ForkSucceeded and CallSucceeded, respectively, to indicate the success or failure of the RPC operation.

### 9.6.4 Mapping of High-Level API Actions on DSM-CC Primitives

It is the responsability of the STU to map the actions in the high-level API to DSM-CC primitives. To provide some indication of how DSM-CC primitives can be used to implement an MHEG-5 runtime engine, table 9-16 provides an example of a possible DSM-CC mapping to the high level API.

**Table 9-16. — Examples of mapping of high-level API actions to DSM-CC primitives**

| *MHEG-5 behavior* | *MHEG-5 Object Type* | *DSM-CC U-U Fuction* |
|---|---|---|
| Launch/Spawn | Application | DirectoryOpen*(app.fileid) -> FileObRef*<br>FileRead*(FileObRef)* |
| Prepare | Scene, Content Object and Stream | DirectoryOpen*(scene.fileid) -> FileObRef*<br>FileRead *(FileObRef)* |
| Run | Video and Audio | DirectoryOpen*(stream.file) -> StreamObRef*<br>StreamResume(*StreamObRef, starttime, 1/1*) |
| Stop | Stream | StreamPause(*StreamObRef, x80000*) |
| StreamMarker | Stream | StreamSubscribe (*StreamObRef, marker*)<br>StreamNotify (*StreamObRef, marker, call back function*)<br>StreamUnSubscribe (*StreamObRef, marker)* |
| StreamTimer | Stream | StreamStatus -> Gets normal playtime |
| FreezeFrame | Stream | StreamPause(*StreamObRef, x80000*)<br>StreamStatus -> Gets stoptime<br>StreamResume(*StreamObRef, stoptime, 1/1*)<br>*Note: In linear broadcast mode, freeze frame is a client function only* |
| RunAsynchronous and RunSynchronous | Application | RPC - UNO |
| OpenConnection | Application | Attach (*ID)* |

# 10. DAVIC Reference Model for Contents Decoding

## 10.1. Scope

ISO/IEC 16500 defines a Normative Reference Decoding Model, RDM. The RDM specifies normative semantic constraint on DAVIC content. in particular for delivery, handling and decoding of contents. The RDM is not a guideline for the design of actual decoders; the RDM does not describe any specific STU architecture. The major problem addressed by the RDM is to ensure interoperability of the memory size and behaviour between content and STUs, and to do so without specifying the internal design and behaviour of the STU.

The Reference Decoder Model provides a virtual STU platform of which the performance is defined in mathematical terms. The RDM provides models for (1) data delivery, (2) memory usage for code and content objects and (3) timing for object handling and instruction execution. In addition the RDM provides rules and constraints for applications for avoiding application failures.

Each DAVIC compliant application is required to run on the RDM without any violation of the RDM requirements. In addition each DAVIC compliant STU is also required to meet the RDM requirements. Therefore the RDM is the DAVIC tool that ensures that each single DAVIC compliant application runs on all DAVIC compliant STUs without any application failure, and without porting the application to each specific STU implementation.

DAVIC application developers use the RDM as the virtual platform for application development and as the tool for verification of application correctness, without requiring any application testing at each specific STU implementation of the DAVIC specification. Developers of DAVIC compliant STUs use the RDM as the reference model for their implementation, thereby compensating for the differences between the theoretical performance of the RDM and the practical performance of their implementation; e.g. different timing characteristics may result in the need for compensation in buffer sizes. By serving as a platform and application independent interface between application and STU developers, the RDM provides a mechanism for independent development of applications and STUs.

## 10.2. Reference Decoder Model

The RDM is based on the MPEG-2 Systems (ISO/IEC 13818-1) Transport System Target Decoder (T-STD). The RDM incorporates the T-STD in its entirety, and adds additional virtual buffers and decoders to support graphics, text, linear audio and both real-time streams and stored objects. ISO/IEC 13818-1 (MPEG-2 Systems) contains the complete specification of the T-STD. The RDM specifies completely the times at which all objects enter and leave each of the various buffers, in terms of the timing model provided by MPEG Systems.

Within the RDM applications are executed by the run-time engine. The RDM provides three storage elements for the execution of applications; these storage elements are defined from application perspective, in a way independent of the implementation of the run-time environment. The three elements are :

1. Buffer Bcontents to store coded objects containing raw contents for use by applications.
2. Buffer Bcode to store MHEG-5 objects in the interchange format; however, included contents are not stored in Bcode, but in Bcontents instead. In addition, JAVA VM byte code is stored in Bcode in the interchange format.
3. Buffer Bexecute which is the dynamic memory available for run-time execution of the application (stack, heap, scratch-pad, etc.)

In addition to these three storage elements, the RDM adds the following to the T-STD:

4. Run-time engine, which executes the code contained in Bcode, using Bexecute as dynamic memory for application execution. The run-time engine controls the time at which contents are decoded and presented and controls the time at which contents are removed from Bcontents. By executing appropriate application code, the run-time engine can control the presentation processor.
5. Graphics/text decoder capable to decode stored objects from Bcontent and real-time graphic streams.
6. Linear audio decoder capable to decode linear audio objects from the object memory.
7. Presentation processor capable to perform operations on the audio and video outputs of the various decoders as required by the application, under control by application code executed in the run-time engine. Examples of such operations are an audio mix and a video wipe.
8. Transport buffers TBn to deliver real-time graphic streams to buffer Bn for graphics and to deliver objects to either buffer Bcontents or buffer Bcode.
9. Data paths to carry data as needed between buffers and decoders.



The RDM is shown in the following figure.

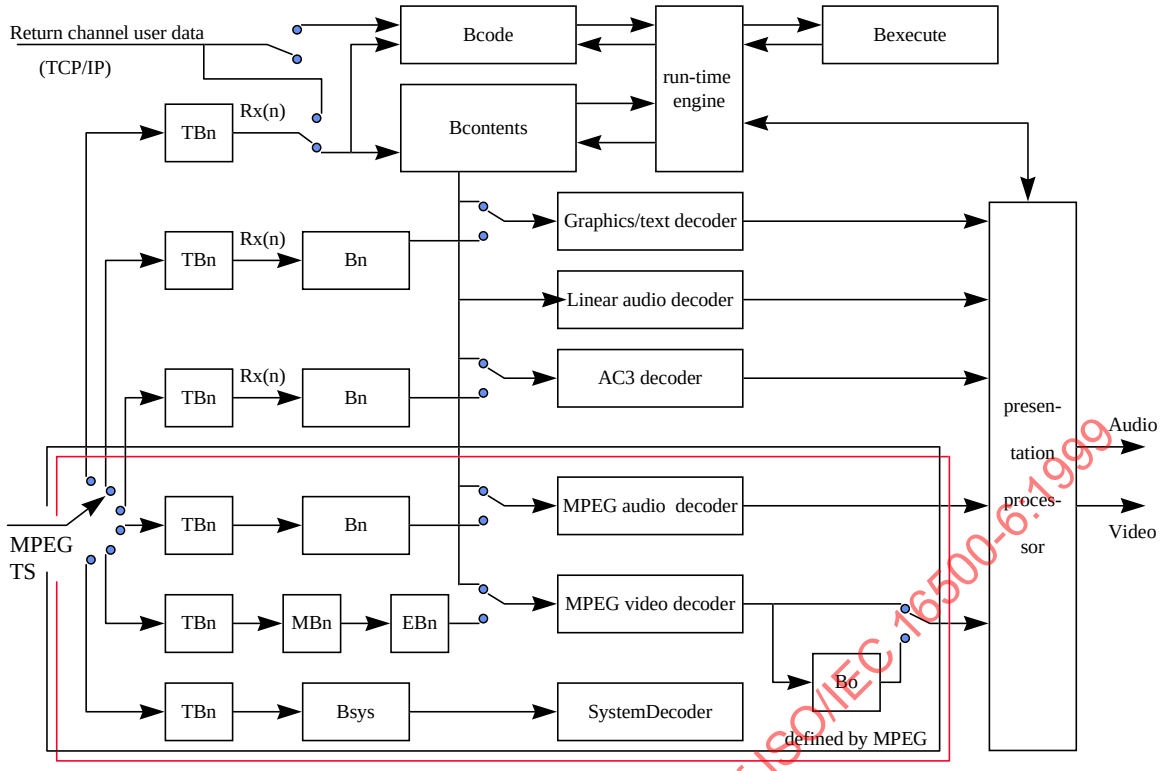



**Figure 10-1. — The DAVIC Reference Decoder Model**

Data enters the RDM via an MPEG-2 Transport Stream or as user data via a return channel. A real-time stream can only be delivered via the MPEG-2 Transport Stream, while for delivery of stored objects both delivery mechanisms can be exploited.When delivered over an MPEG-2 TS, stored objects as well as application code are contained in a DSM-CC User to User Object Carousel carried within MPEG-2 sections. Service Information is delivered while meeting the constraints specified in ETS 300 468. For the delivery and decoding of real-time streams carrying compressed graphics the decoder model and constraints defined in ETS 300 743 apply.

Contents are contained in memory in its coded representation. Contents either contained in a stored objects or in a real-time stream consists of a sequence of one or more entire Access Units. In the case of compressed video, still pictures and compressed audio Access Units consist of one picture or one audio frame respectively, as defined in ISO/IEC 11172 and ISO/IEC 13818. In the case of linear audio, an Access Unit equals one mono audio sample or one pair of stereo audio samples. In the case of AC-3, an audio Access Unit consists of one audio frame as defined in ATSC A/52, Annex A. The size of buffer Bn for AC-3 audio is equal to the size of Bn for MPEG audio (which is specified in ISO/IEC 13818-1). In the case of graphics and text, an Access Unit consists of an entire graphical or text object respectively.

At decode time, as indicated by a PTS or DTS, Access Units are decoded instantaneously. In the case of a real-time stream each Access Unit is removed from buffer Bn instantaneously at decode time. In the case of Access Units stored in Bcontents or Bcode, removal of the objects is controlled by the Run Time Engine, RTE. When the RTE destroys an object, the object is removed instantaneously from the buffer in which it is stored. All RTE processing, including code execution and object handling are instantaneous.

The size of buffer Bn and the leak rate of Rx(n) for graphics depends on the maximum number of colours and the maximum full screen resolution applied in an application at any time. These values are mandatorily provided by each compliant DAVIC application. In ISO/IEC 16500 the following options are allowed :

| | | |
|---|---|---|
| maximum number of colours at any time : | 4 colours, | (2 bits per pixel) |
| | 16 colours, | (4 bits per pixel) |
| | 256 colours or | (8 bits per pixel) |
| | 65536 colours. | (16 bits per pixel) |

maximum full screen resolution at any time :360 pixels by 288 lines (CIF)

720 pixels by 288 lines (601 hor, CIF vert)

720 pixels by 576 lines (601 hor and vert)

1280 pixels by 720 lines

1920 pixels by 1080 lines

Note that the maximum number of colours may be defined in multiple ways, e.g. by one CLUT table with a number of entries equal to the maximum number of colours, or by multiple CLUT tables with less entries, that use in total no more than the maximum number of colours.

Based on these options, 14 different colour and resolution combinations can be made. These combinations can be classified in multiple quality levels, using the required number of bits to store a full screen picture as qualifier. This results in 8 different quality levels, with one, two or three colour / resolution combinations with the same value of the qualifier per quality level. See Table 10-2.

**Table 10-2. — Quality Levels**

| *quality level* | *maximum horizontal resolution at any time* | *maximum vertical resolution at any time* | *maximum number of colours at any time* |
|---|---|---|---|
| 1 | CIF | CIF | 4 |
| 2 | CIF | CIF | 16 |
| | 601 | CIF | 4 |
| 3 | CIF | CIF | 256 |
| | 601 | CIF | 16 |
| | 601 | 601 | 4 |
| 4 | CIF | CIF | 65536 |
| | 601 | CIF | 256 |
| | 601 | 601 | 16 |
| 5 | 601 | CIF | 65536 |
| | 601 | 601 | 256 |
| 6 | 601 | 601 | 65536 |
| 7 | 1280 | 720 | 65536 |
| 8 | 1920 | 1080 | 65536 |



For graphics the size of Bn and the value of the leak rate Rx(n) are defined per quality level. Note that these figures apply to a real-time graphics stream. The buffer size corresponds to a storage capacity equal to about 1.2 times one full screen uncompressed picture. The rates correspond to the constant delivery rate needed to fully load an empty Bn buffer in 2.5 seconds. See Table 10-3.

**Table 10-3. — Size of Bn and value of Rx(n) for graphics per Quality Level**

| *quality level* | *size of buffer Bn for graphics (x 1 000 000 bits)* | *leakrate Rx(n) for graphics (x 1 000 000 bits/sec)* |
|---|---|---|
| 1 | 0.25 | 0.1 |
| 2 | 0.5 | 0.2 |
| 3 | 1.0 | 0.4 |
| 4 | 2.0 | 0.8 |

| | | |
|---|---|---|
| 5 | 4.0 | 1.6 |
| 6 | 8.0 | 3.2 |
| 7 | 17.0 | 7.1 |
| 8 | 40.0 | 16.0 |

The TB buffers operate exactly as in the T-STD. They are 512 bytes in size; they receive data from the Transport stream on the S1 interface at the schedule encoded in the transport stream; data leaves them and enters Bn at rates specifies according to the elementary stream type. The leak rate of TB for graphics and text is defined in Table 10-3.

Buffer Bcontents contains stored objects in their coded representation. The ISO/IEC 16500 coded objects which are accepted and stored in Bcontents are:

1. Graphics/text coded objects
2. Linear audio coded objects
3. MPEG video coded objects, including still pictures
4. MPEG audio coded objects.
5. AC-3 audio coded objects

Bcode contains MHEG-5 objects in interchange format (without included contents) and JAVA VM byte code.

The minimum size of buffers Bcontents and Bcode required to run an application are mandatorily provided by each compliant DAVIC application. In addition the application mandatorily provides the value of the leak rate Rx(n) applied by the application. Two options are available : 1 000 000 and 5 000 000 bits per second.

The RDM constrains all contents and its delivery such that when the contents its delivery is verified using the RDM the buffers shall not overflow. The RDM constraints include the constraints specified by the MPEG-2 Systems standard.

## 10.3. DAVIC Application Resource Descriptor

To inform the STU about the minimum requirements the run-time environment needs to support in order to successfully execute a specific application, the DAVIC application resource descriptor is defined, providing information about required memory resources, the resolution and number of colours applied in the application, and which leak rate Rx(n) is used for transfer of data from TB(n) to Bcontents and Bcode. Each compliant DAVIC application is mandatorily required to provide the DAVIC application resource descriptor. In future DAVIC may extend the descriptor by adding trailing bytes to the descriptor. The descriptor is carried by the MHEG-5 application. The required resources shall be specified for each application and may be specified for each scene within that application. The DAVIC application resource descriptor for an application or a scene is carried within the ObjectInformation attribute of the application or scene.

**Table 10-4. — DAVIC application resource descriptor**

| *Syntax* | *No of bits* | *Mnemonic* |
|---|---|---|
| DAVIC_application_descriptor(){ | | |
| Bcontents_size | 16 | uimsbf |
| Bcode_size | 16 | uimsbf |
| Bexecute_size | 16 | uimsbf |
| maximum_graphics_resolution | 4 | bslbf |
| maximum_number_colours_in_graphics | 4 | bslbf |
| leak_rate_Rx(n)_for_objects | 4 | bslbf |
| reserved | 4 | bslbf |
| maximum_MPEG_video_coded_picture_size | 16 | uimsbf |



```
}
```

**Bcontents_size** : a 16 bit unsigned integer specifying the required size of Bcontents for this DAVIC application. The size is specified in units of 16384 Bytes.

**Bcode_size** : a 16 bit unsigned integer specifying the required size of Bcode for this DAVIC application. The size is specified in units of 16384 Bytes.

**Bexecute_size** : a 16 bit unsigned integer specifying the required size of Bexecute for this DAVIC application. The size is specified in units of 16384 Bytes.

**maximum_graphics_resolution** : a 4 bit field that indicates the maximum full screen resolution used at any time during this entire application, corresponding to the following table.

**Table 10-5. — Maximum full screen resolution applied in application.**

| *Value* | *Meaning* |
|---|---|
| '0000' | reserved |
| '0001' | 360 pixels by 288 lines (CIF) |
| '0010' | 720 pixels by 288 lines |
| '0011' | 720 pixels by 576 lines (ITU-R601) |
| '0100' | 1280 pixels by 720 lines |
| '0110' | 1920 pixels by 1080 lines |
| '0111' - '1111' | reserved |

**maximum_number_colours_in_graphics :** a 4 bit field that indicates the maximum number of different colours used throughout this entire application, corresponding to the following table.

**Table 10-6. Maximum number of colours in application**

| *Value* | *Meaning* |
|---|---|
| '0000' | 4 |
| '0001' | 16 |
| '0010' | 256 |
| '0011' | 65536 |
| '0100' - '1111' | reserved |

**leak_rate_Rx(n)_for_objects** : a 4 bit field that specifies the leak rate Rx(n) applied in this application to transfer objects to either Bcontents or Bcode, corresponding to the following table.

**Table 10-7. — Leak rate Rx(n) for objects**

| *Value* | *Meaning* |
|---|---|
| '0000' | reserved |
| '0001' | 1 000 000 bits per second |
| '0010' | 5 000 000 bits per second |
| '0011' - '1111' | reserved |

**maximum_MPEG_video_coded_picture_size** : a 16 bit field that specifies the maximum size of MPEG Video pictures used at any time in this application in units of macroblocks (256 pixels, equivalent to 16 pixels by 16 lines). A value of zero indicates that no MPEG Video is used in this application. Note that in that case MPEG Still Pictures may be used.



## 10.4. Minimum ISO/IEC 16500 STU requirements

Each compliant ISO/IEC 16500 STU is mandatorily required to implement Bcontents, Bcode and Bexecute of a size that is at least equal to the value specified in table 10-8. Note that these sizes are defined in terms of the RDM. Practical implementations of these buffers may have different sizes, while meeting the minimum requirements defined in Table 10-8.

**Table 10-8. Minimum buffer sizes required in ISO/IEC 16500 STUs**

| | *Minimum size* |
|---|---|
| Bcode | 128 kByte (= 131 072 Bytes) |
| Bcontents | 512 kByte (= 524 288 Bytes |
| Bexecute | 128 kByte (= 131 072 Bytes) |

## 10.5. Support for Graphics in STUs

ISO/IEC 16500 STUs shall support graphics in relation to MPEG Video as specified in table 10-9. When MPEG Video is not used in an application, or when MPEG Still Picture is used in an application, then full screen graphics shall be supported at each graphics quality level. Note that MPEG Video and MPEG Still Pictures are mutually exclusive functions within ISO/IEC 16500.

**Table 10-9. — Minimum requirement for the support of graphics in ISO/IEC 16500 STUs**

| *graphics quality level* | *graphic picture size* | *MPEG Video coded picture size* |
|---|---|---|
| 1-3 | full screen | full screen ITU-R 601 / 50 Hz |
| 4 | <= 0.6 * full screen | (720*576 pixels by lines, |
| 5 | <= 0.3 * full screen | equivalent to 1620 macroblocks) |
| 6 | <= 0.15 * full screen | |
| 1-4 | full screen | full screen ITU-R 601 / 59.94 Hz |
| 5 | <= 0.6 * full screen | (720*480 pixels by lines, |
| 6 | <= 0.3 * full screen | equivalent to 1350 macroblocks) |
| 4 | full screen | <= 1440 macroblocks |
| 5 | full screen | <= 1080 macroblocks |
| 6 | full screen | <= 312 macroblocks |
| 7 | full screen | full screen (1280*720 pixels by lines, progressive) |
| 8 | full screen | full screen (1920*1080 pixels by lines, interlace) |



## 10.6. Persistent Memory

Persistent memory is not explicitly defined in the RDM. ISO/IEC 16500 STUs may contain persistent memory, as an optional resource for the convenience of applications. However, the availability of persistent memory is not a requirement for ISO/IEC 16500 STUs. For ISO/IEC 16500 applications the availability of persistent memory shall not be a condition for successful execution of the application.

# 11. Content Packaging and Metadata

This subclause describes how media content is packaged for delivery from a Content Provider to a Service Provider over the A10 interface. In addition, it describes the ancillary data (metadata) which relates to media content.

Other aspects of the content loading process are described in documents. Specifically:

ISO/IEC TR 16501 describes the basic functionalities required by the content loading process.

DAVIC 1.3.1a Part 3 describes the Content Transfer A10 Architecture from the perspective of the Service Provider System (SPS).

ISO/IEC 16500-5 describes the mid-layer protocols/APIs for content loading.

The way in which content is packaged for delivery is independent of the from the way in which media content data is delivered to the SPS (it may be delivered to a Service Provider either on physical media or over a transmission system).

All programming content is represented in the DAVIC system as multimedia components. Multimedia components comprise one or more monomedia components coupled with the logical relationships between the monomedia components. The multimedia components will be created by content providers for input to the servers. The multimedia components will be accessed by the STU's using the delivery system.

## 11.1. Content package structure

The scope of the content package structure is to support content transfer across the A10 interface, and content handling at the content provider system side and at the service provider system side. Among the important objectives for such a structure are (no priority in the listing):

- Random access to individual content Item Elements, e.g. for transfer, updating, deletion and adding.
- Extensibility of elements, i.e. adding new attributes must be possible with backward compatibility.
- Global referencing possibilities in and out of packages.

### 11.1.1 Content Item Elements

A content Item Element (CIE) is the smallest (and indivisible) content component. A CIE can be part of one or more content items. Examples of such elements are still pictures and video clips.

### 11.1.2 Content Item

A content item (CI) is a collection of content Item Elements that will form a complete application or a complete programme. Different content items may share one or more content Item Elements. A content item could contain another content item. Examples of content items are movies and home shopping applications.



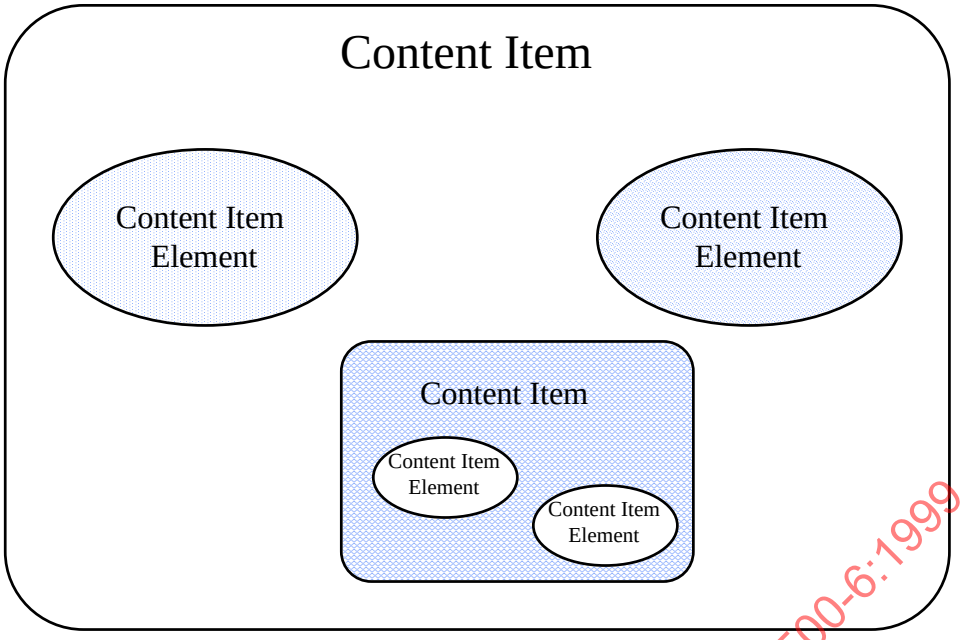



**Figure 11-1. — Illustration of a Content Item**

### 11.1.3 Content package

A content package is a set of content Item Elements and/or content items. Examples of use is delivery and intermediate storage.

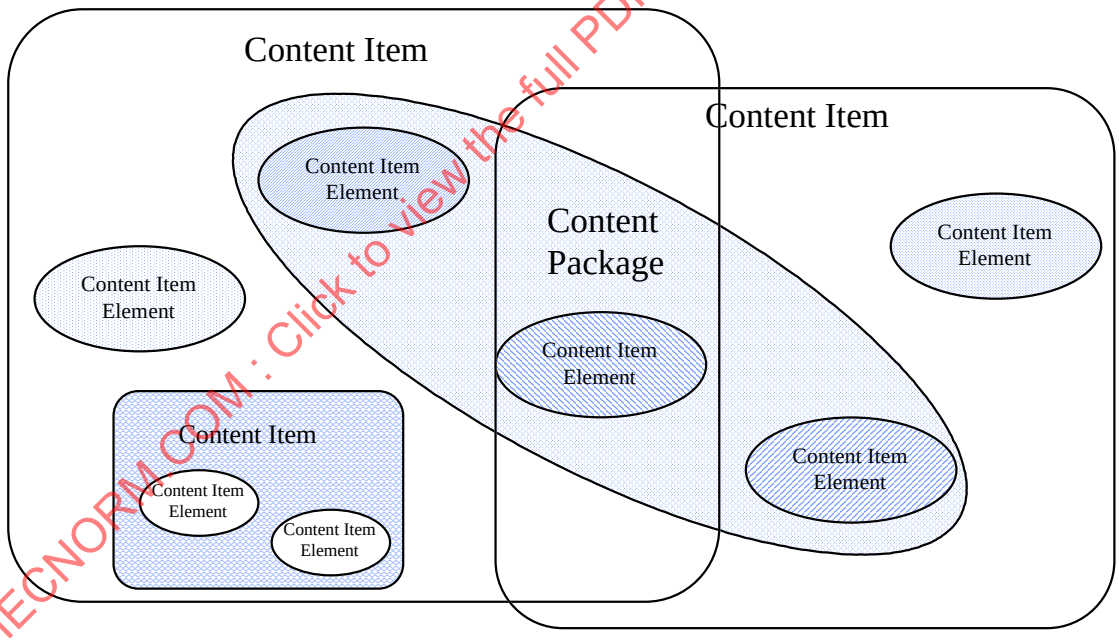



**Figure 11-2. — The relationships between the items, Item Elements and packages.**

As an example, a movie show would typically contain the featured movie, a sequence of trailers and a number of advertisements. Using the DAVIC content package structure the move show, the trailer sequence and the advertisement could be content items. Each trailer and each advertisement could be a content Item Element. The trailers could be assembled from a package of a large number of trailers delivered by some film company.

## 11.2. Content Metadata

Content items transferred by a Content Provider to a Content Server consist of both media content elements (video, audio, stills, etc.) and associated descriptive and control elements–refered to here as “content metadata”. Note that these descriptive elements can be (pointers to) media elements.

We divide content metadata into two categories: Content Management and Navigational. Content management refers to metadata that allows the content data to be managed within an SPS. Content management metadata is used for content transfer and loading, version control, and content verification. Navigational metadata is used to enable use of the server content, as controlled by the client and server applications. This may include navigation, embargo and deletion control, copyright management, and user access control.

A core set of metadata items are required to be provided with each content item. This small core set provides for the basic management of server items.

### 11.2.1 Types

The metadata types listed here are given as guidelines to express the general format of the information. Types will be more specifically defined for the content packaging tool chosen to encode content packages.

| | |
|---|---|
| ISO 639-2 | 24-bit language code, as defined in ISO 639 Part 2 (/B and /T can be used), with each character coded into 8 bits according to ISO 8859-1 (see, for example, explanation in ETS 300 468 subclause 6.2.3) |
| bmp/ISO 10646-1 | Coded byte pairs for international character representation, as per the Basic Multilingual Plane of ISO 10646-1 (see, for example, the European subset informatively stated in ETSI ETS 300 468 Annex A) |
| UTC/MJD | Global date/time: Modified Julian Day/Coordinated Universal Time (see ETSI ETS 300 468 Annex C) |
| UTC/rel | Time period, 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD), for example 12h 45m 01s is coded as “124501”. |
| char_sequence | Set of characters, coded to ISO 8859-1. |
| content_type | Restricted set of char_sequence (see above) to define the type of the CIE, the defined set being: {preview, main, logo, synopsis, poster_image} |
| country_code | 24-bit field identifying a country using the 3-character code as specified in ISO 3166. Each character is coded into 8-bits according to ISO 8859-1 and inserted into the 24-bit field. In the case that the 3 characters represent a number in the range 900 to 999, then country_code specifies an ETSI-defined group of countries. These allocations are found in ETR 162 |
| dimension_name | char_sequence (see above) defining dimension of content rating being considered, specific for the given region defined, for example violence or language rating for USA |
| text_rating | set of bmp/ISO 10646-1 symbols defining the actual rating given for the defined rating dimension for the defined country |
| rating_body | set of bmp/ISO 10646-1 symbols defining the recognised body within the given country that awarded the content item the given age_rating or text_rating |
| role_char | restricted, but extensible, char_sequence (see above) defining the role of the person or entity which will be stated in the role-description. The initial set to be defined as {director, actor, movie_house, producer, composer, author, book_title, musician, production_studio, location} |
| telephone_descriptor | a multi-part type, containing country_prefix_char, international_area_code_char, national_area_code_char, core_number_char: |
| country_prefix_char | a set of 8-bit characters, coded in accordance with ISO 8859-1, the set of alphanumeric characters which make up the country prefix. |
| international_area_code_char | a set of 8-bit characters, coded in accordance with ISO 8859-1, the set of alphanumeric characters which make up the international area code. |



| | |
|---|---|
| national_area_code_char | a set of 8-bit characters, coded in accordance with ISO 8859-1, the set of alphanumeric characters which make up the national area code. |
| core_number_char | a set of 8-bit characters, coded in accordance with ISO 8859-1, the set of alphanumeric characters which make up the core number. |
| age_rating | 8-bit integer of recommended minimum age in years of end user, where minimum age = rating + 3 years (e.g., 04 implies that end users should be at least 7 years old), with a max value 0F. |
| content_descriptor | single byte, consisting of two nibbles, as defined in ETS 300 468 subclause 6.2.4 Table 18, providing classification information for the content. |
| bslbf | bit sequence, left bit first |
| fpvsbf | floating point, value sign bit first |
| uimsbf | unsigned integer, most significant bit first |

## 11.2.2 Semantics of the Loading Commands

The content loader in the SPS executes the loading commands attached to a content element. The following loading commands are available.

### 11.2.2.1 DSM-CC Directory Primitives

Directory primitives, as described in ISO/IEC 13818-6, allow the content Item Elements to be manipulated on loading as follows:

**Figure 11-3. — DSM-CC Directory Primitives**

| *Inherited from **NamingContext**:* | |
|---|---|
| list | Return a list of all the bindings to object references in the context. |
| resolve | Return the object reference bound to a given name. |
| bind | Bind an object reference to a name. |
| bind_context | Bind a naming context to a name. |
| rebind | Bind an object reference to a name, overwriting any previous binding. |
| rebind_context | Bind a context to a name, overwriting any previous binding. |
| unbind | Remove a binding for a name. |
| new_context | Create a new naming context. |
| bind_new_context | Create a new naming context and bind it to the given name. |
| destroy | Destroy the naming context. |
| *Defined in Directory:* | |
| DSM Directory open | Resolve the objects associated with names in the given path. |
| DSM Directory close | Close a reference to a Directory. |
| DSM Directory get | Return the values bound to names in a given path. |
| DSM Directory put | Bind names in a given path to values, overwriting any previous bindings. |



### 11.2.2.2 Execute

The content element gets executed. This is especially suited for “setup” procedures.

#### 11.2.2.3 Priority

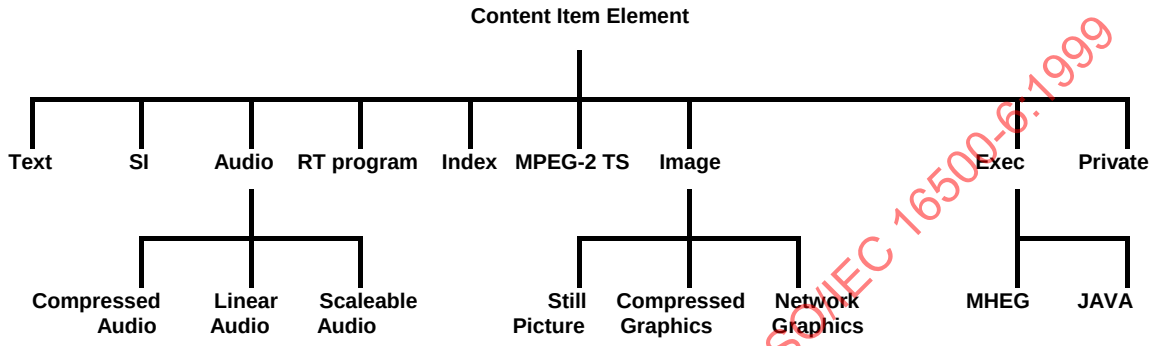
CIE loading instructions take precedence over CI loading instructions if they differ.

### 11.2.3 Metadata Mapping

The lists and definitions given below are not exhaustive. However, it is mandatory that if a type is used, that the definitions given are followed. The navigational data defined is inevitably aimed at the most open applications, such as movies or TV programmes on demand, but are not exclusive to those applications.

The following figure shows the inheritance of monomedia components from the Content Item Elements for metadata attributes. The monomedia components shown are as defined in this part of ISO/IEC 16500. The monomedia components do not inherit attributes from Content packages or Content Items.

**Figure 11-4. — Inheritance from Content Item Element**






In the descriptions below the following assumptions apply:

- All names are COS Naming Service compound names, based on the binding of an object to a name in a name space. This implies that the CIE and CI names may change upon loading, based on the binding of the referenced objects to a new naming context when loading.
- It is possible to add new attributes for a CIE while on SPS, but it isn't allowed change any existing attributes value except for the CIE_path and CIE_assoc_CIlist, where one is allowed to add associations to other CI's.
- Changes and additions to the CI attributes while on the SPS are allowed in all attributes except for the CI_path, CI_guid, CI_owner, CI_ permission, and the CI_title.
- Optional attributes for the CIE and the CI means that it may not always be used , not that it may not be supported in a tool. All metadata marked as optional and mandatory must be supported by the tools.
- Private attributes may be added to any Content package, Content Item, Content **Item Element**, or Monomedia Component. Definitions for such attributes are outside the scope of this specification.

#### 11.2.3.1 Content package Metadata

| *Attribute* | *Type* | *Description* | *M/O*[7] |
|---|---|---|---|
| CP_name | CosNaming::name | COS Naming Service compound name | M |
| CP_path | char_sequence | Physical location of the CP metadata | M |
| CP_TOC | set of {{CIE_guid, CIE_name}, {CI_guid, CI_name}} | List of CIE names and IDs, and CI names and IDs, in the package | M |

[7] Mandatory/Optional

#### 11.2.3.2 Content Item Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| CI_ permission | TBD | Permissions for use that apply to the CI; different users will have different permissions | M |
| CI_guid | char_sequence | Global unique identifier of the Content Item | M |
| CI_name | CosNaming::name | COS Naming Service compound name | M |
| CI_owner | char_sequence | ID of the owner of the content item (Content Provider or SP) | M |
| CI_path | char_sequence | Physical location of the CI metadata | M |
| CI_title | char_sequence | Title of the CI | M |
| CI_TOC | set of {{CIE_guid, CIE_name}, CI_guid, CI_name}} | List of CIE names and IDs and CI names and IDs in the CI | M |
| CI_version | bslbf | Version of the CI (major and minor) | M |
| Award.date_received | UTC/MJD | Date awarded | O |
| Award.description | char_sequence | Award description | O |
| Award.name | bmp/ISO 10646-1 | Award name | O |
| CI_abbr_title | char_sequence | Abbreviated CI title | O |
| CI_date_available | UTC/MJD | Date available to customers | O |
| CI_delete_date | UTC/MJD | Date to delete from the SPS | O |
| CI_episode_name | bmp/ISO 10646-1 | Particular episode name | O |
| CI_expiration_date | UTC/MJD | Date content became unavailable to customers | O |
| CI_price_unit.price | TBD/fpvsbf | Price of content package from the CP to the SP, in appropriate units | O |
| CI_price_unit.user_price | TBD/fpvsbfFixed.2 | CP suggested price of content package from the SP to the user, in appropriate units | O |
| CI_Provider.address | bmp/ISO 10646-1 | Address of content provider | O |
| CI_Provider.contact_name | bmp/ISO 10646-1 | Contact name | O |
| CI_Provider.contact_phone | telephone_descriptor | Contact phone number | O |
| CI_Provider.ID[8] | char_sequence | Global unique ID for the Content Item provider | O |
| CI_Provider.name | bmp/ISO 10646-1 | Name of content provider | O |
| CI_provider_ref | char_sequence | Local ID of the CI to the Service Provider | O |
| CI_Restriction.Country.Dimension.Rating | dimension_name.text_rating | Rating in given country according to a given system (or dimension) by agreed text (e.g. in USA, Warnings, Language, or UK, BBFC, Uc) *three-dimensional* | O |
| CI_Restriction.Country.Rating_Body.Overall_Age_Rating | country_code.rating_body.age_rating | Rating in given country by minimum age by given rating body (e.g., 15 years in UK | O |

[8] CI_Provider.ID will likely not be used at the same time as the other CI_Provider attributes.

| | | | |
|---|---|---|---|
| | | by BVA) *three-dimensional* | |
| Review.critic_name | bmp/ISO 10646-1 | Name of critic | O |
| Review.description | bmp/ISO 10646-1 | Description of review | O |
| Review.language | ISO 639.2 | Language in which review is written | O |
| Review.org | bmp/ISO 10646-1 | Organisation that gave the review | O |
| Review.quote | bmp/ISO 10646-1 | Quote out of review | O |
| Review.rating | char_sequence | Rating on a specific scale (1-4 stars,...) | O |
| Review.text | bmp/ISO 10646-1 | Text of review | O |
| Role.description | bmp/ISO 10646-1 | Description or name of person/people taking that role | O |
| Role.name | role_char | Name of role | O |
| Schedule.channelno | char_sequence | logical channel number | O |
| Schedule.length | UTC/rel | length of program | O |
| Schedule.startdate | UTC/MJD | Starting date of program | O |
| Schedule.starttime | UTC/MJD | Starting time of program | O |

#### 11.2.3.3 Content Item Element Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| CIE_ permission | TBD | Permissions for use that apply to the CIE; different users will have different permission | M |
| CIE_assoc_CI | set of char_sequence | Associated applications | M |
| CIE_comp | enumeration of { RT_program<br>Text<br>Index<br>SI<br>Compressed Audio<br>Linear Audio<br>Compressed Video<br>Still Picture<br>Graphics<br>Java, MHEG-5<br>Private } | Content Item Element monomedia component type | M |
| CIE_guid | char_sequence | Global unique identifier | M |
| CIE_loading | set of {DSM Directory, execute} | List of loading commands to be executed at loading time. | M |
| CIE_name | CosNaming::name | COS Naming Service compound name | M |
| CIE_owner | char_sequence | Global unique identifier of legal owner | M |
| CIE_path | char_sequence, char_sequence | Physical location of the CIE metadata and monomedia component data | M |



| | | | |
|---|---|---|---|
| CIE_size | uimsbf | size in bytes of the content part of the element | M |
| CIE_version | bslbf | version number (major, minor) | M |
| CIE_virus_checked | UTC/MJD | Date and Time the CIE was last virus checked | M |
| CIE _prov_ref | char_sequence | Local ID of the CI to the Service Provider | O |
| CIE_index_start | DSM-CC_appNPT | Index information to reference specific frames of the content (assumes MPEG-2 TS) | O |
| CIE_index_stop | DSM-CC_appNPT | Index information to reference specific frames of the content (assumes MPEG-2 TS) | O |
| CIE_mult_lang | list of lang | Languages available in the CIE | O |
| CIE_Provider.address | bmp/ISO 10646-1 | Address of content provider | O |
| CIE_Provider.contact_name | bmp/ISO 10646-1 | Contact name | O |
| CIE_Provider.contact_phone | telephone_descriptor | Contact phone number | O |
| CIE_Provider.ID[9] | char_sequence | Global unique ID for the Content Item Element provider | O |
| CIE_segment_table[10] | Ordered list of {CIE_name} | List of the CIE segments for a multi-file CIE, in order. | O |
| CIE_type | content _type | Type of content unit (preview, main, logo) | O |
| EncoderProvider.address | bmp/ISO 10646-1 | Address of encoder provider | O |
| EncoderProvider.contact_name | bmp/ISO 10646-1 | Contact name | O |
| EncoderProvider.contact_phone | telephone_descriptor | Contact phone number | O |
| EncoderProvider.name | bmp/ISO 10646-1 | Name of encoder provider | O |

#### 11.2.3.4 Real Time Program Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| RT_prog_comp | list of CIE_name | List of CIEs pointed to by the RT Program (MPEG-2 TS and Index only) | M |

#### 11.2.3.5 MPEG-2 TS Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| bitrate | uimsbf | bit rate in bits per second | M |
| aspect_ratio | bslbf | aspect ratio of the intended display | O |
| frame_rate | fpvsbf | video frame rate in Hz | O |
| horizontal_size | uimsbf | Number of pixels of the frame in the horizontal direction | O |

[9] CIE_Provider.ID will likely not be used at the same time as the other CIE_Provider attributes.

[10] The CIE_segment_table attribute need to be included in the CIE metadata for each segment to which it applies.



| | | | |
|---|---|---|---|
| level_profile | bslbf | level and profile used for video encoding, as specified by ISO/IEC 13818-2 | O |
| mult_lang_audio | set of ISO 639.2 | list of one or more language identifiers for the audio tracks in the TS | O |
| mult_lang_text | set of ISO 639.2 | list of one or more language identifiers for the text tracks (e.g., subtitles) in the TS | O |
| PES_comp | setof {CIE_comp, uimsbf) | The component monomedia that has been mapped to the TS via the PES structure, and location (PID) | O |
| ratio | uimsbf | Amount of speed up for the Fast forward or Fast Rewind streams | O |
| TS_comp | set of {CIE_comp, uimsbf) | The component monomedia that has been mapped to the TS via the TS private section, and location (PID) | O |
| TS_mode | bslbf | Indication of whether the TS is the standard, the fast forward or the fast rewind stream | O |
| vertical_size | uimsbf | Number of pixels of the frame in the vertical direction | O |
| video_comp_type | enumeration of {MPEG-1, MPEG-2} | Type of video encoding | O |

#### 11.2.3.6 Index Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| Index_mode | bslbf | Indication of whether the Index file applies to the standard, the fast forward or the fast rewind stream | O |

#### 11.2.3.7 Text Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| starting corner | uimsbf, uimsbf | location (in pixels) of the starting corner on the screen in terms of (# horizontal pixels, # vertical pixels), and where (0,0) is the upper left-hand corner. | O |
| text_flow | bslbf | Direction of the text flow (l-r, or r-l) | O |

#### 11.2.3.8 System Information Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| SI_tables | char_sequence | The type of DVB SI table | M |

#### 11.2.3.9 Compressed Audio Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| bitrate | uimsbf | The playback bitrate of the audio stream | M |
| audio_type | bslbf | The type of audio encoding - MPEG-1 Layer I, MPEG-1 Layer II, AC-3, or | O |



| | | | |
|---|---|---|---|
| | | MPEG-2 Layer II. | |
| channel_format | enumeration of {sing_chan, dual_chan, joint_stereo, stereo} | The type of channels included in the audio stream | O |
| sample_rate | fpvsbf | The sample frequency of the encoded data | O |
| sample_size | uimsbf | the number of bits per audio sample | O |

#### 11.2.3.10 Linear Audio Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| num_channels | uimsbf | Whether one (mono) or two (stereo) channels are included in the audio stream | O |
| num_sample_frames | uimsbf | Number of sample frames | O |
| sample_rate | fpvsbf | The sample frequency of the encoded data | O |
| sample_size | uimsbf | the number of bits per audio sample | O |

#### 11.2.3.11 Still Picture Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| aspect_ratio | bslbf | aspect ratio of the intended display | O |
| horizontal_size | uimsbf | Number of pixels of the frame in the horizontal direction | O |
| level_profile | bslbf | level and profile used for video encoding, as specified by ISO/IEC 13818-2 | O |
| vertical_size | uimsbf | Number of pixels of the frame in the vertical direction | O |

#### 11.2.3.12 Graphics Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| coding_type | bslbf | Coding type for still graphic - RGB16, CLUT8, CLUT4, or CLUT2 | O |
| dithering_pref | bslbf | preference of whether or not to dither | O |
| horizontal_position | uimsbf | Horizontal offset between the top left corner of the object and the top left corner of the grid | O |
| horizontal_size | uimsbf | Number of pixels of the frame in the horizontal direction | O |
| object_scalability | bslbf | describes minimum CLUT format for the object on display | O |
| pixel_aspect_ratio | bslbf | aspect ration of a pixel | O |
| resolution_ind | bslbf | resolution of the graphical object based on screen size - half frame resolution in both horizontal and vertical, vertical only, horizontal only | O |
| scaling_pref | bslbf | preference for scaling if object ratio does | O |

| | | | |
|---|---|---|---|
| | | not match display ratio | |
| target_background_grid | bslbf | Target background grid | |
| vertical_position | uimsbf | Vertical offset between the top left corner of the object and the top left corner of the grid | O |
| vertical_size | uimsbf | Number of pixels of the frame in the vertical direction | O |

#### 11.2.3.13 Java Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| client_server | bslbf | Indication of whether the Java executable represents client or server code. | M |

#### 11.2.3.14 MHEG-5 Metadata

None

#### 11.2.3.15 Private Metadata

| *Attribute* | *Type* | *Description* | *M/O* |
|---|---|---|---|
| private_data | unspecified | User-defined data | O |

## 11.3. Content Packaging Format

### 11.3.1 Bento

The Bento data model fits very well to the object model of DAVIC. We shall choose Bento as a base file format for content packaging, and describe the definition of DAVIC Objects in this chapter.

### 11.3.2 Bento definition of DAVIC Objects

Bento allows for extension through the definition of new objects, properties, and data types. The DAVIC Objects could be newly defined. Each DAVIC object should have a globally unique ID. The attributes specified in subclause 11.2 will map onto Bento properties. Additional properties may be added by the content loading tool set and will be specified at an appropriate time. Using Bento representation, the DAVIC Objects could be defined as below.

#### 11.3.2.1 Content package Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Content package Object |
| CPID | ObjRef | A globally unique ID and Version number |
| CP_name | CosNaming::name | COS Naming Service compound name |
| CP_path | char_sequence | Physical location of the CP metadata |
| CP_TOC | set of {{CIE_guid, CIE_name}, | List of CIE names and IDs, and CI names and IDs, in the package |



| | | |
|---|---|---|
| | {CI_guid, CI_name}} | |

#### 11.3.2.2 Content Item Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Content Item Object |
| CPID | ObjRef | A globally unique ID indicating the Content Provider |
| PackageID | String | An ASCII character string, unique within the domain of the Content Provider, indicating the Package ID |
| CI_ permission | TBD | Permissions for use that apply to the CI; different users will have different permissions |
| CI_guid | char_sequence | Global unique identifier of the Content Item |
| CI_name | CosNaming::name | COS Naming Service compound name |
| CI_owner | char_sequence | ID of the owner of the content item (Content Provider or SP) |
| CI_path | char_sequence | Physical location of the CI metadata |
| CI_title | char_sequence | Title of the CI |
| CI_TOC | set of {{CIE_guid, CIE_name}, CI_guid, CI_name}} | List of CIE names and IDs and CI names and IDs in the CI |
| CI_version | bslbf | Version of the CI (major and minor) |
| Award.date_received | UTC/MJD | Date awarded |
| Award.description | char_sequence | Award description |
| Award.name | bmp/ISO 10646-1 | Award name |
| CI_abbr_title | char_sequence | Abbreviated CI title |
| CI_date_available | UTC/MJD | Date available to customers |
| CI_delete_date | UTC/MJD | Date to delete from the SPS |
| CI_episode_name | bmp/ISO 10646-1 | Particular episode name |
| CI_expiration_date | UTC/MJD | Date content became unavailable to customers |
| CI_price_unit.price | TBD/fpvsbf | Price of content package from the CP to the SP, in appropriate units |
| CI_price_unit.user_price | TBD/fpvsbfFixed.2 | CP suggested price of content package from the SP to the user, in appropriate units |
| CI_Provider.address | bmp/ISO 10646-1 | Address of content provider |
| CI_Provider.contact_name | bmp/ISO 10646-1 | Contact name |
| CI_Provider.contact_phone | telephone_descriptor | Contact phone number |
| CI_Provider.ID11 | char_sequence | Global unique ID for the Content Item provider |
| CI_Provider.name | bmp/ISO 10646-1 | Name of content provider |

[11] CI_Provider.ID will likely not be used at the same time as the other CI_Provider attributes.

| | | |
|---|---|---|
| CI_provider_ref | char_sequence | Local ID of the CI to the Service Provider |
| CI_Restriction.Country.Dimension.Rating | dimension_name.text_rating | Rating in given country according to a given system (or dimension) by agreed text (e.g. in USA, Warnings, Language, or UK, BBFC, Uc) three-dimensional |
| CI_Restriction.Country.Rating_Body.Overall_Age_Rating | country_code.rating_body.age_rating | Rating in given country by minimum age by given rating body (e.g., 15 years in UK by BVA) three-dimensional |
| Review.critic_name | bmp/ISO 10646-1 | Name of critic |
| Review.description | bmp/ISO 10646-1 | Description of review |
| Review.language | ISO 639.2 | Language in which review is written |
| Review.org | bmp/ISO 10646-1 | Organisation that gave the review |
| Review.quote | bmp/ISO 10646-1 | Quote out of review |
| Review.rating | char_sequence | Rating on a specific scale (1-4 stars,...) |
| Review.text | bmp/ISO 10646-1 | Text of review |
| Role.description | bmp/ISO 10646-1 | Description or name of person/people taking that role |
| Role.name | role_char | Name of role |
| Schedule.channelno | char_sequence | logical channel number |
| Schedule.length | UTC/rel | length of program |
| Schedule.startdate | UTC/MJD | Starting date of program |
| Schedule.starttime | UTC/MJD | Starting time of program |

#### 11.3.2.3 Content Item Element Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Content Item Element Object |
| CPID | ObjRef | A globally unique ID indicating the Content Provider |
| PackageID | String | An ASCII character string, unique within the domain of the Content Provider, indicating the Package ID |
| CIE_ permission | TBD | Permissions for use that apply to the CIE; different users will have different permission |
| CIE_assoc_CI | set of char_sequence | Associated applications |
| CIE_comp | enumeration of { RT_program<br>Text<br>Index<br>SI<br>Compressed Audio<br>Linear Audio<br>Compressed Video | Content Item Element monomedia component type |



| | | |
|---|---|---|
| | Still Picture<br>Graphics<br>Java, MHEG-5<br>Private } | |
| CIE_guid | char_sequence | Global unique identifier |
| CIE_loading | set of {DSM Directory, execute} | List of loading commands to be executed at loading time. |
| CIE_name | CosNaming::name | COS Naming Service compound name |
| CIE_owner | char_sequence | Global unique identifier of legal owner |
| CIE_path | char_sequence, char_sequence | Physical location of the CIE metadata and monomedia component data |
| CIE_size | uimsbf | size in bytes of the content part of the element |
| CIE_version | bslbf | version number (major, minor) |
| CIE_virus_checked | UTC/MJD | Date and Time the CIE was last virus checked |
| CIE _prov_ref | char_sequence | Local ID of the CI to the Service Provider |
| CIE_index_start | DSM-CC_appNPT | Index information to reference specific frames of the content (assumes MPEG-2 TS) |
| CIE_index_stop | DSM-CC_appNPT | Index information to reference specific frames of the content (assumes MPEG-2 TS) |
| CIE_mult_lang | list of lang | Languages available in the CIE |
| CIE_Provider.address | bmp/ISO 10646-1 | Address of content provider |
| CIE_Provider.contact_name | bmp/ISO 10646-1 | Contact name |
| CIE_Provider.contact_phone | telephone_descriptor | Contact phone number |
| CIE_Provider.ID12 | char_sequence | Global unique ID for the Content Item Element provider |
| CIE_segment_table13 | Ordered list of {CIE_name} | List of the CIE segments for a multi-file CIE, in order. |
| CIE_type | content _type | Type of content unit (preview, main, logo) |
| EncoderProvider.address | bmp/ISO 10646-1 | Address of encoder provider |
| EncoderProvider.contact_name | bmp/ISO 10646-1 | Contact name |
| EncoderProvider.contact_phone | telephone_descriptor | Contact phone number |
| EncoderProvider.name | bmp/ISO 10646-1 | Name of encoder provider |



#### 11.3.2.4 Real Time Program Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Real Time Program Object |

[12] CIE_Provider.ID will likely not be used at the same time as the other CIE_Provider attributes.

[13] The CIE_segment_table attribute need to be included in the CIE metadata for each segment to which it applies.

| | | |
|---|---|---|
| Length | long | The length of the Real Time Program object, in bytes |
| Bitrate | uimsbf | Real Time Program bitrate in bits per second |
| Form | MediaTag | Recording format |
| RT_prog_comp | List of CIE_name | List of CIEs pointed to by the RT Program (MPEG-2 TS and Index only) |
| RTPData | ObjRef | Real Time Program data file |

#### 11.3.2.5 MPEG-2 TS Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | MPEG-TS Object |
| Length | long | The length of the MPEG2-TS object, in bytes |
| Bitrate | uimsbf | TS bitrate in bits per second |
| aspect_ratio | bslbf | Aspect ratio of the intended display |
| frame_rate | Fpvsbf | video frame rate in Hz |
| Horizontal_size | uimsbf | Number of pixels of the frame in the horizontal direction |
| level_profile | bslbf | level and profile used for video encoding, as specified by ISO/IEC 13818-2 |
| mult_lang_audio | Set of ISO 639.2 | list of one or more language identifiers for the audio tracks in the TS |
| mult_lang_text | Set of ISO 639.2 | list of one or more language identifiers for the text tracks (e.g., subtitles) in the TS |
| PES_comp | Setof {CIE_comp, uimsbf) | The component monomedia that has been mapped to the TS via the PES structure, and location (PID) |
| Ratio | uimsbf | Amount of speed up for the Fast forward or Fast Rewind streams |
| TS_comp | Set of {CIE_comp, uimsbf) | The component monomedia that has been mapped to the TS via the TS private section, and location (PID) |
| TS_mode | bslbf | Indication of whether the TS is the standard, the fast forward or the fast rewind stream |
| vertical_size | uimsbf | Number of pixels of the frame in the vertical direction |
| video_comp_type | enumeration of {MPEG-1, MPEG-2} | Type of video encoding |
| MPEGData | ObjRef | MPEG2-TS data file |

#### 11.3.2.6 Index Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Index Object |
| Length | long | The length of Index object, in bytes |



| | | |
|---|---|---|
| Index_mode | bslbf | Indication of whether the Index file applies to the standard, the fast forward or the fast rewind stream |
| IndexData | ObjRef | Index data file |

#### 11.3.2.7 Text Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Text Object |
| Length | long | The length of the Text object, in bytes |
| starting_corner | uimsbf | Location (in pixels) of the starting corner on the screen in terms of (# horizontal pixels, # vertical pixels), and where (0,0) is the upper left –hand corner |
| Text _flow | bslbf | Direction of the text flow (l-r, or r-l) |
| TextData | ObjRef | Text data file |

#### 11.3.2.8 System Information Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | System Information Object |
| Length | Long | The length of System Information object, in bytes |
| SI_tables | char_sequence | The type of DVB SI table |
| SIData | ObjRef | System information data file |

#### 11.3.2.9 Compressed Audio Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Compressed Audio Object |
| Length | Long | The length of the Compressed Audio object, in bytes |
| Bitrate | Uimsbf | The playback bitrate of the audio stream |
| audio_type | Bslbf | The type of audio encoding - MPEG-1 Layer I, MPEG-1 Layer II, AC-3, or MPEG-2 Layer II. |
| channel_format | enumeration of {sing_chan, dual_chan, joint_stereo, stereo} | The type of channels included in the audio stream |
| audio_type | Bslbf | The type of audio encoding - MPEG-1 Layer I, MPEG-1 Layer II, or AC-3. |
| channel_format | enumeration of {sing_chan, dual_chan, joint_stereo, stereo} | The type of channels included in the audio stream |
| CAData | ObjRef | Compressed Audio data file |

#### 11.3.2.10 Linear Audio Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Linear Audio Object |
| Length | Long | The length of the Linear Audio object, in bytes |
| Bitrate | Uimsbf | The playback bitrate of the audio stream |
| num_channels | Uimsbf | Whether one (mono) or two (stereo) channels are included in the audio stream |
| num_sample_frames | Uimsbf | Number of sample frames |
| sample_rate | Fpvsbf | The sample frequency of the encoded data |
| sample_size | Uimsbf | the number of bits per audio sample |
| LAData | ObjRef | Compressed Audio data file |

#### 11.3.2.11 Still Picture Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Still Picture Object |
| Length | Long | The length of the Still Picture object, in bytes |
| aspect_ratio | Bslbf | aspect ratio of the intended display |
| Horizontal_size | Uimsbf | Number of pixels of the frame in the horizontal direction |
| level_profile | Bslbf | level and profile used for video encoding, as specified by ISO/IEC 13818-2 |
| vertical_size | Uimsbf | Number of pixels of the frame in the vertical direction |
| SPData | ObjRef | Still Picture data file |

#### 11.3.2.12 Graphics Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Graphics Object |
| Length | Long | The length of the Graphics object, in bytes |
| coding_type | Bslbf | Coding type for still graphic - RGB16, CLUT8, CLUT4, or CLUT2 |
| dithering_pref | Bslbf | preference of whether or not to dither |
| Horizontal_position | Uimsbf | Horizontal offset between the top left corner of the object and the top left corner of the grid |
| Horizontal_size | Uimsbf | Number of pixels of the frame in the horizontal direction |
| object_scalability | Bslbf | describes minimum CLUT format for the object on display |
| pixel_aspect_ratio | Bslbf | aspect ration of a pixel |
| Resolution_ind | Bslbf | resolution of the graphical object based on screen size - half frame resolution in both horizontal and vertical, vertical only, horizontal only |



| | | |
|---|---|---|
| scaling_pref | Bslbf | preference for scaling if object ratio does not match display ratio |
| target_background_grid | Bslbf | Target background grid |
| vertical_position | Uimsbf | Vertical offset between the top left corner of the object and the top left corner of the grid |
| vertical_size | Uimsbf | Number of pixels of the frame in the vertical direction |
| GraphicData | ObjRef | Graphics data file |

#### 11.3.2.13 Java Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Java Object |
| Length | long | The length of the Java object, in bytes |
| Client_server | bslbf | Indication of whether the Java executable represents client or server code |
| JavaData | ObjRef | Java data file |

#### 11.3.2.14 MHEG-5 Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | MHEG-5 Object |
| Length | Long | The length of the MHEG-5 object, in bytes |
| MHEGData | ObjRef | MHEG-5 data file |

#### 11.3.2.15 Private Object

| *Property* | *Type* | *Value* |
|---|---|---|
| ObjID | ObjClass | Private Object |
| Length | Long | The length of the Private object, in bytes |
| private_data | Unspecified | User-defined data |



## 11.4. Content Loading Toolset

### 11.4.1 Scope

This chapter describes the requirements and desired features for content loading from a Content Provider to a Service Provider over the A10 interface. To meet these conditions, we shall choose CMSL (Content Metadata Specification Language) as the content loading toolset, and describe the specification in this chapter.

### 11.4.2 Toolset requirements

The requirements for the toolset are listed below.

1. Ability to create new CI/CIE Metadata objects based on the Bento Format.

2. Ability to specify the set of Metadata attributes (Bento properties) associated with a particular CI/CIE type (e.g. Text object).
3. Must be Available and complete.
4. Ability to transfer content packages from a Content Provider to a Service Provider over A10 interface.

### 11.4.3 Toolset features

These are non-essential features of a toolset but should be used as a guide to the sort of functionality required of a toolset for packaging metadata and monomedia components into content packages.

The features can be used as a means of assessing tools against each other. The features are listed in priority order.

1. Support for the toolset.
2. Ability to extend the standard set of attributes defined in ISO/IEC 16500 to include private attributes.
3. Lightweight content packages (minimum redundant information).
4. Intelligent file transfer mechanism (e.g. doesn't send files twice for different CIs)
5. Platform independent.
6. The components within the toolset should have simple, open and well defined interfaces so that they may interoperate.
7. Easy to use.
8. Metadata batch input function (e.g. creating new CI/CIE Metadata objects by extracting metadata from legacy databases).
9. Metadata validation (e.g. List of Values).
10. Ability to test the content (metadata and monomedia components) for completeness (against the specified set) and correctness of attribute values.

### 11.4.4 Toolset Specifications

A content specification is essentially a template from which instances of a particular CI or CIE type can be built, both in terms of its physical characteristics and its metadata requirements. For example, there may exist a specification for a Movie offering and an instance of it would be the 'Star Wars' movie. Only when specific values are assigned, is an instance of a content specification created. In this respect it is similar to a class in Object Oriented Technology, however, the specifications have additional parameters to define default values, constraints etc.

#### 11.4.4.1 CI/CIE Relationships and Components

Relationships identify and define the links between a CI or CIE and its components. For example a Movie CI type may have an 'Actors' attribute assigned to it which could take several values i.e. there is a one to many relationship between the CI and its 'Actors' attribute. Alternatively a Movie package may contain several 'Movie' offerings for which there would be a one to many relationship. Some of these relationships may be mandatory and others optional.



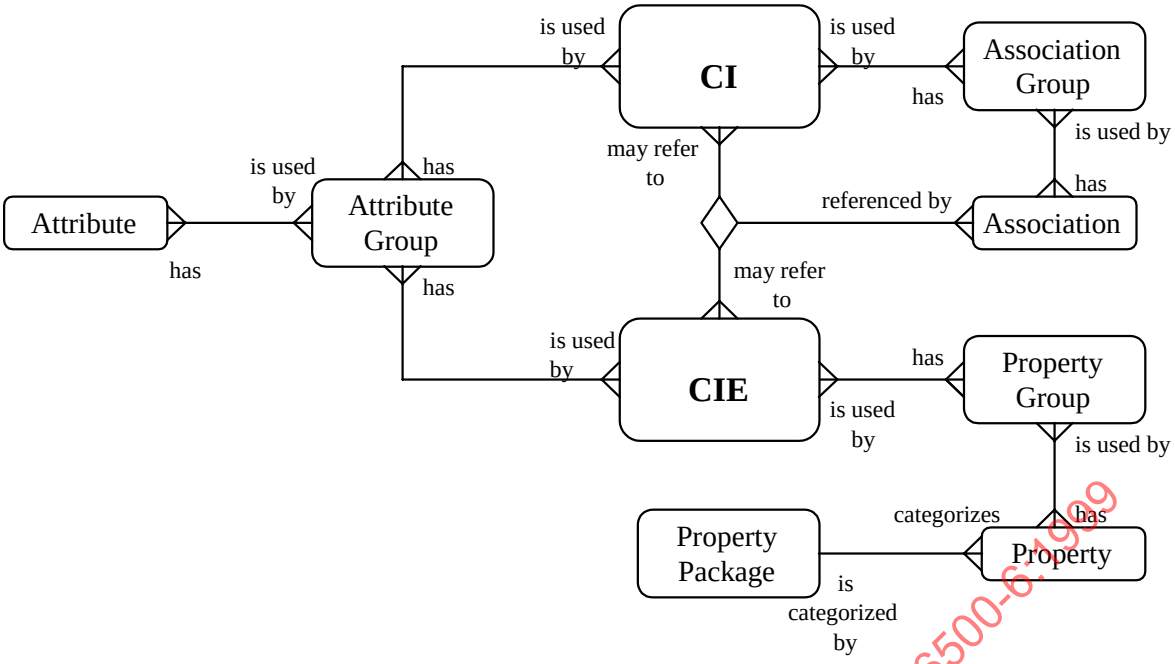



**Figure 11-5. — Entity Relationship Diagram**

The diagram in Figure 11.1 shows the different relationship types required to specify CI/CIE's in the toolset.

**Attributes -** These are used to describe characteristics of a specific CI/CIE metadata. For example a 'Movie' offering may use 'Actor', 'Rating' and 'Genre' attributes. Alternatively, a home shopping application may include a 'Shoes' offering which may use a 'Style' attribute. Within a CI or CIE , attributes are generic, mandatory or optional. The complete list of mandatory and optional attributes can be found in Subclause 11.2.3.2 and 11.2.3.3. Private attributes can also be defined.

**Properties** - Physical media is built to a set of media properties e.g. a JPEG image must be built with a size of 200x200. Properties are not required as metadata by a target Service. They are for use by the CP only.

**Attribute Groups** - These are used to group together related metadata attributes. By default, all attributes declared in the group must be considered mandatory unless stated otherwise.

**Property Groups** - These are used to declare media property values from a set of media properties declared in a Property Package.

**Property Packages** - These are sets of properties that can be used in a property group. A property package declares all the properties that may be used within a property group. A property group always contains a subset of the properties held in a property package.

**Association** - A CI or CIE may be a required or optional component of a CI. In some cases the context of the relationship may be important as e.g. "help", "background", "reference", etc.

**Association Groups** - A Relationship between a CI and CIE (or another CI) is declared in an Association Group.

**CIE** - This defines the specification that the actual physical CIE media should be built to by the included set of Property Groups. Any metadata required by the CIE is defined by the included set Attribute Groups.

**CI** - This defines any associated CIs and CIEs by the included set of Association Groups. Any attributes required by the CI are defined by the included set of Attribute Groups.



### 11.4.4.2 Specifying Relationships and Components using CMSL

Content Metadata Specification Language (CMSL) is a means of defining media and metadata specifications in an open and standardised language format, offering complete re-usability, flexibility and portability. The CMSL language consists of a number of major syntactic constructs, each of which is enclosed with its own unique keyword and common end marker e.g. :

**CI_SPEC**

**…**

**END**

Within the bounds of a keyword / End marker, other CMSL constructs and keywords can be declared in accordance with the syntactic rules of the language. In some cases parameters can be added to define maximum / minimum values, default values, optional values, number of duplicate occurrences, etc.

As in many languages comments may be included in the code by preceding them with two forward slashes '//'.

#### 11.4.4.3 CMSL Language Specification

The full specification of the CMSL language, using Backus-Naur Format (BNF), can be found in the informative Annex O. CMSL is currently at Draft Version 2.0. It is anticipated that a new Version 3.0 will be produced this year which will have a modified entity-relationship model. These modifications will be small but nevertheless significant. A CMSL Example used to define a simple Movie Offering can be found in Annex P.

### 11.4.5 Content Loading by CMSL

A CI/CIE CMSL specification must not be thought of as an instance of an actual CI/CIE. It defines the attribute names and associations of the CI/CIE together with any value constraints, not the actual values. However, using the same entity-relationship model as CMSL a set of generic class definitions can be defined, based on the same major constructs in CMSL but requiring additional elements for instantiation. So, by interpreting a CI/CIE CMSL specification, a CI/CIE object of a particular type can be created and then populated with (meaningful) metadata.

Instantiated CI/CIE's cannot be loaded directly into a Bento file container. They must first of all be encoded into Bento Object format using the Bento API. Similarly, they can only be extracted from the Bento container after decoding.



**(Blank page)**

# Annex A
(normative)
# Coding of Outline Fonts

## A.1 Font Format Specification

The scaleable outline format representation will be referred to as a Portable font resource (PFR) that can be stored statically in ROM or hard disks, or moved dynamically within a DAVIC network. This dynamic aspect is the reason the font resource is often referred to as portable. The file representation of the PFR is designed with two, sometimes conflicting, goals in mind. One is to minimize the size of the file representation; the other is to provide the information in a way that optimizes rendering performance even if the amount of memory is limited at playback time.

A Portable Font Resource consists of the following sections in order:

- PFR header
- Logical font directory
- Logical font section
- Physical font section
- Glyph program strings
- PFR trailer

The PFR header contains global information about the PFR and the fonts contained within it.

The logical font directory consists of a table of pointers to the logical fonts contained within the PFR.

The logical font section contains the logical font records themselves. Each logical font record defines the transformation (size, oblique effect, condense, expand) to be applied to a physical font. It therefore represents an instance of a physical font.

The physical font section consists of a set of physical font records. Each physical font records contains information about one physical font contained within the PFR including a table of character codes defined for that physical font. A physical font record may optionally be immediately followed by bitmap size and bitmap character table records associated with that physical font.

The glyph program strings section contains the definition of the shapes of each of the characters defined within the font. Both outline and bitmap image shapes are defined by glyph program strings. Glyph program strings are shared across all physical fonts within a PFR.

All integers are written most significant bit first.

All offsets except those in glyph program strings are relative to the first byte of the portable font resource.

All offsets within glyph program strings are relative to the first byte of the first glyph program string.

Many of the concepts used in this specification are based on the Adobe Type 1 font format version 1.1 (Addison-Wesley Publishing Company, Inc. 1991).



## A.2 PFR Header

The PFR header is the first block of data in a Portable Font Resource. It contains global information about the PFR and its constituent fonts.

The size of the PFR header is a fixed 58 bytes. Its structure is as follows:

**Table A-1. — PFR Header**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| pfrHeader() { | | |
| pfrHeaderSig | 32 | bslbf |

| | | |
|---|---|---|
| pfrVersion | 16 | uimsbf |
| pfrHeaderSig2 | 16 | bslbf |
| pfrHeaderSize | 16 | uimsbf |
| logFontDirSize | 16 | uimsbf |
| logFontDirOffset | | 16 |
| uimsbf | | |
| logFontMaxSize | 16 | uimsbf |
| logFontSectionSize | 24 | uimsbf |
| logFontSectionOffset | 24 | uimsbf |
| physFontMaxSize | 16 | uimsbf |
| physFontSectionSize | 24 | uimsbf |
| physFontSectionOffset | 24 | uimsbf |
| gpsMaxSize | 16 | uimsbf |
| gpsSectionSize | 24 | uimsbf |
| gpsSectionOffset | 24 | uimsbf |
| maxBlueValues | 8 | uimsbf |
| maxXorus | 8 | uimsbf |
| maxYorus | 8 | uimsbf |
| physFontMaxSizeHighByte | 8 | uimsbf |
| zeros | 6 | bslbf |
| pfrInvertBitmap | 1 | bslbf |
| pfrBlackPixel | 1 | bslbf |
| bctMaxSize | 24 | uimsbf |
| bctSetMaxSize | 24 | uimsbf |
| pftBctSetMaxSize | 24 | uimsbf |
| nPhysFonts | 16 | uimsbf |
| maxStemSnapVsize | 8 | uimsbf |
| maxStemSnapHsize | 8 | uimsbf |
| maxChars | 16 | uimsbf |
| } | | |



**pfrHeaderSig:** A byte string which indicates the file type and format. This field shall be set to the constant value 0x50465230 representing the ASCII string “PFR0”.

**pfrVersion:** An unsigned integer indicating the PFR format version number. This field shall be set to the value 4.

**pfrHeaderSig2:** A byte string which further confirms the integrity of the PFR. This field shall be set to the constant value 0x0d0a representing the ASCII characters carriage return and line feed.

**pfrHeaderSize:** An unsigned integer indicating the number of bytes in the PFR header. This field shall be set to the constant value 58.

**logFontDirSize:** An unsigned integer indicating the total size of the logical font directory in bytes.

**logFontDirOffset:** An unsigned integer indicating the byte offset of the first byte of the logical font directory relative to the first byte of the PFR header.

**logFontMaxSize:** An unsigned integer indicating the size in bytes of the largest logical font record. This may be used to allocate a buffer capable of holding any logical font record.

**logFontSectionSize:** An unsigned integer indicating the size in bytes of the entire set of logical font records. This may be used to allocate a buffer capable of holding the entire logical font section.

**logFontSectionOffset:** An unsigned integer indicating the byte offset of the first byte of the first logical font record in the logical font section. The offset is relative to the first byte of the PFR header.

**physFontMaxSize:** An unsigned integer indicating the size in bytes of the largest physical font record. This may be used to allocate a buffer capable of holding any physical font record.

**physFontSectionSize:** An unsigned integer indicating the size in bytes of the entire set of physical font records. This may be used to allocate a buffer capable of holding the entire physical font section.

**physFontSectionOffset:** An unsigned integer indicating the byte offset of the first byte of the first physical font in the physical font section. The offset is relative to the first byte of the PFR header.

**gpsMaxSize:** An unsigned integer indicating the size in bytes of the largest glyph program string. In the case of compound glyphs, the size must include the total size of its component glyphs. This may be used to allocate a buffer capable of holding any glyph program string.

**gpsSectionSize:** An unsigned integer indicating the size in bytes of the entire set of glyph program strings. This may be used to allocate a buffer capable of holding the entire set of glyph program strings.

**gpsSectionOffset:** An unsigned integer indicating the byte offset to the first byte of the first glyph program string in the glyph program string section. The offset is relative to the first byte of the PFR header.

**maxBlueValues:** An unsigned integer indicating the maximum number of vertical alignment zones defined in any physical font record.

**maxXorus:** An unsigned integer indicating the maximum number of controlled X coordinates in any glyph program string. The number of controlled X coordinates in a glyph program string includes primary stroke edges, secondary stroke edges and secondary edges.

**maxYorus:** An unsigned integer indicating the maximum number of controlled Y coordinates in any glyph program string. The number of controlled Y coordinates in a glyph program string includes primary stroke edges, secondary stroke edges and secondary edges.

**physFontMaxSizeHighByte:** An unsigned number indicating the number of times 65536 should be added to the specified value of physFontMaxSize. This provides a means of handling physical font records whose size exceeds 64K bytes.

**zeros:** A concatenation of bits that shall all be set to zero.

**pfrInvertBitmap:** A bit flag that indicates, if set, that image data in bitmap glyph program strings is stored in decreasing order of Y value.

**pfrBlackPixel:** A bit flag that indicates the bit value used to represent black in image data contained in bitmap glyph program strings.

**bctMaxSize:** An unsigned integer that indicates the maximum size in bytes of any bitmap character table in any physical font record. This may be used to allocate a buffer capable of holding any bitmap character table.

**bctSetMaxSize:** An unsigned integer that indicates the maximum size in bytes of any complete set of bitmap character tables in any physical font. This may be used to allocate a buffer capable of holding the set of bitmap character tables for any physical font.

**pftBctSetMaxSize:** An unsigned integer that indicates the maximum size in bytes of any physical font record together with all of the bitmap character tables associated with it. This may be used to allocate a buffer capable of holding any physical font record and its associated bitmap character tables.

**nPhysFonts:** An unsigned integer that indicates the number of physical font records in the physical font section.

**maxStemSnapVsize:** An unsigned integer that indicates the number of values contained in the largest vertical stem snap table in any physical font record.

**maxStemSnapHsize:** An unsigned integer that indicates the number of values contained in the largest horizontal stem snap table in any physical font record.

**maxChars:** An unsigned integer that indicates the maximum number of characters in any of the physical font records.



## A.3 Logical font directory

The logical font directory consists of a table of pointers to all of the logical font records contained with the PFR. The table is indexed by the logical font code. The structure of the logical font directory is as follows.

**Table A-2. — Logical Font Directory**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| logFontDir() { | | |
| nLogFonts | 16 | uimsbf |
| for (i = 0; i < nLogFonts; i++){ | | |
| logFontSize | 16 | uimsbf |

| | | | |
|---|---|---|---|
| | logFontOffset | 24 | uimsbf |
| | } | | |
| } | | | |

**nLogFonts:** An unsigned integer indicating the total number of logical fonts contained in the logical font directory.

**logFontSize:** An unsigned integer indicating the size in bytes of one logical font record.

**logFontOffset:** An unsigned integer indicating the byte offset to the first byte of that logical font record. The offset is relative to the first byte of the PFR header.

## A.4 Logical font section

The logical font section consists of zero or more logical font records. Each logical record contains information about one logical font. It is accessed via the logFontOffset value in the appropriate entry in the logical font directory. The structure of the logical font section is as follows.

**Table A-3. — Logical Font Section**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| logFontSection() { | | |
| for (i = 0; i < nLogFonts; i++){ | | |
| logFontRecord() | | |
| } | | |
| } | | |

The structure of each logical font record is as follows.

**Table A-4. — Logical Font Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| logFontRecord() { | | |
| fontMatrix[0] | 24 | tcimsbf |
| fontMatrix[1] | 24 | tcimsbf |
| fontMatrix[2] | 24 | tcimsbf |
| fontMatrix[3] | 24 | tcimsbf |
| zero | 1 | bslbf |
| extraItemsPresent | 1 | bslbf |
| twoByteBoldThicknessValue | 1 | bslbf |
| boldFlag | 1 | bslbf |
| twoByteStrokeThicknessValue | 1 | bslbf |
| strokeFlag | 1 | bslbf |
| lineJoinType | 2 | bslbf |
| if (strokeFlag){ | | |
| if (twoByteStrokeThicknessValue) | | |
| strokeThickness | 16 | tcimsbf |
| else | | |
| strokeThickness | 8 | uimsbf |
| if (lineJoinType == MITERLINEJOIN) | | |
| miterLimit | 24 | tcimsbf |
| } | | |
| else if (boldFlag){ | | |
| if (twoByteBoldThicknessValue) | | |
| boldThickness | 16 | tcimsbf |
| else | | |
| boldThickness | 8 | uimsbf |
| } | | |
| if (extraItemsPresent){ | | |
| nExtraItems | 8 | uimsbf |
| for (i = 0; i < nExtraItems; i++){ | | |

| | | |
|---|---|---|
| extraItemSize | 8 | uimsbf |
| extraItemType | 8 | uimsbf |
| for (j = 0; j < extraItemSize; j++){ | | |
| extraItemData | 8 | uimsbf |
| } | | |
| } | | |
| } | | |
| physFontSize | 16 | uimsbf |
| physFontOffset | 24 | uimsbf |
| if (pfrHeader.physFontMaxSizeHighByte){ | | |
| physFontSizeIncrement | 8 | uimsbf |
| } | | |
| } | | |

**fontMatrix[]:** An array of four signed integers representing the coefficients of the transformation matrix (in units of 1/256) from the font's coordinate system to the document coordinate system. The defined transformation is:

x' = (fontMatrix[0] * x + fontMatrix[2] * y) / (256 * outlineResolution)

y' = (fontMatrix[1] * x + fontMatrix[3] * y) / (256 * outlineResolution)

where (x, y) is a point in the character outline resolution unit coordinate system, outlineResolution is the resolution of the associated physical font and (x', y') is the corresponding point in the (scaled) logical font.

**zero:** A bit flag that shall be set to zero.

**extraItemsPresent:** A bit flag that indicates that the logical font contains extra data items. This should be set to zero for the current version as no extra item types are defined for logical font records.

**twoByteBoldThicknessValue:** A bit flag that indicates that the bold thickness value is expressed as a signed 16-bit integer rather than as an unsigned 8-bit integer.

**boldFlag:** A bit flag that indicates that emboldening should be enabled when rendering this logical font.

**twoByteStrokeThicknessValue:** A bit flag that indicates that the stroke thickness value is expressed as a signed 16-bit integer rather than as an unsigned 8-bit integer.

**strokeFlag:** A bit flag that indicates that this logical font should be rendered by drawing a stroke with the specified thickness around the outline rather than by conventionally filling the interior of the outline. Note that if this flag is set, it overrides the bold flag.

**lineJoinType:** A two-bit field that indicates how convex corners should be handled during stroked rendering. Its values are the standard PostScript definitions:

0: MITER_LINE_JOIN

1: ROUND_LINE_JOIN

2: BEVEL_LINE_JOIN

3: Undefined

**strokeThickness:** This is a signed integer that indicates the thickness of the stroke to be used to render the character in stroke mode. The units are character coordinates (outline resolution units). If twoByteStrokeThicknessValue is equal to zero, this value is represented by an unsigned 8-bit field. If twoByteStrokeThicknessValue is equal to one, this value is represented by an signed 16-bit field. The effect of using a negative value for stroke thickness is undefined.

**miterLimit:** A signed integer representing the limit beyond which mitered corners should be rendered as beveled corners. It is represented by the standard PostScript value for miterLimit. The value represents the maximum value of the miter ratio for any mitered corner in units of 1/65536. The miter ratio is the distance of the mitered corner from the outline corner divided by half the bold or stroke thickness.

**boldThickness:** This is a signed integer that indicates the total amount by which rendered characters should be emboldened. The units are character coordinates (outline resolution units). Thus, for example, a 100 by 200 square emboldened by 10 units would be rendered as if the square were 110 by 210 character units. If twoByteBoldThicknessValue is equal to zero, this value is represented by an unsigned 8-bit field. If twoByteBoldThicknessValue is equal to one, this value is represented by an signed 16-bit field. A negative value for



boldThickness may be used to specify a reduced boldness for a character. This should be used with caution as an excessively negative value for boldThickness can cause thin parts of a character shape to turn inside out.

**nExtraItems:** An unsigned integer that indicates the number of extra data items present. Extra data items added to a logical font contain data that will be ignored by earlier versions of the PFR interpreter and used by later versions of the PFR interpreter.

**extraItemSize:** An unsigned integer indicating the size in bytes of one extra data item. The size includes only the extra item data following the extraItemType field.

**extraItemType:** An unsigned integer indicating the type of extra item data present. No extra data item types have been defined for logical font records at this time.

**extraItemData:** One byte of extra item data. This data is interpreted in accordance with the extraItemType defined for logical font records. All undefined extra item types will be ignored.

**physFontSize:** An unsigned integer that defines the size in bytes of the associated physical font record.

**physFontOffset:** An unsigned integer that defines the offset in bytes of the first byte of the associated physical font record. The offset is relative to the first byte of the PFR header.

**physFontSizeIncrement:** An unsigned integer that allows physical font sizes in excess of 64K bytes to be supported. If the physFontMaxSizeHighByte field in the PFR header is non-zero, the value of physFontSizeIncrement is multiplied by 65535 and added to physFontSize. In other words, it provides a high byte for the physical font size.

## A.5 Physical font record

The physical font section contains information about each of the physical fonts contained in the PFR. Each physical font is represented by a physical font record containing information about one physical font. It is accessed via the physFontOffset value in the parent logical font record. In the case that bitmaps are associated with a physical font, the bitmap size records and bitmap character tables appear immediately following the parent physical font. The structure of the physical font section is as follows.

**Table A-5. — Physical Font Section**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| physFontSection() { | | |
| for (i = 0; i < nPhysFonts; i++) | | |
| { | | |
| physFontRecord() | | |
| } | | |
| } | | |

The structure of each physical font record in the physical font section is as follows.

**Table A-6. — Physical Font Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| physFontRecord() { | | |
| fontRefNumber | 16 | uimsbf |
| outlineResolution | 16 | uimsbf |
| metricsResolution | 16 | uimsbf |
| xMin | 16 | tcimsbf |
| yMin | 16 | tcimsbf |
| xMax | 16 | tcimsbf |
| yMax | 16 | tcimsbf |
| extraItemsPresent | 1 | bslbf |
| zero | 1 | bslbf |
| threeByteGpsOffset | 1 | bslbf |
| twoByteGpsSize | 1 | bslbf |
| asciiCodeSpecified | 1 | bslbf |
| proportionalEscapement | 1 | bslbf |
| twoByteCharCode | 1 | bslbf |



```
verticalEscapement                                          1       bslbf
if (!proportionalEscapement)
        standardSetWidth                                            16
tcimsbf
if (extraItemsPresent){
        nExtraItems                                         8       uimsbf
        for (i = 0; i < nExtraItems; i++){
                extraItemSize                               8       uimsbf
                extraItemType                               8       uimsbf
                switch(extraItemType){
                case 1:
                        bitmapInfo()
                        break;
                case 2:
                        fontID()
                        break;
                case 3:
                        stemSnapTables()
                        break;
                default:
                        for (j = 0; j < extraItemSize; j++){
                                extraItemData               8       uimsbf
                                }
                        break;
                        }
                }
        }
nAuxBytes                                                   24      uimsbf
for(I=0; I<nAuxBytes; I++)
        {
        auxData                                             8       uimsbf
        }
nBlueValues                                                 8       uimsbf
for(i = 0; i < nBlueValues; i++)
        {
        blueValue[i]                                        16      tcimsbf
        }
blueFuzz                                                    8       uimsbf
blueScale                                                   8       uimsbf
stdVW                                                       16      uimsbf
stdHW                                                       16      uimsbf
nCharacters                                                 16      uimsbf
for (i = 0; i < nCharacters; i++)
        {
        charRecord()
        }
```



**fontRefNumber:** An unsigned integer that uniquely defines the physical font record within the PFR. Conventionally, physical fonts are numbered in sequence starting at 0.

**outlineResolution:** A signed integer that defines the resolution of the coordinate system of the character outlines. The value represents the number of units in one em.

**metricsResolution;** A signed integer that defines the resolution of the coordinate system of the character set width values. The value represents the number of units in one em.

**xMin:** A signed integer whose value is the smallest value of any X-coordinate of any point in the outline representation of any character in the physical font.

**yMin:** A signed integer whose value is the smallest value of any Y-coordinate of any point in the outline representation of any character in the physical font.

**xMax:** A signed integer whose value is the largest value of any X-coordinate of any point in the outline representation of any character in the physical font.

**yMax:** A signed integer whose value is the largest value of any Y-coordinate of any point in the outline representation of any character in the physical font.

**extraItemsPresent:** A bit flag that indicates that the physical font contains extra data items. This field is normally set to one because the fontID extra item is required to be present.

**zero:** A bit flag that shall be set to zero.

**threeByteGpsOffset:** A bit flag that indicates that the value of gpsOffset is represented by a three-byte rather than by a two-byte integer.

**twoByteGpsSize:** A bit flag that indicates that the value of gpsSize is represented by a two-byte rather than by a single-byte integer.

**asciiCodeSpecified:** Obsolete; set to zero.

**proportionalEscapement:** A bit flag that indicates that the set width is specified for each character rather than for all characters.

**twoByteCharCode:** A bit flag that indicates that the value of charCode is represented by a two-byte rather than by a single byte integer.

**verticalEscapement:** A bit flag that indicates that set width value should be interpreted as a vertical rather than horizontal escapement value.

**standardSetWidth:** A signed integer whose value is the set width of all characters in the font.

**nExtraItems:** An unsigned integer that indicates the number of extra data items present. Extra data items added to a logical font contain data that will be ignored by earlier versions of the PFR interpreter and used by later versions of the PFR interpreter. This field normally has a value of at least one because the fontID extra item is required in the current version.

**extraItemSize:** An unsigned integer indicating the size in bytes of one extra data item. The size includes only the extra item data following the extraItemType field.

**extraItemType:** An unsigned integer indicating the type of extra item data present. Three extra data item types (values 1 - 3) have been defined for physical font records.

**extraItemData:** One byte of extra item data. This data is interpreted in accordance with the values of extraItemType defined for physical font records. All undefined extra item types will be ignored.

**nAuxBytes:** An unsigned integer defining the number of bytes of auxiliary data that follow.

**auxData:** nAuxBytes bytes of arbitrary data.

**nBlueValues:** An unsigned integer defining the number of vertical alignment edges. The number of alignment edges shall always be an even number.

**blueValue:** A signed integer defining the Y-coordinate of one edge of a blue zone. The contained blue values must be in ascending order. Each succeeding pair of blue values defines one blue zone. See the Adobe Type 1 Font Format specification for details on the effect of defining blue zones.

**blueFuzz:** An unsigned integer defining the value of blueFuzz. See the Adobe Type 1 Font Format specification for details on the effect of this value.

**blueScale:** An unsigned integer defining the value of blueScale See the Adobe Type 1 Font Format specification for details on the effect of this value.

**stdVW:** An unsigned integer defining the value of stdVW See the Adobe Type 1 Font Format specification for details on the effect of this value.

**stdHW:** An unsigned integer defining the value of stdHW See the Adobe Type 1 Font Format specification for details on the effect of this value.

**nCharacters:** An unsigned integer defining the number of character records present. Character records following this field must be in ascending order of charCode.

The format of each character record is as follows.



**Table A-7. — Character Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| charRecord() { | | |
| if (twoByteCharCode) | | |
| charCode | 16 | uimsbf |
| else | | |
| charCode | 8 | uimsbf |
| if (proportionalEscapement) | | |
| charSetWidth | 16 | tcimsbf |
| if (asciiCodeSpecified) | | |
| asciiCodeValue | 8 | uimsbf |
| if (twoByteGpsSize) | | |
| gpsSize | 16 | uimsbf |
| else | | |
| gpsSize | 8 | uimsbf |
| if (threeByteGpsOffset) | | |
| gpsOffset | 24 | uimsbf |
| else | | |
| gpsOffset | 16 | uimsbf |
| } | | |

**charCode:** An unsigned integer defining the character code value.

**charSetWidth:** A signed integer defining the set width of the character. This determines the horizontal or vertical distance from the origin of the current character to the origin of the immediately following character.

**asciiCodeValue:** This field if present should be ignored.

**gpsSize:** An unsigned integer indicating the size in bytes of the glyph program string containing the outline representation of the character.

**gpsOffset:** An unsigned integer indicating the byte offset of the first byte of the glyph program string containing the outline representation of the character. The offset is relative to the first byte of the first glyph program string in the glyph program string section of the PFR.

### A.5.1 Bitmap Information

Optional bitmap information, contained in one or more extra data items in a physical font record, associates a set of bitmap character tables with that physical font record. These bitmap character tables must be written immediately following the parent physical font record. As these bitmap character tables are individually accessed via bitmap size records their order is arbitrary. Each bitmap character table contains bitmap character records for a single bitmap size. Bitmap size is measured in pixels per em. The horizontal size of a bitmap image may be different from its vertical size. The bitmap information record consists of a header followed by one or more bitmap size records. The structure of the bitmap information record is as follows.

**Table A-8. — Bitmap Information Extra Data Item**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| bitmapInfo(){ | | |
| fontBctSize | 24 | tcimsbf |
| zeros | 3 | bslbf |
| twoByteNBmapChars | 1 | bslbf |
| threeByteBctOffset | 1 | bslbf |
| threeByteBctSize | 1 | bslbf |
| twoByteYppm | 1 | bslbf |
| twoByteXppm | 1 | bslbf |
| nBitmapSizes | 8 | uimsbf |
| for (i = 0; i < nBitmapSizes; i++){ | | |
| bmapSizeRecord() | | |
| } | | |



```
        }
```

**fontBctSize:** A signed integer that represents the total size in bytes of all bitmap character tables associated with this physical font record. Note that if there are multiple bitmap information records associated with a physical font record, all must have the same value of fontBctSize.

**zeros:** A concatenation of bits that shall all be set to zero.

**twoByteNBmapChars:** A bit flag that is set to indicate that the nBmapChars field in each bitmap size record within this bitmap information record is represented by two bytes rather than by a single byte field.

**threeByteBctOffset:** A bit flag that is set to indicate that the bctOffset field in each bitmap size record within this bitmap information record is represented by three bytes rather than by a two byte field.

**threeByteBctSize:** A bit flag that is set to indicate that the bctSize field in each bitmap size record within this bitmap information record is represented by three bytes rather than by a two byte field.

**twoByteYppm:** A bit flag that is set to indicate that the yppm field in each bitmap size record within this bitmap information record is represented by two bytes rather than by a single byte field.

**twoByteXppm:** A bit flag that is set to indicate that the xppm field in each bitmap size record within this bitmap information record is represented by a two bytes rather than by a single byte field.

**nBitmapSizes:** An unsigned integer indicating the number of bitmap size records that appear in the remainder of the bitmap information record.

The number of bitmap size records that can fit in one extra data item is limited by the 256 byte limit on the total size of any extra data item. Multiple extra data items may be used to get around this limitation. Each bitmap size record contains information about one bitmap character table. Within each extra data item, bitmap size records must be in ascending order of Y pixels per em (X pixels per em is a secondary sort key in the event of duplicate values of Y pixels per em). The format of each bitmap size record is as follows.

**Table A-9. — Bitmap Size Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| bmapSizeRecord() { | | |
| if (twoByteXppm) | | |
| xppm | 16 | uimsbf |
| else | | |
| xppm | 8 | uimsbf |
| if (twoByteYppm) | | |
| yppm | 16 | uimsbf |
| else | | |
| yppm | 8 | uimsbf |
| zeros | 5 | bslbf |
| threeByteGpsOffset | 1 | bslbf |
| twoByteGpsSize | 1 | bslbf |
| twoByteCharCode | 1 | bslbf |
| if (threeByteBctSize) | | |
| bctSize | 24 | uimsbf |
| else | | |
| bctsize | 16 | uimsbf |
| if (threeByteBctOffset) | | |
| bctOffset | 24 | uimsbf |
| else | | |
| bctOffset | 16 | uimsbf |
| if (twoByteNBmapChars) | | |
| nBmapChars | 16 | uimsbf |
| else | | |
| nBmapChars | 8 | uimsbf |
| } | | |

**xppm:** An unsigned integer that represents the number of pixels per em in the X dimension

**yppm:** An unsigned integer that represents the number of pixels per em in the Y dimension

**zeros:** A concatenation of bits that shall all be set to zero.

**threeByteGpsOffset:** A bit flag that is set to indicate that the value of gpsOffset is represented by a three-byte rather than by a two-byte integer.

**twoByteGpsSize:** A bit flag that is set to indicate that the value of gpsSize is represented by a two-byte integer rather than by a single-byte integer.

**twoByteCharCode:** A bit flag that is set to indicate that the value of charCode is represented by a two-byte flag rather than by a single-byte flag.

**bctSize:** An unsigned integer that represents the total size in bytes of the bitmap character table for the specified values of xppm and yppm.

**bctOffset:** An unsigned integer that represents the offset in bytes of the first byte of the bitmap character table for the specified values of xppm and yppm. The offset is relative to the first byte of the parent physical font record.

**nBmapChars:** An unsigned integer indicating the number of bitmap character records provided in the bitmap character table.

### A.5.2 Font ID

The font ID provides a means of uniquely identifying the physical font. It is structured as a type 2 extra data item for physical font records. Its data consists of 1 to 254 non-null bytes followed by a null byte. This extra data item must be present.

The structure of the fontID record is as follows:

**Table A-10. — Font ID Extra Data Item**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| fontID() { | | |
| for (i = 0; ; i++){ | | |
| character[i] | 8 | uimsbf |
| if (character[i] == 0) | | |
| break | | |
| } | | |
| } | | |

**character[]:** An unsigned integer representing each character in the name of the physical font. Although the preferred coding system is ASCII, any 8-bit coding system can be used as long as it is consistent with the manner in which the font is referred to.

### A.5.3 Stem snap tables

Stem snap tables may be specified to enhance stem weight consistency during rendering by providing values of secondary stem weights (other than the primary values of stdVW and stdHW) which can be used for dynamic stem weight regularization. See the Adobe Type 1 font format specification for details on the behavior of stem snap tables. The vertical stem snap table contains zero or more values of vertical stem sizes measured in character outline resolution units. The horizontal stem snap table contains zero or more values of horizontal stem sizes measured in character outline resolution units. Stem snap tables are structured as a type 3 extra data item for a physical font. The format of the stem snap table data is as follows.

**Table A-11. — Stem Snap Tables**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| stemSnapTables() { | | |
| sshSize | 4 | uimsbf |
| ssvSize | 4 | uimsbf |
| for (i = 0; i < ssvSize; i++){ | | |
| stemSnapV[i] | 16 | tcimsbf |



| | | |
|---|---|---|
| }<br>for (i = 0; i < sshSize; i++){<br>stemSnapH[i]<br>}<br>} | 16 | tcimsbf |

The values in each of the stem snap tables (vertical and horizontal) must be in ascending order.

**sshSize:** An unsigned integer indicating the number of horizontal stem snap values provided.

**ssvSize:** An unsigned integer indicating the number of vertical stem snap values provided.

**stemSnapV:** A signed integer representing one secondary vertical stem weight in character outline units.

**stemSnapH:** A signed integer representing one secondary horizontal stem weight in character outline units.

### A.5.4 Bitmap character tables

Bitmap character tables provide a means of finding an optional bitmap image associated with a character code. A bitmap character table consists of a list of character codes and for each character code a pointer to the bitmap glyph program string containing the character image.

Bitmap character tables are written immediately following the physical font record with which they are associated. Each bitmap character table consists of one or more bitmap character records arranged in increasing order of character code.

The format of each bitmap character record in a bitmap character table is as follows.

**Table A-12. — Bitmap Character Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| bmapCharRecord() { | | |
| if (twoByteCharCode) | | |
| charCode | 16 | uimsbf |
| else | | |
| charCode | 8 | uimsbf |
| if (twoByteGpsSize) | | |
| gpsSize | 16 | uimsbf |
| else | | |
| gpsSize | 8 | uimsbf |
| if (ThreeByteGpsOffset) | | |
| gpsOffset | 24 | uimsbf |
| else | | |
| gpsOffset | 16 | uimsbf |
| } | | |



Note that because twoByteCharCode, twoByteGpsSize and ThreeByteGpsOffset are defined in the parent bitmap size record, they apply to all bitmap character records in a given bitmap character table. This ensures that the size in bytes of every record in a bitmap character table is the same.

**charCode:** An unsigned integer defining the bitmap character code value.

**gpsSize:** An unsigned integer indicating the size of the glyph program string containing the bitmap image of the character.

**gpsOffset:** An unsigned integer indicating the byte offset of the first byte of the glyph program string containing the bitmap image of the character. The offset is relative to the first byte of the first glyph program string in the glyph program string section of the PFR.

## A.6 Glyph program strings

Glyph program strings define character shapes and images. There are three kinds of glyph program strings supported:

- Simple glyph program strings that encode a scaleable glyph shape

- Compound glyph program strings that define a scaleable glyph in terms of one or more simple glyph program strings.
- Bitmap glyph program strings that encode a bitmap image.

## A.7 Simple glyph program strings

A simple glyph program string defines the shape of a character as zero or more outline contours. The points defining the outline are in character outline resolution units based on the value of outlineResolution in the parent physical font record. The structure of a glyph program string is as follows.

**Table A-13. — Simple Glyph Program String**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| simpleGps() { | | |
| isCompoundGlyph | 1 | bslbf |
| zeros | 3 | bslbf |
| extraItemsPresent | 1 | bslbf |
| oneByteXYCoordCount | 1 | bslbf |
| controlledYCoords | 1 | bslbf |
| controlledXCoords | 1 | bslbf |
| if (oneByteXYCoordCount) | | |
| { | | |
| nYorus | 4 | uimsbf |
| nXorus | 4 | uimsbf |
| } | | |
| else | | |
| { | | |
| if (controlledXCoords) | | |
| { | | |
| nXorus | 8 | uimsbf |
| } | | |
| if (controlledYCoords) | | |
| { | | |
| nYorus | 8 | uimsbf |
| } | | |
| } | | |
| for (i = 0; i < (nXorus + nYorus); i++) | | |
| { | | |
| if (i & 7 == 0) | | |
| { | | |
| twoByteCoord[7] | 1 | bslbf |
| twoByteCoord[6] | 1 | bslbf |
| twoByteCoord[5] | 1 | bslbf |
| twoByteCoord[4] | 1 | bslbf |
| twoByteCoord[3] | 1 | bslbf |
| twoByteCoord[2] | 1 | bslbf |
| twoByteCoord[1] | 1 | bslbf |
| twoByteCoord[0] | 1 | bslbf |
| } | | |
| if (twoByteCoord[i & 7]) | | |
| oruValue | 16 | tcimsbf |
| else | | |
| oruValue | 8 | uimsbf |
| } | | |
| if (extraItemsPresent) | | |
| { | | |
| nExtraItems | 8 | uimsbf |
| for (i = 0; i < nExtraItems; i++){ | | |
| extraItemSize | 8 | uimsbf |



```
                    extraItemType                                   8       uimsbf
                    switch(extraItemType){
                    case 1:
                            secondaryStrokeInfo()
                            break;
                    case 2:
                            secondaryEdgeInfo()
                            break;
                    default:
                            for (j = 0; j < extraItemSize, j++){
                                    extraItemData                   8       uimsbf
                                    }
                            break;
                            }
                    }
            }
    do
            {
            glyphOutlineRecord()
            } while (!endGlyph)
    }
```

**isCompoundGlyph:** A bit flag that indicates that the glyph is compound. This flag is always clear for a simple glyph program string.

**zeros:** A concatenation of bits that shall all be set to zero.

**extraItemsPresent:** A bit flag that indicates extra data items are present.

**oneByteXYCoordCount:** A bit flag indicating that the values of nXorus and nYorus are packed into a single byte.

**controlledYCoords:** A bit flag indicating that there are one or more controlled Y coordinates.

**controlledXCoords:** A bit flag indicating that there are one or more controlled X coordinates.

**nXorus:** An unsigned integer indicating the number of controlled X coordinates

**nYorus:** An unsigned integer indicating the number of controlled Y coordinates.

**twoByteCoord[]:** A bit flag indicating the format and method of interpreting a controlled coordinate value. If this bit flag is clear, the controlled coordinate value is represented by one byte which is interpreted as an unsigned coordinate value in outline resolution units relative to the preceding coordinate value. In the case of the first controlled coordinate value, the preceding value is deemed to be at the origin. If this bit flag is set, the controlled coordinate value is represented by two bytes whose integer value is interpreted as an absolute signed coordinate value in outline resolution units.

**oruValue:** A signed integer representing a controlled coordinate value in X or Y. Controlled coordinate values must be in ascending order within each dimension. Controlled X coordinates are listed first, controlled Y coordinates are listed second.

**nExtraItems:** An unsigned integer that indicates the number of extra data items present. Extra data items added to a simple glyph program string contain data that will be ignored by earlier versions of the PFR interpreter and used by later versions of the PFR interpreter.

**extraItemSize:** An unsigned integer indicating the size in bytes of one extra data item. The size includes only the extra item data following the extraItemType field.

**extraItemType:** An unsigned integer indicating the type of extra item data present. Two extra data item types (values 1 - 2) have been defined for simple glyph program strings at this time.

**extraItemData:** One byte of extra item data. This data is interpreted in accordance with the values of extraItemType defined for simple glyph program strings. All undefined extra item types will be ignored.

**secondaryStrokeInfo():** A block of data representing one or more secondary stroke definitions.

**secondaryEdgeInfo():** A block of data representing one or more secondary edge definitions.

**glyphOutlineRecord():** A block of data representing one segment of an outline definition. Four types of glyph outline records are defined:

moveTo()

lineTo()

curveTo()

endGlyph()

The first glyph record must be a moveTo record. This defines the start point of the first contour. The shape of a contour is defined by a sequence of lineTo and curveTo records in any order. The outline shape defining a contour should not be self-intersecting. Each successive moveTo record terminates the preceding contour and starts a new one. If the end point of the previous contour is not coincident with its start point, the contour is closed as if a lineTo record back to the start point of the contour had been included. The last contour must be terminated by an endGlyph record.

Glyph outline records make use of the concept of an argument format that defines one of four possible formats for specifying an X or Y coordinate. Argument formats have the following meaning:

**Table A-14. — X - Coordinate Argument Format**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| xArg(xArgFormat) { | | |
| switch (xArgFormat){ | | |
| case 0: xIndex | 8 | uimsbf |
| break | | |
| case 1: xValue | 16 | tcimsbf |
| break | | |
| case 2: xIncrement | 8 | tcimsbf |
| break | | |
| case 3: | | |
| } | | |
| } | | |

**xIndex:** An unsigned integer representing the index in the controlled X coordinate table at which the value is found. Note that only controlled coordinates representing the edges of primary strokes may be used in this manner.

**xValue:** A signed integer representing the X coordinate of the point.

**xIncrement:** A signed integer representing the change in the X coordinate value relative to the X coordinate of the preceding point.

**Table A-15. — Y - Coordinate Argument Format**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| yArg(yArgFormat) { | | |
| switch (yArgFormat){ | | |
| case 0: yIndex | 8 | uimsbf |
| break | | |
| case 1: yValue | 16 | tcimsbf |
| break | | |
| case 2: yIncrement | 8 | tcimsbf |
| break | | |
| case 3: | | |
| } | | |
| } | | |

**yIndex:** An unsigned integer representing the index in the controlled Y coordinate table at which the value of the Y coordinate may be found. Note that only controlled coordinates representing the edges of primary horizontal strokes may be used in this manner.

**yValue:** A signed integer representing the Y coordinate of the point.

**yIncrement:** A signed integer representing the change in the Y coordinate value relative to the Y coordinate of the preceding point.

### A.7.1 MoveTo glyph record

The moveTo glyph record starts a new contour at a specified point.

Its structure is:

**Table A-16. — Move To Glyph Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| moveTo() { | | |
| moveOp | 3 | uimsbf |
| isOutsideContour | 1 | bslbf |
| yArgFormat | 2 | uimsbf |
| xArgFormat | 2 | uimsbf |
| xArg(xArgFormat) | | |
| yArg(yArgFormat) | | |
| } | | |

**moveOp:** An unsigned integer constant with value 2; this field provides, together with the subsequent isOutsideContour field, a unique identification of a moveTo glyph record.

**isOutsideContour:** A bit flag that is set to indicate that the contour is an outside contour whose direction is counterclockwise

**yArgFormat:** An unsigned integer that defines the encoding format of a Y coordinate value. In the case of the first move in a glyph program string, the preceding Y coordinate value is deemed to have a value of 0.

**xArgFormat:** An unsigned integer that defines the encoding format of a X coordinate value. In the case of the first move in a glyph program string, the preceding X coordinate value is deemed to have a value of 0.

**xArg:** The X coordinate of the start point of the contour as defined above.

**yArg:** The Y coordinate of the start point of the contour as defined above.

### A.7.2 LineTo glyph record

The lineTo glyph record continues a contour from the current point in a straight line to a specified point. Its structure is as follows.

**Table A-17. — Line To Glyph Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| lineTo() { | | |
| lineOp | 4 | uimsbf |
| yArgFormat | 2 | uimsbf |
| xArgFormat | 2 | uimsbf |
| xArg(xArgFormat) | | |
| yArg(yArgFormat) | | |
| } | | |

**lineOp:** An unsigned integer constant with value 1

**yArgFormat:** An unsigned integer that defines the encoding format of a Y coordinate value.

**xArgFormat:** An unsigned integer that defines the encoding format of a X coordinate value.

**xArg:** The X coordinate of the end point of the line as defined above.

**yArg:** The Y coordinate of the end point of the line as defined above.

### A.7.3 CurveTo glyph record

The curveTo glyph record continues a contour from the current point in a curved outline to a specified point. The shape of the intervening curve is a cubic bezier and is defined by a pair of curve control points. Its structure is as follows.

**Table A-18. — Curve To Glyph Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| curveTo() { | | |
| curveOp | 1 | uimsbf |
| curveDepth | 3 | uimsbf |
| y1ArgFormat | 2 | uimsbf |
| x1ArgFormat | 2 | uimsbf |
| xArg(x1ArgFormat) | | |
| yArg(y1ArgFormat) | | |
| y3ArgFormat | 2 | uimsbf |
| x3ArgFormat | 2 | uimsbf |
| y2ArgFormat | 2 | uimsbf |
| x1ArgFormat | 2 | uimsbf |
| xArg(x2ArgFormat) | | |
| yArg(y2ArgFormat) | | |
| xArg(x3ArgFormat) | | |
| yArg(y3ArgFormat) | | |
| } | | |

**curveOp:** An unsigned integer constant with value 1; this field provides, together with the subsequent curveDepth field, a unique identification of a curveTo glyph record.

**curveDepth:** An unsigned integer indicating the number of recursive subdivisions required to result in a polygonal representation with an error less than one half of an outline resolution unit.

**y1ArgFormat:** An unsigned integer that defines the encoding format of the Y coordinate of the first curve control point.

**x1ArgFormat:** An unsigned integer that defines the encoding format of the X coordinate of the first curve control point.

**xArg(x1ArgFormat):** The X coordinate of the first curve control point.

**yArg(y1ArgFormat):** The Y coordinate of the first curve control point.

**y3ArgFormat:** An unsigned integer that defines the encoding format of the Y coordinate of the curve end point.

**x3ArgFormat:** An unsigned integer that defines the encoding format of the X coordinate of the curve end point.

**y2ArgFormat:** An unsigned integer that defines the encoding format of the Y coordinate of the second curve control point.

**x2ArgFormat:** An unsigned integer that defines the encoding format of the X coordinate of the second curve control point.

**xArg(x2ArgFormat):** The X coordinate of the second curve control point.

**yArg(y2ArgFormat):** The Y coordinate of the second curve control point.

**xArg(x3ArgFormat):** The X coordinate of the curve end point.

**yArg(y3ArgFormat):** The Y coordinate of the curve end point.

### A.7.4 EndGlyph record

The endGlyph record terminates a contour with a line back to the start point of the contour. Its structure is as follows.



**Table A-19. — End Glyph Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| endGlyph() { | | |
| endGlyphOp | 8 | uimsbf |
| } | | |

**endGlyphOp:** An unsigned integer constant with value 0

### A.7.5 Short-form glyph records

In addition to these standard glyph outline records, there are several special-case versions to provide more compact representations of common shapes:

### A.7.6 hLineTo glyph record

For horizontal straight lines that end on a X controlled coordinate, the hLineTo glyph outline record is provided. Its structure is as follows.

**Table A-20. — Horizontal Line To Glyph Outline Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| hLineTo() { | | |
| hLineOp | 4 | uimsbf |
| xIndex | 4 | uimsbf |
| } | | |

**hLineop:** an unsigned integer constant with value 2.

**xIndex:** an unsigned integer indicating the index into the table of controlled X coordinates at which the X coordinate of the end point of the line is found. The first entry of this table has index value zero.

### A.7.7 vLineTo glyph record

For vertical straight lines that end on a controlled X coordinate, the vLineTo glyph outline record is provided. Its structure is as follows.

**Table A-21. — Vertical Line To Glyph Outline Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| vLineTo() { | | |
| vLineOp | 4 | uimsbf |
| yIndex | 4 | uimsbf |
| } | | |

**vLineOp:** an unsigned integer constant with value 3.

**yIndex:** an unsigned integer indicating the index into the table of controlled Y coordinates at which the Y coordinate of the end point of the line is found. The first entry of this table has index value zero.

### A.7.8 hvCurveTo glyph record

For curves that start in a horizontal direction and end in a vertical direction along a controlled X coordinate, the hvCurveTo glyph outline record is provided. Its structure is as follows.

**Table A-22. — Horizontal to Vertical Curve Glyph Outline Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| hvCurveTo() { | | |
| hvCurveOp | 4 | uimsbf |

| | | |
|---|---|---|
| zero | 1 | bslbf |
| curveDepth | 3 | uimsbf |
| x1Increment | 8 | tcimsbf |
| xIndex | 4 | uimsbf |
| y2Increment | 8 | tcimsbf |
| y3Increment | 8 | tcimsbf |
| } | | |

**hvCurveOp:** an unsigned integer constant with value 6

**zero:** A bit flag that shall be set to zero.

**curveDepth:** An unsigned integer indicating the number of recursive subdivisions required to result in a polygonal representation with an error less than one half of an outline resolution unit.

**x1Increment:** A signed integer representing the X coordinate of the first curve control point relative to the start point of the curve.

**xIndex:** an unsigned integer indicating the index into the table of controlled X coordinates at which the X coordinate of the second control point and the end point of the curve is found. The first entry of this table has index value zero.

**y2Increment:** A signed integer representing the Y coordinate of the second curve control point relative to the first curve control point.

**y3Increment:** A signed integer representing the Y coordinate of the end point of the curve relative to the second curve control point.

### A.7.9 vhCurveTo glyph record

For curves that start in a vertical direction and end in a horizontal direction along a controlled Y coordinate, the vhCurveTo glyph outline record is provided. Its structure is as follows.

**Table A-23. — Vertical to Horizontal Curve Glyph Outline Record**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| vhCurveTo() { | | |
| vhCurveOp | 4 | uimsbf |
| zero | 1 | bslbf |
| curveDepth | 3 | uimsbf |
| y1Increment | 8 | tcimsbf |
| x2Increment | 8 | tcimsbf |
| yIndex | 4 | uimsbf |
| x3Increment | 8 | tcimsbf |
| } | | |

**vhCurveOp:** an unsigned integer constant with value 7.

**zero:** A bit flag that shall be set to zero.

**curveDepth:** An unsigned integer indicating the number of recursive subdivisions required to result in a polygonal representation with an error less than one half of an outline resolution unit.

**y1Increment:** A signed integer representing the Y coordinate of the first curve control point relative to the start point of the curve.

**x2Increment:** A signed integer representing the X coordinate of the second curve control point relative to the first curve control point.

**yIndex:** an unsigned integer indicating the index into the table of controlled Y coordinates at which the Y coordinate of the second control point and the end point of the curve is found.

**x3Increment:** A signed integer representing the X coordinate of the end point of the curve relative to the second curve control point.



### A.7.10 Secondary stroke definitions

Primary strokes cannot be mutually overlapping. Secondary strokes that overlap primary strokes may be other secondary strokes that are encoded into secondary stroke information. Secondary strokes that overlap a primary stroke are positioned relative to the primary stroke after the primary stroke has been positioned. Secondary stroke information is structured as a type 1 extra data item. The format for a secondary stroke information is as follows.

**Table A-24. — Secondary Stroke Information Extra Data Item**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| secondaryStrokeInfo() { | | |
| nVertSecStrokes | 8 | uimsbf |
| for (i = 0; i < nVertSecStrokes; i++){ | | |
| leftEdge[i] | 16 | tcimsbf |
| rightEdge[i] | 16 | tcimsbf |
| } | | |
| nHorizSecStrokes | 8 | uimsbf |
| for (i = 0; i < nHorizSecStrokes; i++) | | |
| { | | |
| bottomEdge[i] | 16 | tcimsbf |
| topEdge[i] | 16 | tcimsbf |
| } | | |
| } | | |

**nVertSecStrokes:** An unsigned integer representing the number of secondary vertical strokes defined for the current simple glyph.

**leftEdge[]:** A signed integer representing the X coordinate of the left edge of a secondary vertical stroke in outline resolution units.

**rightEdge[]:** A signed integer representing the X coordinate of the right edge of a secondary vertical stroke in outline resolution units.

**nHorizSecStrokes:** An unsigned integer representing the number of secondary horizontal strokes defined for the current simple glyph.

**bottomEdge[]:** A signed integer representing the Y coordinate of the lower edge of a secondary horizontal stroke in outline resolution units.

**topEdge[]:** A signed integer representing the Y coordinate of the upper edge of a secondary horizontal stroke in outline resolution units.

Because the maximum size of a secondary stroke definition item is 255 bytes, the maximum number of secondary strokes that may be defined in one extra data item is 63. Secondary vertical strokes must be in increasing order of their left edge. Secondary horizontal strokes must be in increasing order of their lower edge.

### A.7.11 Secondary edge definitions

When the edge of a stroke is represented by a shallow curve or other irregularity, it is often desirable to straighten the outline at small sizes and low resolutions. A secondary edge may be defined relative to any stroke edge (its parent). At small sizes and low resolutions, the secondary edge is snapped to the position of its parent. This has the effect of squeezing outline points between the parent edge and the secondary edge onto the primary edge thus resulting in a locally straightened outline. Either edge of any primary or secondary stroke may have one or two secondary edges associated with it. Two edges allow the squeezing operation to take place from both sides of the parent edge. A secondary edge is a generalization of the flex mechanism used in Type 1 fonts which is restricted to certain specific curve structures. Secondary edges may be used with any shape that should be flattened at small sizes. Secondary edge information is structured as a type 2 extra data item. The format for secondary edge information is as follows.

**Table A-25. — Secondary Edge Information Extra Data Item**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| secondaryEdgeInfo() { | | |
| nVertSecEdges | 8 | uimsbf |

| | | |
|---|---|---|
| for (i = 0; i < nVertSecEdges; i++){ | | |
| secEdgeDef() | | |
| } | | |
| nHorizSecEdges | 8 | uimsbf |
| for (i = 0; i < nHorizSecEdges; i++){ | | |
| secEdgeDef() | | |
| } | | |
| } | | |

**nVertSecEdges:** An unsigned integer indicating the number of vertical secondary edge definitions provided.

**nHorizSecEdges:** An unsigned integer indicating the number of horizontal secondary edge definitions provided.

The briefest format for a secondary edge definition (either horizontal or vertical) is as follows.

**Table A-26. — Simplified Secondary Edge Definition**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| secEdgeDef() { | | |
| secEdgeFormat | 1 | bslbf |
| deltaIndex | 3 | uimsbf |
| deltaOrus | 4 | tcimsbf |
| } | | |

**secEdgeFormat:** A bit flag with a constant value of 0

**deltaIndex:** An unsigned integer in the range 0 to 7 representing the index of the parent edge relative to the index of the parent edge of the immediately preceding secondary edge. In the case of the first edge in each dimension, deltaIndex is interpreted absolutely as the index of the parent edge.

**deltaOrus:** A signed integer in the range -8 to +7 representing the position of the secondary edge relative to its parent edge in units of character outline resolution units.

In this format, a standard secondary edge snap threshold of 1 pixel is assumed.

A more general (and longer) format for a secondary edge definition is as follows.

**Table A-27. — General Secondary Edge Definition**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| secEdgeDef() { | | |
| secEdgeFormat | 1 | bslbf |
| threshFlag | 1 | bslbf |
| index | 6 | uimsbf |
| if (threshFlag == 0) | | |
| thresh | 8 | uimsbf |
| deltaOrus | 8 | tcimsbf |
| if (deltaOrus == 0) | | |
| deltaOrus | 16 | tcimsbf |
| } | | |

**secEdgeFormat:** A bit flag with a constant value of 1

**threshFlag:** A bit flag that indicates how the threshold value is represented. If set, a standard threshold value of 1 pixel is assumed. If clear, an explicit value is provided.

**index:** An unsigned integer in the range 0 to 63. A value of zero indicates that the index of the parent coordinate is explicitly specified. Any other value indicates that the index of the parent coordinate is index - 1.



**thresh:** An unsigned integer representing the threshold at which the secondary edge should be snapped to its parent. The units are 1/16 pixel.

**deltaOrus:** A signed integer representing the position of the secondary edge relative to its parent in units of character outline resolution units.

## A.8 Compound glyph program strings

A compound glyph program string is constructed out of one or more simple glyph program strings. Each of the elements may be independently scaled and positioned in the process of constructing the compound glyph.

The structure of a compound glyph program string is as follows.

**Table A-28. — Compound Glyph Program String**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| compoundGps() { | | |
| isCompoundGlyph | 1 | bslbf |
| extraItemsPresent | 1 | bslbf |
| nElements | 6 | uimsbf |
| if (extraItemsPresent) | | |
| { | | |
| nExtraItems | 8 | uimsbf |
| for (i = 0; i < nExtraItems; i++) | | |
| { | | |
| extraItemSize | 8 | uimsbf |
| extraItemType | 8 | uimsbf |
| switch(extraItemType){ | | |
| default: | | |
| for (j = 0; j < extraItemSize; j++){ | | |
| extraItemData | 8 | uimsbf |
| } | | |
| break; | | |
| } | | |
| } | | |
| for (i = 0; i < nElements; i++){ | | |
| threeByteGpsOffset | 1 | bslbf |
| twoByteGpsSize | 1 | bslbf |
| yScalePresent | 1 | bslbf |
| xScalePresent | 1 | bslbf |
| yPosFormat | 2 | uimsbf |
| xPosFormat | 2 | uimsbf |
| if (xScalePresent) | | |
| xScale | 16 | tcimsbf |
| if (yScalePresent) | | |
| yScale | 16 | tcimsbf |
| switch(xPosFormat) | | |
| { | | |
| case 1: xPos | 16 | tcimsbf |
| break | | |
| case 2: xPos | 8 | tcimsbf |
| break | | |
| } | | |
| switch(yPosFormat) | | |
| { | | |
| case 1: yPos | 16 | tcimsbf |
| break | | |
| case 2: yPos | 8 | tcimsbf |
| break | | |
| } | | |
| if (twoByteGpsSize) | | |

```
                    gpsSize                     16          uimsbf
                else
                    gpsSize                      8          uimsbf
                if (threeByteGpsOffset)
                    gpsOffset                   24          uimsbf
                else
                    gpsOffset                   16          uimsbf
                }
            }
        }
```

**isCompoundGlyph:** A bit flag with a constant value to 1 to indicate that the glyph program string should be interpreted as a compound glyph program string.

**extraItemsPresent:** A bit flag that indicates extra data items are present. This should be set to zero for the current version.

**nElements:** An unsigned integer indicating the number of elements in the compound character.

**nExtraItems:** An unsigned integer that indicates the number of extra data items present. Extra data items added to a compound glyph program string contain data that will be ignored by earlier versions of the PFR interpreter and may be used by later versions of the PFR interpreter.

**extraItemSize:** An unsigned integer indicating the size in bytes of one extra data item. The size includes only the extra item data following the extraItemType field.

**extraItemType:** An unsigned integer indicating the type of extra item data present. No extra data item types have been defined for compound glyph program strings at this time

**extraItemData:** One byte of extra item data. This data is interpreted in accordance with the values of extraItemType defined for compound glyph program strings. All undefined extra item types will be ignored.

**threeByteGpsOffset:** A bit flag that indicates, if set, that the gpsOffset value is defined as a 3-byte integer rather than by 2-byte integer

**twoByteGpsSize**: A bit flag that indicates, if set, that the value of gpsSize is defined as a 2-byte integer rather than as a single-byte integer.

**yScalePresent:** A bit flag that indicates, if set, that an explicit value of xScale is defined.

**xScalePresent:** A bit flag that indicates, if set, that an explicit value of yScale is defined.

**yPosFormat:** An unsigned integer that indicates how the value of yPos is defined. A value of 1 indicates that it is defined as a 2-byte absolute value; a value of 2 indicates that it is defined as a single-byte value relative to the previous value of yPos; a value of 3 indicates that it is identical to the previous value of yPos.

**xPosFormat:** An unsigned integer that indicates how the value of xPos is defined. A value of 1 indicates that it is defined as a 2-byte absolute value; a value of 2 indicates that it is defined as a single-byte value relative to the previous value of xPos; a value of 3 indicates that it is identical to the previous value of xPos.

**xScale:** A signed integer representing the scale factor to be applied to the glyph element in the X dimension. This field is in units of 1/4096.

**yScale:** A signed integer representing the scale factor to be applied to the glyph element in the Y dimension. This field is in units of 1/4096.

**xPos:** A signed integer representing the amount by which the glyph element should be shifted in the X dimension. This field is in units of character outline resolution units.

**yPos:** A signed integer representing the amount by which the glyph element should be shifted in the Y dimension. This field is in units of character outline resolution units.

**gpsSize:** An unsigned integer representing the size in bytes of the glyph program string defining the glyph element.

**gpsOffset:** An unsigned integer representing the byte offset of the first byte of the glyph program string that defines the glyph element. The offset is relative to the first byte of the first glyph program string in the glyph program string section.



### A.8.1 Bitmap glyph program string

A bitmap glyph program string defines the image of a glyph in the form of a bitmap. Its structure is as follows.

**Table A-29. — Bitmap Glyph Program String**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| bitmapGps() { | | |
| imageFormat | 2 | uimsbf |
| escapementFormat | 2 | uimsbf |
| sizeFormat | 2 | uimsbf |
| positionFormat | 2 | uimsbf |
| switch(positionFormat) | | |
| { | | |
| case 0: | | |
| xPos | 4 | tcimsbf |
| yPos | 4 | tcimsbf |
| break | | |
| case 1: | | |
| xPos | 8 | tcimsbf |
| yPos | 8 | tcimsbf |
| break | | |
| case 2: | | |
| xPos | 16 | tcimsbf |
| yPos | 16 | tcimsbf |
| break | | |
| case 3: | | |
| xPos | 24 | tcimsbf |
| yPos | 24 | tcimsbf |
| break | | |
| } | | |
| switch(sizeFormat) | | |
| { | | |
| case 0: | | |
| break | | |
| case 1: | | |
| xSize | 4 | uimsbf |
| ySize | 4 | uimsbf |
| break; | | |
| case 2: | | |
| xSize | 8 | uimsbf |
| ySize | 8 | uimsbf |
| break; | | |
| case 3: | | |
| xSize | 16 | uimsbf |
| ySize | 16 | uimsbf |
| break; | | |
| } | | |
| switch(escapementFormat) | | |
| { | | |
| case 0: | | |
| break: | | |
| case 1: | | |
| setWidth | 8 | tcimsbf |
| break; | | |
| case 2: | | |
| setWidth | 16 | tcimsbf |
| break; | | |
| case 3: | | |
| setWidth | 24 | tcimsbf |

| | | |
|---|---|---|
| | break;<br>} | |
| imageData<br>} | | variable |

**imageFormat:** An unsigned integer that indicates how the bitmap image is represented.

A value of 0 indicates that the image is stored directly as a bitmap fully packed with no padding between rows.

A value of 1 indicates that the image is run-length encoded in which each byte specifies the unsigned number of white bits in the most significant 4 bits and the unsigned number of following black bits in the least significant 4 bits. A run of more than 15 bits of the same color is handled by multiple bytes Adjacent rows are encoded together without regard to the end of each row. Trailing white bits must be encoded.

A value of 2 indicates that the image is run-length encoded in which each pair of bytes specifies the unsigned number of white bits in the first byte and the unsigned number of following black bits in the second byte. A run of more than 255 bits of the same color is handled by multiple pairs of bytes. Adjacent rows are encoded together without regard to the end of each row. Trailing white bits must be encoded.

A value of 3 is undefined.

**escapementFormat:** An unsigned integer that indicates how the escapement value is represented.

A value of 0 indicates that no bitmap escapement data is included and that the linearly scaled outline width should be used without rounding.

A value of 1 indicates that the bitmap escapement is represented by a signed single-byte value in units of whole pixels.

A value of 2 indicates that the bitmap escapement is represented by a signed 2-byte value in units of 1/256 pixels.

A value of 3 indicates that the bitmap escapement is represented by a signed 3-byte value in units of 1/256 pixels.

**sizeFormat:** An unsigned integer that indicates how the dimensions of the bitmap image are represented.

A value of 0 indicates that that bitmap image is blank and no image data is present.

A value of 1 indicates that the width and the height of the bitmap image are each represented by an unsigned 4-bit value in units of whole pixels.

A value of 2 indicates that the width and the height of the bitmap image are each represented by an unsigned 8-bit value in units of whole pixels.

A value of 3 indicates that the width and the height of the bitmap image are each represented by an unsigned 2-byte value in units of whole pixels.

**positionFormat:** An unsigned integer that indicates how the (x, y) position of the first pixel in the bitmap image is represented.

A value of 0 indicates that the X and the Y coordinates are each represented by a signed 4-bit value in units of whole pixels.

A value of 1 indicates that the X and the Y coordinates are each represented by a signed single-byte value in units of whole pixels.

A value of 2 indicates that the X and the Y coordinates are each represented by a signed 2-byte value in units of 1/256 pixels.

A value of 3 indicates that the X and the Y coordinates are each represented by a signed 3-byte value in units of 1/256 pixels.

**xPos:** A signed integer representing the X position of the first pixel in each row of the image. The position is in units of pixels and is relative to the character origin.

**yPos:** A signed integer representing the Y position of the first row of the image. If the value of pfrInvertBitmap in the PFR header is 0, the position refers to the start of the lowest row of pixels in the character image. If the value of pfrInvertBitmap in the PFR header is 1, the position refers to the start of the highest row of pixels in the character image.

**xSize:** An unsigned integer representing the width of the bitmap image in pixels.



**ySize:** An unsigned integer representing the height of the bitmap image in pixels.

**setWidth:** A signed integer representing the distance in pixels (or 1/256 pixels depending upon the value of escapementFormat) the current rendering position should be moved by prior to imaging the next character. If the value of verticalEscapement in the parent physical font record is 1, the direction of the escapement vector is vertical. Otherwise, it is horizontal.

**imageData:** This data is interpreted depending upon the value of imageFormat.

## A.9 Portable font resource trailer

The PFR trailer block shall be the last block of data in the Portable Font Resource. Its primary use it to facilitate the location of the start of a PFR that ends at the end of a file. Its structure is:

**Table A-30. — Portable Font Resource Trailer**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| pfrTrailer() { | | |
| pfrSize | 24 | uimsbf |
| pfrTrailerSig | 40 | bslbf |
| } | | |

**pfrSize:** An unsigned integer representing the total size of the PFR in bytes.

**pfrTrailerSig:** A bit pattern used as a PFR trailer signature. It shall have the constant value 0x2450465224 representing the string “$PFR$”.

## A.10 Kerning data

Kerning data for a physical font is stored as one or more extra data items attached to the physical font for which the kerning data applies. Track and pair kerning data are stored in separate ExtraItem types.

### A.10.1 Pair Kerning Data

The format of a pair kerning data block is as follows:

**Table A-31. — Pair Kerning Data**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| pairKernData() { | | |
| extraItemSize 8 | uimsbf | |
| extraItemType 8 | uimsbf | |
| nKernPairs | 8 | uimsbf |
| baseAdjustment | 16 | tcimsbf |
| zeros | 6 | bslbf |
| twoByteAdjValues | 1 | bslbf |
| twoByteCharCodes | 1 | bslbf |
| for (i = 0; i < nKernPairs; i++){ | | |
| if (twoByteCharCodes) { | | |
| charCode1 | 16 | uimsbf |
| charCode2 | 16 | uimsbf |
| } | | |
| else { | | |
| charCode1 | 8 | uimsbf |
| charCode2 | 8 | uimsbf |
| } | | |
| if (twoByteAdjustment) | | |
| adjustment | 16 | tcimsbf |
| else | | |
| adjustment | 8 | uimsbf |
| } | | |
| } | | |



**extraItemSize:** This is the number bytes of data in the extra data item. This does not include the two bytes for the extraItemSize and extraItemType.

**extraItemType:** This is a constant with a value of 4. It identifies the extra data item as kerning pair data.

**nKernPairs:** The number of kerning pairs included in the table.

**baseAdjustment:** The base value of the adjustment, in metrics resolution units, relative to which all adjustment values are encoded. It is primarily intended to facilitate compaction from the use of the single byte adjustment values. A positive value indicates an increased spacing, a negative value a reduced spacing.

**zeros:** a concatenation of 6 bits all set to zero.

**twoByteAdjValues:** a bit flag defining how all kerning adjustment values are encoded. A zero indicates that every kerning adjustment value is encoded as a one byte unsigned integer relative to the base adjustment. A one indicates that every kerning adjustment value is encoded as a 2-byte two-complement integer relative to the base adjustment.

**twoByteCharCodes:** a bit flag defining how all character codes are encoded. A zero indicates that each character code is encoded as an unsigned byte. A one indicates that each character code is encoded as 2-byte unsigned integer.

**charCode1:** the character code for the left character of each kerning pair.

**charCode2:** the character code for the right character of each kerning pair.

**adjustment:** the amount by which the escapement is to be adjusted between the left and right characters of the kerning pair in metrics resolution units. The adjustment is positive to increase the spacing, negative to reduce the spacing. The adjustment is relative to the value of baseAdjustment for the block of kerning data.

The order of the kerning pair records is required to be in increasing order of charCode1. Groups of records with a common value of charCode1 are required to be in increasing order of charCode2.

Because the maximum number of bytes in an extra data item is limited to 255, there is a limit on the number of kerning pairs that may be included in one extra data item. Multiple extra data items may be used to overcome this limit. The order of such multiple items must be in ascending order of character pair codes. This allows the search for a specific character code pair to scan the first entry in each type 4 extra data item to determine which block contains the pair.



## A.10.2 Track Kerning Data

The format of a track kerning data block is as follows:

**Table A-32. — Track Kerning Data**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| `trackKernData() {` | | |
| `  extraItemSize  8` | uimsbf | |
| `  extraItemType  8` | uimsbf | |
| `  nKernTracks` | 8 | uimsbf |
| `  for (I = 0; i < nKernTracks; i++){` | | |
| `    degree` | 16 | uimsbf |
| `    minPointSize` | 16 | uimsbf |
| `    minAdjust` | 16 | tcimsbf |
| `    maxPointSize` | 16 | uimsbf |
| `    maxAdjust` | 16 | tcimsbf |
| `    }` | | |
| `  }` | | |

**extraItemSize:** This is the number bytes of data in the extra data item. This does not include the two bytes for the extraItemSize and extraItemType.

**extraItemType:** This is a constant with a value of 5. It identifies the extra data item as kerning track data.

**nKernTracks:** The number of track kerning entries.

**degree:** This identifies the amount of track kerning. Standard values are 1 for light kerning, 2 for medium kerning and 3 for tight kerning. All other values are reserved.

**minPointSize:** This is the minimum point size at which the track kerning takes place for the current track. Its value is in units of points.

**minAdjust:** This is the spacing adjustment to be applied between each pair of characters at the minimum point size. Its value is in metrics resolution units. A positive value indicates an increase in spacing; a negative value indicates a decrease in spacing.

**maxPointSize:** This is the maximum point size at which the track kerning takes place for the current track. Its value is in units of points.

**maxAdjust:** This is the spacing adjustment to be applied between each pair of characters at the maximum point size. Its value is in metrics resolution units. A positive value indicates an increase in spacing; a negative value indicates a decrease in spacing.

Note : The 255 byte limit on the size of an extra data item is not significant as this allows about 25 kerning tracks to be included.

# Annex B
(normative)
# Coding of Linear Audio

## B.1 Coding format for Linear Audio

The coding of linear audio in ISO/IEC 16500 is based on the AIFF-C format. ISO/IEC 16500 applies the commonly used, more restrictive but AIFF-C compliant format as specified in this Annex. For ISO/IEC 16500 the Form chunk, Format Version chunk, Extended Common chunk, and the Sound Data chunk shall each appear only once and in the order shown below in a linear audio sample. Any Private chunks shall appear after the required chunks and are not restricted.

**Table B-1. — Linear audio**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| Linear_audio (){ | | |
| ckID_FC | 32 | bslbf |
| ckSize_FC | 32 | uimsbf |
| formType_FC | 32 | bslbf |
| Format_Version_Chunk () | | |
| Extended_Common_Chunk () | | |
| Sound_Data_Chunk () | | |
| for (i=N; i<ckSize_FC; i++) { | | |
| Private_Chunk () | | |
| } | | |
| } | | |
| | | |
| Format_Version_Chunk(){ | | |
| ckID_FVC | 32 | bslbf |
| ckSize_FVC | 32 | uimsbf |
| version_date | 32 | uimsbf |
| } | | |
| | | |
| Extended_Commom_Chunk(){ | | |
| ckID_ECC | 32 | bslbf |
| ckSize_ECC | 32 | uimsbf |
| numChannels | 16 | uimsbf |
| numSampleFrames | 32 | uimsbf |
| sampleSize | 16 | uimsbf |
| sampleRate | 80 | fpvsbf |



```
        compressionType                          32              bslbf
        compNameLength                           8               uimsbf
        compressionName                          compNameLength*8 bslbf
        if ((compNameLength % 2) == 0){
            compNamePadbyte                      8               bslbf
        }
}


Sound_Data_Chunk(){
        ckID_SDC                                 32              bslbf
        ckSize_SDC                               32              uimsbf
        offset                                   32              uimsbf
        blockSize                                32              uimsbf
        for (i=0; i<numSampleFrames; i++){
            for (j=0; j<numChannels; j++){
                sound_data                       sampleSize      uimsbf
            }
        }
        if (((numChannels % 2)==1) &&
            ((numSampleFrames % 2)==1) &&
            (sampleSize==8)){
            pad_byte                             8               uimsbf
        }
}


Private_Chunk (){
        ckID_PC                                  32              bslbf
        ckSize_PC                                32              uimsbf
        for(i=0; i<ckSize_PC; i++){
            reserved                             8               uimsbf
        }
}
```



**ckID_FC:** A byte string which specifies the chunk type as the Form chunk. It is used as a header for the entire linear audio sample. It shall be set to the constant value 0x464F524D, which is the ASCII value for 'FORM'.

**ckSize_FC:** An unsigned integer indicating the size (in bytes) of the entire linear audio file after the ckSize_FC field.

**formType_FC:** A byte string which defines the format of the linear audio file as an AIFF-C file format. This field shall be set to the constant value 0x41494643, which is the ASCII value for 'AIFC'.

**ckID_FVC:** A byte string which specifies the chunk type as the Format Version chunk. This field shall be set to the constant value 0x46564552, which is the ASCII value for 'FVER'.

**ckSize_FVC:** An unsigned integer indicating the size (in bytes) of the Format Version chunk. This field shall be set to the constant value 0x00000004.

**version_date:** An unsigned integer indicating the creation date of the AIFF-C version used to code this linear audio file. This field shall be set to the constant value 0xA2805140, which represents the date May 23, 1990, 2:40 PM.

**ckID_ECC:** A byte string which specifies the chunk type as the Extended Common chunk. This field shall be set to the constant value 0x434F4D4D, which is the ASCII value for 'COMM'.

**ckSize_ECC:** An unsigned integer indicating the size (in bytes) of the Extended Common chunk. For ISO/IEC 16500, this field shall be set to the constant value 0x00000026.

**numChannels:** An unsigned integer indicating the number of audio channels used in the linear audio sample. This field is restricted to values of 1 or 2 indicating mono or stereo audio, respectively, for ISO/IEC 16500.

**numSampleFrames:** An unsigned integer indicating the number of sample frames in the linear audio sample. The total number of sample points is figured numChannels * numSampleFrames.

**sampleSize:** An unsigned integer indicating the number of bits in each sample point. This field can contain any integer from 1 to 32, but is restricted to values of 8 or 16 for ISO/IEC 16500.

**sampleRate:** An 80-bit floating point value indicating the rate at which the sound was sampled. The format of the floating point value is double-extended precision floating point, which includes one sign bit, 15-bit exponent, and 64-bit mantissa according to the IEEE 96-bit floating point representation (using only 15 bits instead of 31 for the exponent). For ISO/IEC 16500, the only valid sample rates are show in the table below.

**Table B-2. — Sample rate assignments**

| *Sample Rate* | *Hex Representation* |
|---|---|
| 16.000 kHz | 0x400CFA00000000000000 |
| 22.050 kHz | 0x400DAC44000000000000 |
| 24.000 kHz | 0x400DBB80000000000000 |
| 32.000 kHz | 0x400DFA00000000000000 |
| 44.100 kHz | 0x400EAC44000000000000 |
| 48.000 kHz | 0x400EBB80000000000000 |

The value is represented with sign bit first. ['fpvsbf' stands for 'floating point value sign bit first'.]

**compressionType:** A byte string which indicates the type of compression algorithm, if any, used on the sound data. Compression is not used for ISO/IEC 16500, so this field shall be set to the constant value 0x4E4F4E45, which is the ASCII value for 'NONE'.

**compNameLength:** An unsigned integer which indicates the number of ASCII characters used in compressionName.

**compressionName:** A byte string which contains the compression algorithm ID specified in the compressionType field. Since compression is not used in ISO/IEC 16500, this field shall be set to the constant value 0x6E6F7420636F7072657373656564, which is the ASCII value for 'not compressed'.

**compNamePadbyte:** The constant value 0x00 which is inserted if the value of compNameLength is even. ISO/IEC 16500 requires the pad byte.

**ckID_SDC:** A byte string which specifies the chunk type as the Sound Data chunk. This field shall be set to the constant value 0x53534E44, which is the ASCII value for 'SSND'.

**ckSize_SDC:** An unsigned integer indicating the size (in bytes) of the Sound Data chunk excluding the pad_byte, if included. This value will equal numChannels*numSampleFrames*(sampleSize/8) + 8.

**offset:** An unsigned integer indicating the offset (in bytes) to the beginning of the first sample frame in the chunk data. For ISO/IEC 16500, this field shall be set to 0x00000000.



**blockSize:** An unsigned integer indicating the size (in bytes) of the blocks to which the sound data is aligned. This field is used in conjunction with the offset field for aligning sound data to blocks. For ISO/IEC 16500, this field shall be set to 0x00000000.

**sound_data:** A variable bit field representing a linear audio sample point. The width of this field is equal to sampleSize in the Extended_Commom_Chunk. For two channel audio, the sample points are interleaved left channel first, then right.

**pad_byte:** An unsigned integer with the fixed value of 0x00. A pad_byte shall be included if the number of sound data bytes is odd.

**ckID_PC:** A byte string which specifies the chunk type as a Private chunk. This field can assume any 4 ASCII characters in the range ‘ ‘ (space character) through ‘~’ (i.e. 0x20 through 0x7E) other than those specified for ckID_FC, ckID_FVC, ckID_ECC, and ckID_SDC. Space (ASCII 0x20) cannot precede printing characters, but trailing spaces are allowed.

**ckSize_PC:** An unsigned integer indicating the size (in bytes) of the Private chunk.

*Note : Support of Linear Audio as a real-time stream in future versions of the DAVIC specifications may require concatenation of objects containing linear audio with a play back duration of up to 0.7 second each.*

# Annex C
(normative)
# Default CLUT for single bitmaps with compressed graphics

## C.1 Default CLUT specification

**Table C-1. — Default CLUT entries**

| *Transparency Level* | *Default CLUT entry definition*<br>*Format : clut[n] = [R, G, B], where*<br>*n denotes the clut entry, and*<br>*R,G and B denote the values of the Red, Green and Blue components associated to clut entry* | | |
|---|---|---|---|
| 0 %<br>(fully opaque) | clut[0] = [ 0, 0, 0] | clut[45] = [ 63,191, 0] | clut[90] = [191, 95, 0] |
| | clut[1] = [ 0, 0,127] | clut[46] = [ 63,191,127] | clut[91] = [191, 95,127] |
| | clut[2] = [ 0, 0,255] | clut[47] = [ 63,191,255] | clut[92] = [191, 95,255] |
| | clut[3] = [ 0, 31, 0] | clut[48] = [ 63,223, 0] | clut[93] = [191,127, 0] |
| | clut[4] = [ 0, 31,127] | clut[49] = [ 63,223,127] | clut[94] = [191,127,127] |
| | clut[5] = [ 0, 31,255] | clut[50] = [ 63,223,255] | clut[95] = [191,127,255] |
| | clut[6] = [ 0, 63, 0] | clut[51] = [ 63,255, 0] | clut[96] = [191,159, 0] |
| | clut[7] = [ 0, 63,127] | clut[52] = [ 63,255,127] | clut[97] = [191,159,127] |
| | clut[8] = [ 0, 63,255] | clut[53] = [ 63,255,255] | clut[98] = [191,159,255] |
| | clut[9] = [ 0, 95, 0] | clut[54] = [127, 0, 0] | clut[99] = [191,191, 0] |
| | clut[10] = [ 0, 95,127] | clut[55] = [127, 0,127] | clut[100] = [191,191,127] |
| | clut[11] = [ 0, 95,255] | clut[56] = [127, 0,255] | clut[101] = [191,191,255] |
| | clut[12] = [ 0,127, 0] | clut[57] = [127, 31, 0] | clut[102] = [191,223, 0] |
| | clut[13] = [ 0,127,127] | clut[58] = [127, 31,127] | clut[103] = [191,223,127] |
| | clut[14] = [ 0,127,255] | clut[59] = [127, 31,255] | clut[104] = [191,223,255] |
| | clut[15] = [ 0,159, 0] | clut[60] = [127, 63, 0] | clut[105] = [191,255, 0] |
| | clut[16] = [ 0,159,127] | clut[61] = [127, 63,127] | clut[106] = [191,255,127] |
| | clut[17] = [ 0,159,255] | clut[62] = [127, 63,255] | clut[107] = [191,255,255] |
| | clut[18] = [ 0,191, 0] | clut[63] = [127, 95, 0] | clut[108] = [255, 0, 0] |
| | clut[19] = [ 0,191,127] | clut[64] = [127, 95,127] | clut[109] = [255, 0,127] |
| | clut[20] = [ 0,191,255] | clut[65] = [127, 95,255] | clut[110] = [255, 0,255] |



| | | | |
|---|---|---|---|
| | clut[21] = [ 0,223, 0] | clut[66] = [127,127, 0] | clut[111] = [255, 31, 0] |
| | clut[22] = [ 0,223,127] | clut[67] = [127,127,127] | clut[112] = [255, 31,127] |
| | clut[23] = [ 0,223,255] | clut[68] = [127,127,255] | clut[113] = [255, 31,255] |
| | clut[24] = [ 0,255, 0] | clut[69] = [127,159, 0] | clut[114] = [255, 63, 0] |
| | clut[25] = [ 0,255,127] | clut[70] = [127,159,127] | clut[115] = [255, 63,127] |
| | clut[26] = [ 0,255,255] | clut[71] = [127,159,255] | clut[116] = [255, 63,255] |
| | clut[27] = [ 63, 0, 0] | clut[72] = [127,191, 0] | clut[117] = [255, 95, 0] |
| | clut[28] = [ 63, 0,127] | clut[73] = [127,191,127] | clut[118] = [255, 95,127] |
| | clut[29] = [ 63, 0,255] | clut[74] = [127,191,255] | clut[119] = [255, 95,255] |
| | clut[30] = [ 63, 31, 0] | clut[75] = [127,223, 0] | clut[120] = [255,127, 0] |
| | clut[31] = [ 63, 31,127] | clut[76] = [127,223,127] | clut[121] = [255,127,127] |
| | clut[32] = [ 63, 31,255] | clut[77] = [127,223,255] | clut[122] = [255,127,255] |
| | clut[33] = [ 63, 63, 0] | clut[78] = [127,255, 0] | clut[123] = [255,159, 0] |
| | clut[34] = [ 63, 63,127] | clut[79] = [127,255,127] | clut[124] = [255,159,127] |
| | clut[35] = [ 63, 63,255] | clut[80] = [127,255,255] | clut[125] = [255,159,255] |
| | clut[36] = [ 63, 95, 0] | clut[81] = [191, 0, 0] | clut[126] = [255,191, 0] |
| | clut[37] = [ 63, 95,127] | clut[82] = [191, 0,127] | clut[127] = [255,191,127] |
| | clut[38] = [ 63, 95,255] | clut[83] = [191, 0,255] | clut[128] = [255,191,255] |
| | clut[39] = [ 63,127, 0] | clut[84] = [191, 31, 0] | clut[129] = [255,223, 0] |
| | clut[40] = [ 63,127,127] | clut[85] = [191, 31,127] | clut[130] = [255,223,127] |
| | clut[41] = [ 63,127,255] | clut[86] = [191, 31,255] | clut[131] = [255,223,255] |
| | clut[42] = [ 63,159, 0] | clut[87] = [191, 63, 0] | clut[132] = [255,255, 0] |
| | clut[43] = [ 63,159,127] | clut[88] = [191, 63,127] | clut[133] = [255,255,127] |
| | clut[44] = [ 63,159,255] | clut[89] = [191, 63,255] | clut[134] = [255,255,255] |

| *Transparency Level* | *Default CLUT entry definition*<br>*Format : clut[n] = [R, G, B], where*<br>*n denotes the clut entry, and*<br>*R, G and B denote the values of the Red, Green and Blue components associated to clut entry* | | |
|---|---|---|---|
| 25 % | clut[135] = [ 0, 0, 0] | clut[154] = [ 85, 85,255] | clut[173] = [170,212, 0] |
| | clut[136] = [ 0, 0,255] | clut[155] = [ 85,127, 0] | clut[174] = [170,212,255] |
| | clut[137] = [ 0, 42, 0] | clut[156] = [ 85,127,255] | clut[175] = [170,255, 0] |
| | clut[138] = [ 0, 42,255] | clut[157] = [ 85,170, 0] | clut[176] = [170,255,255] |
| | clut[139] = [ 0, 85, 0] | clut[158] = [ 85,170,255] | clut[177] = [255, 0, 0] |
| | clut[140] = [ 0, 85,255] | clut[159] = [ 85,212, 0] | clut[178] = [255, 0,255] |
| | clut[141] = [ 0,127, 0] | clut[160] = [ 85,212,255] | clut[179] = [255, 42, 0] |
| | clut[142] = [ 0,127,255] | clut[161] = [ 85,255, 0] | clut[180] = [255, 42,255] |
| | clut[143] = [ 0,170, 0] | clut[162] = [ 85,255,255] | clut[181] = [255, 85, 0] |
| | clut[144] = [ 0,170,255] | clut[163] = [170, 0, 0] | clut[182] = [255, 85,255] |
| | clut[145] = [ 0,212, 0] | clut[164] = [170, 0,255] | clut[183] = [255,127, 0] |
| | clut[146] = [ 0,212,255] | clut[165] = [170, 42, 0] | clut[184] = [255,127,255] |
| | clut[147] = [ 0,255, 0] | clut[166] = [170, 42,255] | clut[185] = [255,170, 0] |
| | clut[148] = [ 0,255,255] | clut[167] = [170, 85, 0] | clut[186] = [255,170,255] |
| | clut[149] = [ 85, 0, 0] | clut[168] = [170, 85,255] | clut[187] = [255,212, 0] |
| | clut[150] = [ 85, 0,255] | clut[169] = [170,127, 0] | clut[188] = [255,212,255] |
| | clut[151] = [ 85, 42, 0] | clut[170] = [170,127,255] | clut[189] = [255,255, 0] |
| | clut[152] = [ 85, 42,255] | clut[171] = [170,170, 0] | clut[190] = [255,255,255] |
| | clut[153] = [ 85, 85, 0] | clut[172] = [170,170,255] | |
| 50 % | clut[191] = [ 0, 0, 0] | clut[203] = [127, 0, 0] | clut[215] = [255, 0, 0] |
| | clut[192] = [ 0, 0,255] | clut[204] = [127, 0,255] | clut[216] = [255, 0,255] |
| | clut[193] = [ 0, 51, 0] | clut[205] = [127, 51, 0] | clut[217] = [255, 51, 0] |
| | clut[194] = [ 0, 51,255] | clut[206] = [127, 51,255] | clut[218] = [255, 51,255] |
| | clut[195] = [ 0,102, 0] | clut[207] = [127,102, 0] | clut[219] = [255,102, 0] |
| | clut[196] = [ 0,102,255] | clut[208] = [127,102,255] | clut[220] = [255,102,255] |
| | clut[197] = [ 0,153, 0] | clut[209] = [127,153, 0] | clut[221] = [255,153, 0] |
| | clut[198] = [ 0,153,255] | clut[210] = [127,153,255] | clut[222] = [255,153,255] |
| | clut[199] = [ 0,204, 0] | clut[211] = [127,204, 0] | clut[223] = [255,204, 0] |
| | clut[200] = [ 0,204,255] | clut[212] = [127,204,255] | clut[224] = [255,204,255] |
| | clut[201] = [ 0,255, 0] | clut[213] = [127,255, 0] | clut[225] = [255,255, 0] |
| | clut[202] = [ 0,255,255] | clut[214] = [127,255,255] | clut[226] = [255,255,255] |
| 75 % | clut[227] = [ 0, 0, 0] | clut[233] = [ 0,255, 0] | clut[238] = [255, 85,255] |
| | clut[228] = [ 0, 0,255] | clut[234] = [ 0,255,255] | clut[239] = [255,170, 0] |



| | | | |
|---|---|---|---|
| | clut[229] = [ 0, 85, 0]<br>clut[230] = [ 0, 85,255]<br>clut[231] = [ 0,170, 0]<br>clut[232] = [ 0,170,255] | clut[235] = [255, 0, 0]<br>clut[236] = [255, 0,255]<br>clut[237] = [255, 85, 0] | clut[240] = [255,170,255]<br>clut[241] = [255,255, 0]<br>clut[242] = [255,255,255] |
| 100 % (fully transparent) | clut[243] = [ x, x, x], where x indicates “don’t care”. | | |
| privately definable | clut[244] : reserved for private use<br>clut[245] : reserved for private use<br>clut[246] : reserved for private use<br>clut[247] : reserved for private use<br>clut[248] : reserved for private use<br>clut[249] : reserved for private use | clut[250] : reserved for private use<br>clut[251] : reserved for private use<br>clut[252] : reserved for private use<br>clut[253] : reserved for private use<br>clut[254] : reserved for private use<br>clut[255] : reserved for private use | |

# Annex D
(normative)
# PES Packetization of DAVIC defined Monomedia Components

## D.1 Packetization into PES packets

Each monomedia object such as DAVIC defined linear audio is packetized in PES packets.

The following constraints apply for such packetization, in addition to the constraints defined in Clause 7-3 of this specification.

- In the case of linear audio, an encoded PTS shall refer to the presentation time of the first audio sample contained in the payload of the packet in which the PTS is coded.
- A PTS shall be encoded in the PES packet that contains the first byte of the elementary object. In each composite object the PTS of the elementary object that is presented first, shall be encoded with a value zero. A PTS is encoded in the PES packet that contains the first byte of the elementary object. In each composite object the PTS of the elementary object that is presented first, shall be encoded with a value zero.
- the trick_mode_flag shall be encoded with a value '0'.
- the stream_id field in the PES header shall be coded with a value equal to '1011 1101', indicating a private stream 1.
- A DTS shall be not be encoded.

To allow PES packets of private stream 1 to contain multiple monomedia components, the syntax of such PES packets is extended, corresponding to the structure defined below.

**Table D-1. — Extended PES Packet Header of Type Private Stream 1**

| *Syntax* | *Number of bits* | *Mnemonic* |
|---|---|---|
| Private_stream_1_PES_packet() { | | |
| private_stream_1_PES_packet_header() | | |
| data_identifier | 8 | bslbf |
| private_stream_id | 8 | bslbf |
| private_PES_packet_data() | | |
| } | | |

**Private_stream_1_PES_packet_header()**: A PES packet header that complies to ISO/IEC 13818-1 with the stream_id set to a value of '1011 1101', indicating "private_stream_1".

**data_identifier**: A byte indicating the type of data contained in this PES packet, corresponding to the following table.

**Table D-2. — Data Identifier Assignments**

| *Value* | *Meaning* |
|---|---|
| 0x00 - 0x0F | reserved for use by DAVIC |
| 0x10 - 0x1F | user-private |
| 0x20 | forbidden (*used for DVB subtitling*) |
| 0x21 - 0x2F | user-private |
| 0x30 | objects with a coding format defined in ISO/IEC |



| | |
|---|---|
| 16500 | |
| 0x31 - 0x7F | reserved for use by DAVIC |
| 0x80 - 0xFF | user-private |

**private_stream_id**: A byte indicating the type of private stream. DAVIC only specifies the coding of this field if the data_identifier field is coded with a value that is reserved for use by DAVIC. If the data_identifier field is encoded with the value 0x30, indicating an objects with a coding format defined in DAVIC, then the private_stream_id shall be coded corresponding to the following table.

**Table D-3. — Private Stream Id Assignments for Objects with a Coding Format Defined in ISO/IEC 16500**

| *Value* | *Meaning* |
|---|---|
| 0x00 - 0x0F | reserved |
| 0x10 - 0x1F | linear audio |
| 0x20 - 0xDF | reserved |
| 0xE0 - 0xFF | user private |

**private_PES_packet_data()** : A field containing data from a monomedia object. The objects are aligned with the extended PES packet header, i.e. the first byte of the object is the first byte of the PES packet data. The data from each object may be stored in multiple PES packets, all of which include the extended header syntax defined in this Clause. DAVIC will not specify data contained in the private_PES_packet_data field of user private PES packets.

# Annex E
(normative)
# MPEG-2 Section Filter API

## E.1 Objective

The objective of this API is to provide a general mechanism allowing access to data held in MPEG-2 private sections. This will provide a mechanism for inter-operable access to data which is too specialized to be supported by the high level DVB-SI API or which is not actually related to service information.

## E.2 Requirements

### E.2.1 Non-Functional Requirements

- Openness
- The interfaces should be specified in such a way that they can be used also in the implementation of other interfaces.
- Abstraction
- However, the interface should not expose its implementation, making it possible to implement it for instance in native code.
- Memory and CPU power
  - The random-access memory requirements of a typical implementation should not substantially add to the requirements of the underlying native services.
  - The CPU cycle budget of a typical implementation should not substantially add to the budget of a similar service in a native implementation. Specifically, where native applications takes advantage of hardware support to carry out a certain function, the interface specification must allow re-use of that feature.
- Response times should not be substantially higher than the response times of the underlying native services.
- Some resource handling functionality should be defined to control scarce resources. All interfaces using scarce resources should use this functionality.
- The interface should allow everything in the range from a complete software implementation to an almost complete hardware implementation. The interface should hide all aspects of the software/hardware trade-off from the implementation..
- The system must be flexible and easily extendible to be future proof.
- Maximal use should be made of the object-oriented paradigm of Java to allow sub-typing in new packages or in the application.
- The interfaces should have a Java look and feel.

### E.2.2 Functional Requirements

- The basic functional requirement on org.davic.mpeg.sections is to provide convenient application-level access to any data transported in MPEG sections independent of transmission scheme.
- The section filter should be associated with one transport stream and, if possible, also one transport stream source.
- The API should support the following section types ( [MPEG], §2.4.4.10 ) :-
  - Both “long header syntax” and “short header syntax”
  - Both 4k and 1k sections.



– It should be possible to indicate the buffer size for desired sections at filter creation time.

- There should be an ability to filter on bytes 0, 3-9 of a section.
- There should be an efficient mechanism to monitor change in sections.
- After a filter has been started, it must be possible to terminate the operation without any data having arrived.
- The package should enable the application to be notified in the event that filtering resources are not sufficient to honor a request including cases where a filter becomes not available.
- The package should allow parsing of header data as defined by MPEG-2, table 2.30 of ISO/IEC 13818-1.
- There should be an ability to notify the user of the API when a complete section has arrived.

NOTE It is a desirable feature that the package supports at least two different priority levels of filters (for background filtering) and also avoid double buffering/copying.

## E.3 Overview of Specification

This subclause presents the specification of the org.davic.mpeg.sections package. The specification is given using the OMT method.

The aim is to provide a platform-neutral interface which allows access to the MPEG2 sections present in a MPEG2 Transport Stream. For readers not familiar with the MPEG2 system layer and the use of the section format, see the MPEG2 system layer specification [MPEG2].

The package allows an application to create section filters, to connect section filters to a section source (Transport Stream) and to manage connecting and disconnecting of resources.

### E.3.1 Object diagram

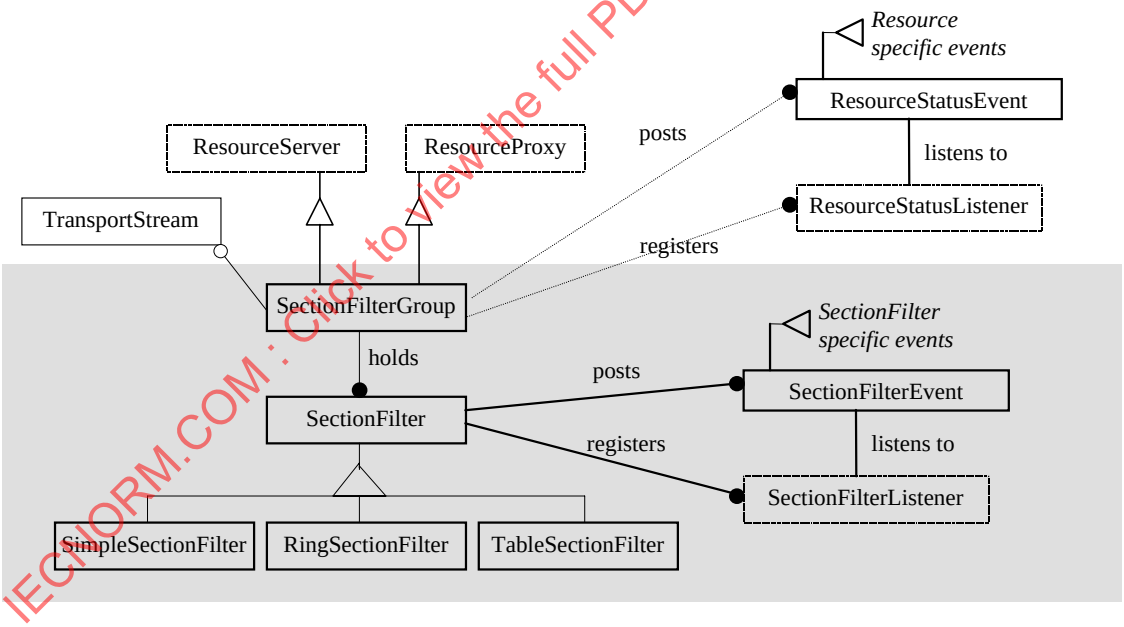



**Figure E-1. — Object diagram**

In Figure E-1 the total object diagram of the package is presented. In this diagram interfaces are presented with dotted boxes and classes with straight boxes. The classes and interfaces part of the section filtering API are presented within the gray area. The classes and interfaces outside this area are defined in the Resource Notification API and MPEG component API.

In this picture the relations between the following section filtering classes are defined:

- **SectionFilterGroup**, the class responsible for creating SectionFilter objects and maintaining the connection with a TransportStream. This class uses the ResourceProxy and ResourceServer interface the manage the section filter resources.



- **SectionFilter**, the abstract super class of all section filter classes. This class provides the methods to start and stop section filters.
- **Section**, this class contains the sections filtered from a Transport Stream by a SectionFilter instance.
- **SectionFilterListener**, an interface used to 'listen' to events happening in a SectionFilter instance.
- **SectionFilterEvent**, the main class of the SectionFilterGroup events.
- **SimpleSectionFilter**, a subclass of SectionFilter providing basic filter functionality to filter one section.
- **RingSectionFilter**, a subclass of SectionFilter to filter a continuous stream of MPEG-2 sections.
- **TableSectionFilter**, a subclass of SectionFilter filtering a table of MPEG-2 sections.

### E.3.2 Interclass relations

Before SectionFilter can be created first a SectionFilterGroup has to be found. A SectionFilterGroup is created with the maximum number of section filters to be used and a priority indication.

The created SectionFilterGroup is responsible for:

- the creation of SectionFilters
- the connection with a TransportStream
- the managing and control of the resources needed for section filtering.

Before filtering can commence a connection with a TransportStream has to be made.

The SectionFilterGroup has to attach itself to the TransportStream. This mechanism is controlled by the resource mechanism.

The activity of the SectionFilterGroup can be monitored by implementing the ResourceStatusEventListener interface. Classes of this type registered with a SectionFilterGroup receive ResourceStatusEvents each time something noteworthy has happened in that SectionFilterGroup. A complete list of ResourceStatusEvents is given later on.

From the SectionFilterGroup the required SectionFilters can be obtained. Three different SectionFilter classes exist. At any one time any number of SectionFilter objects may be associated with this SectionFilterGroup, but only the number of section filters of this SectionFilterGroup can be active.

A SectionFilterGroup is responsible for maintaining the connection with a TransportStream, a SectionFilter is responsible for the actual filtering. The objects of the SectionFilter class allow a user to start a filter with filter parameters and read any filtered section from the Transport Stream.

Similar to the SectionFilterGroup the SectionFilter objects allow clients to register SectionFilterListeners to which SectionFilterEvents are sent. The registered SectionFilterListeners receive events every time a section arrives and when the filter starts and stops.

### E.3.3 Event mechanism

The event mechanism allows the application to respond to the different processes and changes in the lifecycle of the different objects. They are used to signal errors and important state changes.

The SectionFilter and SectionFilterGroup both use an event mechanism. The class diagram of the events in this interface is given in Figure E-2.



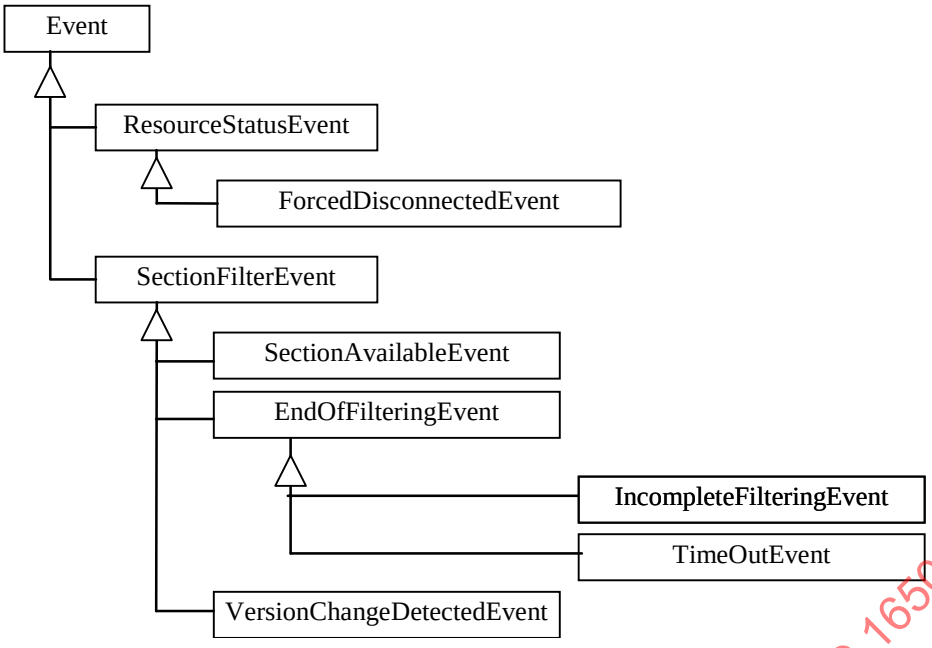



**Figure E-2. — Class diagram of events generated by the SectionFilterGroup and the SectionFilter**

Events generated by the SectionFilterGroup are :

- **ResourceStatusEvent**, a general event generated when the status of a resource changes.
- **ForcedDisconnectedEvent**, this event is generated when the SectionFilterGroup is detached by the underlying system.

Events generated by the Section Filter are :

- **SectionAvailableEvent**, indication of the SectionFilter to the user that a new section has been filtered and can be read.
- **EndOfFilteringEvent**, the filter process was stopped because it was completed or for some other reason. It is not used to signal a call to stopFiltering
- **IncompleteFilteringEvent**, the filter process for a TableSectionFilter was stopped because the filter parameters have been incompletely defined, resulting in a blocking filter or a non MPEG-2 compliant result
- **TimeOutEvent**, the SectionFilter has timed out.
- **VersionChangeDetectedEvent**, indication of the TableSectionFilter to the user that a section has been encountered that contained a different version_number from earlier sections that were filtered.

All events are sent asynchronously. A consequence of this is that events originating from a filtering actions can still come in when a new filtering action has already been started. It is the responsibility of the application to check if the event is still relevant.



### E.3.4 SectionFilterGroups

The SectionFilterGroup is responsible for:

- the creation of SectionFilters
- the connection with a TransportStream
- the managing and control of the resources needed for section filtering

The behaviour of the connection between TransportStream and SectionFilterGroup is defined by a finite state machine; see Figure E-3.

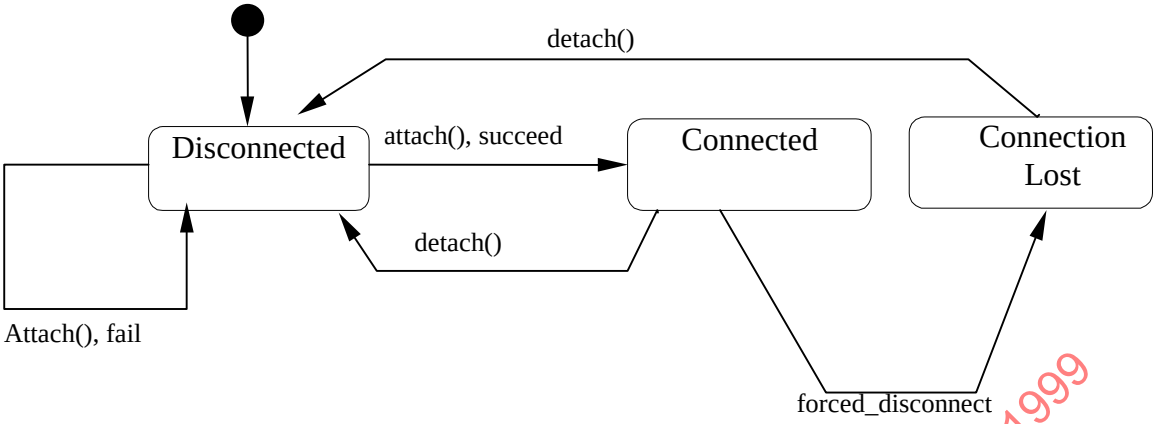



**Figure E-3 — Finite State Machine for connection between TransportStream and SectionFilterGroup**

In Figure E-3, method calls are indicated with a ().

In this finite state machine the states Disconnected and Connected occur:

- **Disconnected**, the state the SectionFilterGroup starts in. In this state no connection is made with a TransportStream and no resources are claimed by the SectionFilterGroup. In this state calls to startFiltering() methods are effectively queued until the SectionFilterGroup enters the Connected state.
- **Connected**, the SectionFilterGroup is connected to a TransportStream and has the number of section filter resources indicated when created. All active SectionFilters are filtering data. In this state calls to startFiltering() methods cause filtering to start upto the limit of available section filters.
- **ConnectionLost**, the SectionFilterGroup has stopped filtering due to the loss of a connection or of a resource. In this case, calls to startFiltering() methods always throw exceptions.

The following events occur:

- **attach(), fail**, an attempt was made to connect the SectionFilterGroup to a TransportStream and failed due to a incorrect TransportStream or not enough resources are available.
- **attach(),succeed**, the SectionFilterGroup was successfully connected to a TransportStream and claimed the number of section filter resources indicated when created. All active SectionFilters start filtering.
- **detach()**, the user has disconnected the SectionFilterGroup from the TransportStream. All section filter resources are released and all filtering is stopped.
- **force_disconnect**, the system has reclaimed the section filter resources of this SectionFilterGroup or the transport stream is no longer available. When this happens, all SectionFilter objects in this SectionFilterGroup are stopped as if the application had called the stopFiltering() method for each such object.

### E.3.5 SectionFilters

The objects of the SectionFilter class represent the actual section filter functionality. Three different SectionFilter classes exist:

- **SimpleSectionFilter**, to filter one section
- **TableSectionFilter**, to filter an MPEG2 section table
- **RingSectionFilter**, for continuous filtering.

All SectionFilter objects use the same methods to set a filter. A SectionFilter claims a section filter resource from its SectionFilterGroup when startFiltering is called.

A SectionFilterGroup knows the number of section filter resources it holds. This is the maximum number of active SectionFilters connected to this SectionFilterGroup. The number of active SectionFilters connected to a



SectionFilterGroup is independent of the state of the SectionFilterGroup. This allows a user to set a number of SectionFilters and start and stop them with the attach and detach method of SectionFilterGroup.

## E.4 Interfaces

### E.4.1 SectionFilterListener

public interface **SectionFilterListener**

The SectionFilterListener interface is implemented by classes which wish to be informed about events relating to section filters.

#### Methods

**sectionFilterUpdate**

```
public abstract void sectionFilterUpdate(SectionFilterEvent event)
```

When a section filter event happens, the sectionFilterUpdate method of all listeners connected to the originating object will be called.

## E.5 Classes

### E.5.1 SectionFilterGroup

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterGroup
```

public class **SectionFilterGroup**

extends Object

implements ResourceProxy, ResourceServer

This class represents a group of MPEG-2 section filters to be activated and de-activated as an atomic operation. The purpose of this class is to minimize the potential for resource deadlock between independent pieces of application(s).

#### Constructors

```
public SectionFilterGroup(int numberOfFilters)
```

Creates a section filter group object with an associated number of section filters needed to be reserved when the object is to be connected to an active source of MPEG-2 sections. The object will have a default high resource priority should the number of section filters available to the package become insufficient.

**Parameters:**

numberOfFilters - the number of section filters needed for the object.

```
public SectionFilterGroup(int numberOfFilters,
                          boolean resourcePriority)
```

Creates a section filter group object with an associated number of section filters needed to be reserved when the object is to be connected to an active source of MPEG-2 sections.

**Parameters:**



numberOfFilters - the number of section filters needed for the object

resourcePriority - the resource priority of the object should the number of section filters available to the package become insufficient. High priority is indicated by true and low priority by false.

## Methods

**newSimpleSectionFilter**

```
public SimpleSectionFilter newSimpleSectionFilter()
```

Creates a new simple section filter object within the parent section filter group. On activation (succesfull startFiltering) the SimpleSectionFilter object will use section filters from the total specified when the parent SectionFilterGroup was created. The section filter object will have a buffer suitable to hold a default long section.

**newSimpleSectionFilter**

```
public SimpleSectionFilter newSimpleSectionFilter(int sectionSize)
```

Creates a new simple section filter object within the parent section filter group. On activation (succesfull startFiltering) the SimpleSectionFilter object will use section filters from the total specified when the parent SectionFilterGroup was created.

**Parameters:**

sectionSize - specifies the size in bytes of the buffer to be created to hold data captured by the SectionFilter. If sections are filtered which are larger than this then the extra data will be dropped and filtering continue without any notification to the application.

**newRingSectionFilter**

```
public RingSectionFilter newRingSectionFilter(int ringSize)
```

Creates a new ring section filter within the parent section filter group. On activation (succesfull startFiltering) the new RingSectionFilter object will use section filters from the total specified when the parent SectionFilterGroup was created.

**Parameters:**

ringSize - the number of Section objects to be created for use in the ring.

**newRingSectionFilter**

```
public RingSectionFilter newRingSectionFilter(int ringSize,
                                              int sectionSize)
```

Creates a new ring section filter within the parent section filter group. On activation (succesfull startFiltering) the new RingSectionFilter object will use section filters from the total specified when the parent SectionFilterGroup was created.

**Parameters:**

ringSize - the number of Section objects to be created for use in the ring.

sectionSize - the size in bytes of the buffer for each Section object. If sections are filtered which are larger than this then the extra data will be dropped and filtering continue without any notification to the application.

**newTableSectionFilter**

```
public TableSectionFilter newTableSectionFilter()
```

Creates a new table section filter object within the parent section filter group. On activation (succesfull startFiltering) the new TableSectionFilter object will use section filters from the total specified when the parent



SectionFilterGroup was created. Each Section created for the table section filter object will have a buffer suitable to hold a default long section.

**newTableSectionFilter**

```
public TableSectionFilter newTableSectionFilter(int sectionSize)
```

Creates a new table section filter object within the parent section filter group. On activation (succesfull startFiltering) the new TableSectionFilter object will use section filters from the total specified when the parent SectionFilterGroup was created.

**Parameters:**

sectionSize - specifies the size in bytes of the buffer to be created to hold data captured by the SectionFilter. When the first section has been captured and the total number of sections in the table known, each Section created will have a buffer of this size. If sections are filtered which are larger than this then the extra data will be dropped and filtering continue without any notification to the application.

**attach**

```
public void attach(TransportStream stream, ResourceClient client,
       Object requestData) throws FilterResourceException,
       InvalidSourceException, TuningException
```

Connects a SectionFilterGroup to an MPEG-2 transport stream. The SectionFilterGroup will attempt to acquire the number of section filters specified when it was created. Any SectionFilter objects which are part of the group concerned and whose filtering has been started will become active and start filtering the source for sections matching the specified patterns. A call to attach on a attached SectionFilterGroup will be treated as a detach followed by the new attach.

Parameters:

stream- specifies the source of MPEG-2 sections for filtering

client - specifies an object to be notified if the section filters acquired during this method are later removed by the environment for any reason.

requestData - application specific data for use by the resource notification API

Throws: FilterResourceException

if reserving the specified section filters fails.

**Throws:** InvalidSourceException

if the source is not a valid source of MPEG-2 sections.

**Throws:** TuningException

if the source is not currently tuned to

**detach**

```
public void detach()
```

Returns a SectionFilterGroup to the disconnected state. When called for a SectionFilterGroup in the connected state, it disconnects a SectionFilterGroup from a source of MPEG-2 sections. The section filters held by the SectionFilterGroup will be released back to the environment. Any running filtering operations will be terminated. This method will have no effect for SectionFilterGroups already in the disconnected state.

**getSource**

```
public TransportStream getSource()
```

Returns the MPEG-2 transport stream to which a SectionFilterGroup is currently connected. If the SectionFilterGroup is not connected to a transport stream then the method will return null.

**getClient**

```
public ResourceClient getClient()
```

Returns the ResourceClient object specified in the last call to the attach() method as to be notified in the case that the section filters acquired by the SectionFilterGroup during that call to attach() are removed by the environment for any reason. If the SectionFilterGroup is not connected to a source then the method will return null.

**addResourceStatusEventListener**

```
public void addResourceStatusEventListener(ResourceStatusListener listener)
```

Specifies an object to be notified of changes in the status of resources related to a SectionFilterGroup object. If this call is made more than once, each specified listener will be notified of each change in resource status.

**Parameters:**

listener - the object to be notified

**removeResourceStatusEventListener**

```
public void removeResourceStatusEventListener(ResourceStatusListener
listener)
```

Indicates that an object is no longer to be notified of changes in the status of resources as setup by addResourceStatusEventListener. If an object was not specified as to be notified then this method will be have no effect.

**Parameters:**

listener - the object no longer to be notified

### E.5.2 SectionFilter

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilter
```

public abstract class **SectionFilter**

extends Object

This class is the base class for a set of classes describing section filters with different characteristics of life cycle and buffering.

## Methods

**startFiltering**

```
public void startFiltering(Object appData,
                           int pid) throws FilterResourceException,
NotAuthorizedException, IllegalFilterDefinitionException,
ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**



appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** IllegalFilterDefinitionException

if called for a TableSectionFilter.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**startFiltering**

```
 public void startFiltering(Object appData,
                            int pid,
                            int table_id) throws FilterResourceException,
NotAuthorizedException, ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID and table_id. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**

appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections

table_id - the value of the table_id to filter for in incoming sections

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**startFiltering**

```
 public void startFiltering(Object appData,
                            int pid,
```

```
                              int table_id,
                              byte posFilterDef[],
                              byte posFilterMask[]) throws
FilterResourceException, IllegalFilterDefinitionException,
NotAuthorizedException, ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID and table_id, and where contents of the section match the specified filter pattern. The first byte of each array corresponds to the third byte of the section. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**

appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections.

table_id - the value of the table_id field to filter for in incoming sections

posFilterDef - defines values to match for bits in the section, as defined in clause H7.

posFilterMask - defines which bits in the section are to be compared against the values specified in the posFilterDef parameter, as defined in clause H7.

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** IllegalFilterDefinitionException

the filter definition specified is illegal either because the posFilterDef and posFilterMask arrays are of different sizes or because their length is beyond the filtering capacity of the system.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**startFiltering**

```
 public void startFiltering(Object appData,
                              int pid,
                              int table_id,
                              int offset,
                              byte posFilterDef[],
                              byte posFilterMask[]) throws
FilterResourceException, IllegalFilterDefinitionException,
NotAuthorizedException, ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID and table_id, and where contents of the section match the specified filter pattern. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**

appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can



use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections.

table_id - the value of the table_id field to filter for in incoming sections

offset - defines the offset within the section which the first byte of the posFilterDef and posFilterMask arrays is intended to match. The offset must be less than 31 as described in ISO/IEC 16500-7 subclause 12.5.3. The offset must be equal to or greater than 3.

posFilterDef - defines values to match for bits in the section, as defined in clause H7.

posFilterMask - defines which bits in the section are to be compared against the values specified in the posFilterDef parameter, as defined in clause H7.

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** IllegalFilterDefinitionException

the filter definition specified is illegal either because the posFilterDef and posFilterMask arrays are not the same size or because their length is beyond the filtering capacity of the system or because the specified offset is too large.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**startFiltering**

```
 public void startFiltering(Object appData,
                            int pid,
                            int table_id,
                            byte posFilterDef[],
                            byte posFilterMask[],
                            byte negFilterDef[],
                            byte negFilterMask[]) throws
FilterResourceException, IllegalFilterDefinitionException,
NotAuthorizedException, ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID and table_id, and where contents of the section match the specified filter pattern. The first byte of each array corresponds to the third byte of the section. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**

appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections.

table_id - the value of the table_id field to filter for in incoming sections



posFilterDef - defines values to match for bits in the section, as defined in clause H7.

posFilterMask - defines which bits in the section are to be compared against the values specified in the posFilterDef parameter, as defined in clause H7.

negFilterDef - defines values to match for bits in the section, as defined in clause H7.

negFilterMask - defines which bits in the section are to be compared against the values specified in the negFilterDef parameter, as defined in clause H7.

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** IllegalFilterDefinitionException

the filter definition specified is illegal either because the arrays posFilterDef, posFilterMask, negFilterDef, negFilterMask are not all the same size or because their length is beyond the filtering capacity of the system.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**startFiltering**

```
 public void startFiltering(Object appData,
                            int pid,
                            int table_id,
                            int offset,
                            byte posFilterDef[],
                            byte posFilterMask[],
                            byte negFilterDef[],
                            byte negFilterMask[]) throws
FilterResourceException, IllegalFilterDefinitionException,
NotAuthorizedException, ConnectionLostException
```

Defines a SectionFilter object as filtering only for sections matching a specific PID and table_id, and where contents of the section match the specified filter pattern. If the parent SectionFilterGroup is attached to a TransportStream then filtering will start immediately.

**Parameters:**

appData - An object supplied by the application. This object will be delivered to the subscribed section filter listener as part of all SectionFilterEvents that will be generated because of this method call. The application can use this object for internal communication purposes. If the application does not need any application data, the parameter can be null.

pid - the value of the PID to filter for in incoming sections.

table_id - the value of the table_id field to filter for in incoming sections

offset - defines the offset within the section which the first byte of the posFilterDef, posFilterMask, negFilterDef and negFilterMask arrays is intended to match. The offset must be less than 31 as described in ISO/IEC 16500-7 subclause 12.5.3. The offset must be equal to or greater than 3.

posFilterDef - defines values to match for bits in the section, as defined in clause H7.

posFilterMask - defines which bits in the section are to be compared against the values specified in the



posFilterDef parameter, as defined in clause H7.

negFilterDef - defines values to match for bits in the section, as defined in clause H7.

negFilterMask - defines which bits in the section are to be compared against the values specified in the negFilterDef parameter, as defined in clause H7.

**Throws:** FilterResourceException

if all the number of started SectionFilters for the parent SectionFilterGroup is already equal to the number of section filters associated with the SectionFilterGroup when it was created.

**Throws:** IllegalFilterDefinitionException

the filter definition specified is illegal either because the posFilterDef, posFilterMask, negFilterDef, negFilterMask arrays are not all the same size or because their length is beyond the filtering capacity of the system or because the specified offset is too large.

**Throws:** NotAuthorizedException

if the information requested is scrambled and permission to descramble it is refused.

**Throws:** ConnectionLostException

if the parent SectionFilterGroup is in the ConnectionLost state and hence is unable to satisfy the method call due to absence of resources or absence of sections to filter.

**stopFiltering**

```
public void stopFilter ing()
```

If the parent SectionFilterGroup is attached to a TransportStream then filtering for sections matching this SectionFilter object will stop. If the parent is not attached then should it become attached, filtering for sections matching this SectionFilter object will not start.

**setTimeOut**

```
public void setTimeOut(long milliseconds) throws IllegalArgumentException
```

Sets the time-out for this section filter. When the time-out happens, a TimeOutEvent will be generated and sent to the SectionFilter object and filtering stops. For a SimpleSectionFilter this will be generated if no sections arrive within the specified period. For a TableSectionFilter, this will be generated if the complete table does not arrive within the specified time. For a RingSectionFilter, this will be generated if the specified time has elapsed since the arrival of the last section being successfully filtered. Setting a time-out of 0 milliseconds has the effect of removing a possible time-out. A set time-out only applies to subsequent filter activations, not to a possible filter activation that is currently in progress when the call to this method is made. The default time-out value is 0.

**Parameters:**

milliseconds - the time out period

**Throws:** IllegalArgumentException

if the 'milliseconds' parameter is negative

**addSectionFilterListener**

```
public void addSectionFilterListener(SectionFilterListener listener)
```

Specifies an object to be notified of events relating to this SectionFilter object.

**Parameters:**

listener - the object to be notified of events

**removeSectionFilterListener**

```
public void removeSectionFilterListener(SectionFilterListener listener)
```

Indicates that an object is no longer to be notified of events relating to this SectionFilter object. If the object was not specified as to be notified then this method has no effect.

**Parameters:**

listener - the object no longer to be notified of events

### E.5.3 SimpleSectionFilter

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilter
           |
           +----org.davic.mpeg.sections.SimpleSectionFilter
```

public class **SimpleSectionFilter**

extends SectionFilter

This class defines a simple section filter intended to be used to capture a single section once only. When a section matching the specified filter pattern is found, SimpleSectionFilter objects will stop filtering as if the stopFiltering method had been called.

## Methods

**getSection**

```
public Section getSection() throws FilteringInterruptedException
```

This method retrieves a Section object describing the last MPEG-2 section which matched the specified filter characteristics. If the SimpleSectionFilter object is currently filtering, this method will block until filtering stops. Repeated calls to this method will return the same Section object, provided that no new calls to startFiltering have been made in the interim. Each time a new filtering operation is started, a new Section object will be created. All references except any in the application to the previous Section object will be removed. All data accessing methods on the previous Section object will throw a NoDataAvailableException.

**Throws:** FilteringInterruptedException

if filtering stops before a matching section is found

### E.5.4 RingSectionFilter

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilter
           |
           +----org.davic.mpeg.sections.RingSectionFilter
```

public class **RingSectionFilter**

extends SectionFilter

This class defines a section filter intended to be used to capture a continuous stream of MPEG-2 sections without needing to stop and re-arm a filter. A RingSectionFilter object has a pre-defined number of Section objects as part of it. Incoming sections are loaded sequentially into these Section objects. Filtering proceeds while empty Section objects remain. Applications wishing filtering to proceed indefinitely must use the setEmpty method of the Section object to mark Section objects as empty before the filling process reaches them. If the filtering process reaches a non-empty Section object, it will terminate at that point. On each occasion when startFiltering



is called, the sections will be captured starting from the beginning of the array.

## Methods

**getSections**

```
public Section[] getSections()
```

This method returns the Section objects of the RingSectionFilter in an array. The array will be fully populated at all times, it is the responsibility of the application to check which of these contain valid data. Repeated calls to this method will always return the same result.

### E.5.5 TableSectionFilter

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilter
           |
           +----org.davic.mpeg.sections.TableSectionFilter
```

public class **TableSectionFilter**

extends SectionFilter

This class defines a section filter operation optimized to capture entire tables with minimum intervention required from the application. When filtering is started, first one section matching the specified pattern will be filtered. Once that section has been found, the last_section_number field will be used to determine the number of Section objects required to hold the entire table. This number of objects will be created and filtering re-started to capture all the sections of the table. The SectionAvailableEvent will be generated each time a Section is captured. The EndOfFilteringEvent will be generated when the complete table has been captured. The version_number of all sections of the table will be the same. If a section is captured with a version_number that differs from the version_number of the section first captured, a VersionChangeDetectedEvent will be generated. The newly captured section will be ignored and filtering will continue on the table with the version number of the first captured section. Only one VersionChangeDetectedEvent will be sent per filtering action. Care should be taking in setting the filter parameters, a too restrictive filter will never stop automatically and a too wide filter can produce inconsistent results (e.g. filtering short sections using a TableSectionFilter) When the API detects a filtering situation where the filter parameters have been incompletely defined, resulting in a blocking filter or a non MPEG-2 compliant result, an InCompleteFilteringEvent is sent and filtering is stopped.

## Methods

**getSections**

```
public Section[] getSections() throws FilteringInterruptedException
```

This method returns an array of Section objects corresponding to the sections of the table. The sections in the array will be ordered according to their section_number. Any sections which have not yet been filtered from the source will have the corresponding entry in the array set to null. If no sections have been filtered then this method will block until at least one section is available or filtering stops. Repeated calls to this method will return the same array, provided that no new calls to startFiltering have been made in the interim. Each time a new filtering operation is started, a new array of Section objects will be created. All references except any in the application to the previous array and Section objects will be removed. All data accessing methods on the previous Section objects will throw a NoDataAvailableException.

**Throws:** FilteringInterruptedException

if filtering stops before one section is available

### E.5.6 Section

```
java.lang.Object
```

```
|
+----org.davic.mpeg.sections.Section
```

public class **Section**

extends Object

implements Cloneable

This class describes a single section as filtered from an MPEG transport stream. A cloned Section object is a new and separate object. It is unaffected by changes in the state of the original Section object or restarting of the SectionFilter the source Section object originated from. The clone method must be implemented without declaring exceptions.

## Methods

**getData**

```
public byte[] getData() throws NoDataAvailableException
```

This method returns all data from the filtered section in the Section object, including the section header. Each call to this method results in a new a copy of the section data (everything after the length field, not including a CRC check).

**Throws:** NoDataAvailableException

if no valid data is available.

**getData**

```
public byte[] getData(int index,
                      int length) throws NoDataAvailableException,
IndexOutOfBoundsException
```

This method returns the specified part of the filtered data. Each call to this method results in a new a copy of the section data (everything after the length field, not including a CRC check).

**Parameters:**

index - defines within the filtered section the index of the first byte of the data to be retrieved. The first byte of the section (the table_id field) has index 1.

length - defines the number of consecutive bytes from the filtered section to be retrieved.

**Throws:** NoDataAvailableException

if no valid data is available.

**Throws:** IndexOutOfBoundsException

if any part of the filtered data requested would be outside the range of data in the section.

**getByteAt**

```
public byte getByteAt(int index) throws NoDataAvailableException,
IndexOutOfBoundsException
```

This method returns one byte from the filtered data.

**Parameters:**

index - defines within the filtered section the index of the byte to be retrieved. The first byte of the section (the table_id field) has index 1.

**Throws:** NoDataAvailableException



if no valid data is available.

**Throws:** IndexOutOfBoundsException

if the byte requested would be outside the range of data in the section.

**table_id**

```
public int table_id() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**section_syntax_indicator**

```
public boolean section_syntax_indicator() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**private_indicator**

```
public boolean private_indicator() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**section_length**

```
public int section_length() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**table_id_extension**

```
public int table_id_extension() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**version_number**

```
public short version_number() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**current_next_indicator**

```
public boolean current_next_indicator() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**section_number**

```
public int section_number() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**last_section_number**

```
public int last_section_number() throws NoDataAvailableException
```

This method returns the value of the corresponding field from an MPEG-2 section header.

**Throws:** NoDataAvailableException

thrown if no valid data is available

**getFullStatus**

```
public boolean getFullStatus()
```

This method reads whether a Section object contains valid data.

**setEmpty**

```
public void setEmpty()
```

This method sets a Section object such that any data contained within it is no longer valid. This is intended to be used with RingSectionFilters to indicate that the particular object can be re-used.

## E.6 Events

### E.6.1 SectionFilterEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
```

public class **SectionFilterEvent**

extends Object

This class is the base class for Events in the section filter API.

**Constructors**

```
public SectionFilterEvent (SectionFilter f, Object appData)
```

This constructs a SectionFilterEvent for the specified SectionFilterObject.



**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilter object which was the source of the event.

**getAppData**

```
public Object getAppData()
```

Returns the application data that was passed to the startFiltering method

**Returns:**

the application data

### E.6.2 SectionAvailableEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
           |
           +----org.davic.mpeg.sections.SectionAvailableEvent
```

public class **SectionAvailableEvent**

extends SectionFilterEvent

This class is used to report a complete section being filtered. It is generated by SimpleSectionFilter, TableSectionFilter and RingSectionFilter objects when a section matching the filtering pattern is successfully filtered from the transport stream.

## Constructors

```
public SectionAvailableEvent (SectionFilter f, Object appData)
```

This constructs a SectionAvailableEvent for the specified SectionFilterObject.

**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilter object which filtered the data.

**Overrides:**

getSource in class SectionFilterEvent

### E.6.3 EndOfFilteringEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
           |
           +----org.davic.mpeg.sections.EndOfFilteringEvent
```

public class **EndOfFilteringEvent**

extends SectionFilterEvent

This class is used to report the end of a filtering operation with one exception: It is not generated when filtering stops for a SimpleSectionFilter under normal circumstances (i.e. after one section has successfully been filtered).

## Constructors

```
public EndOfFilteringEvent (SectionFilter f, Object appData)
```

This constructs an EndOfFilteringEvent for the specified SectionFilterObject.

**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilter object which filtered the data.

**Overrides:**

getSource in class SectionFilterEvent

### E.6.4 ForcedDisconnectedEvent

```
java.lang.Object
   |
   +----org.davic.resources.ResourceStatusEvent
           |
           +----org.davic.mpeg.sections.ForcedDisconnectedEvent
```

public class **ForcedDisconnectedEvent**

extends ResourceStatusEvent

This class is used to report when a TransportStream which was available becomes no longer available or if the section filter resources are removed from a connected SectionFilterGroup. In this second case, the notifyRelease() method of the ResourceClient will also be called in addition to this event being generated.

## Constructors

```
public ForcedDisconnectedEvent (SectionFilterGroup f)
```

This constructs a ForcedDisconnectedEvent for the specified SectionFilterGroup object.

**Parameters:**



f - the SectionFilter object where the event originated

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilterGroup object which filtered the data.

**Overrides:**

getSource in class ResourceStatusEvent

### E.6.5 VersionChangeDetectedEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
           |
           +----org.davic.mpeg.sections.VersionChangeDetectedEvent
```

public class **VersionChangeDetectedEvent**

extends SectionFilterEvent

This class is used by TableSectionFilter to report that a section has been encountered which has a different version_number from earlier sections. It is generated only once per filtering action. The section with a different version_number is ignored.

## Constructors

```
public VersionChangeDetectedEvent (SectionFilter f, Object appData)
```

This constructs a VersionChangeDetectedEvent for the specified SectionFilterObject.

**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilter object which filtered the data.

**Overrides:**

getSource in class SectionFilterEvent

**getOriginalVersion**

```
public int getOriginalVersion()
```

This returns the original version number of the table.

**getNewVersion**

```
public int getNewVersion()
```

This returns the version number of the new table.

### E.6.6 IncompleteFilteringEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
           |
           +----org.davic.mpeg.sections.EndOfFilteringEvent
                   |
                   +----org.davic.mpeg.sections.IncompleteFilteringEvent
```

public class **IncompleteFilteringEvent**

extends EndofFilteringEvent

This class is used to report the end of a filtering operation started by TableSectionFilter. This event is generated when the API detects a filtering situation where the filter parameters have been incompletely defined, resulting in a blocking filter or a non MPEG-2 compliant result.

## Constructors

```
public IncompleteFilteringEvent (SectionFilter f, Object appData)
```

This constructs an IncompleteFilteringEvent for the specified SectionFilterObject.

**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## Methods

**getSource**

```
public Object getSource()
```

This returns the SectionFilter object which filtered the data.

**Overrides:**

getSource in class EndOfFilteringEvent

### E.6.7 TimeOutEvent

```
java.lang.Object
   |
   +----org.davic.mpeg.sections.SectionFilterEvent
           |
           +----org.davic.mpeg.sections.EndOfFilteringEvent
                   |
                   +----org.davic.mpeg.sections.TimeOutEvent
```

public class **TimeOutEvent**

extends EndOfFilteringEvent

This event is generated if section filter operations time out within the period specified by the setTimeOut() method. For a SimpleSectionFilter it will be generated if no sections arrive within the specified period. For a TableSectionFilter, it will be generated if the complete table does not arrive within the specified time. For a



RingSectionFilter, it will be generated if the specified time has elapsed since the arrival of the last section being successfully filtered.

### Constructors

```
public TimeOutEvent (SectionFilter f, Object appData)
```

This constructs a TimeOutEvent for the specified SectionFilterObject.

**Parameters:**

f - the SectionFilter object where the event originated

appData - application data that was passed to the startFiltering method

## E.7 Exceptions

### E.7.1 SectionFilterException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.davic.mpeg.sections.SectionFilterException
```

public class **SectionFilterException**

extends Exception

This is the base class for exceptions in the section filter API.

### Constructors

```
public SectionFilterException()
```

Constructs a SectionFilterException with no detail message

```
public SectionFilterException(String s)
```

Constructs a SectionFilterException with the specified detail message

**Parameters:**

s - the detail message

### E.7.2 FilterResourceException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.davic.mpeg.sections.SectionFilterException
                           |
                           +----
org.davic.mpeg.sections.FilterResourceException
```

public class **FilterResourceException**

extends SectionFilterException

Signals that inadequate resources are available to support the requested operation when a SectionFilterGroup is in the connected or disconnected states.

## Constructors

```
public FilterResourceException()
```

Constructs a resource Exception.

```
public FilterResourceException(String s)
```

Constructs a FilterResourceException with the specified detail message

**Parameters:**

s - the detail message

### E.7.3 ConnectionLostException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.davic.mpeg.sections.SectionFilterException
                           |
                           +----
org.davic.mpeg.sections.ConnectionLostException
```

public class **ConnectionLostException**

extends SectionFilterException

Signals that a SectionFilterGroup has lost its connection or resources and hence is unable to satisfy a call to a startFiltering method. It is only generated for SectionFilterGroups which are in the ConnectionLost state.

## Constructors

```
public ConnectionLostException()
```

Constructs an exception.

```
public ConnectionLostException(String s)
```

Constructs a ConnectionLostException with the specified detail message

**Parameters:**

s - the detail message

### E.7.4 InvalidSourceException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
```



```
                    |
                    +----org.davic.mpeg.sections.SectionFilterException
                            |
                            +----org.davic.mpeg.sections.InvalidSourceException
```

public class **InvalidSourceException**

extends SectionFilterException

Signals that the section source specified is not valid for some reason.

## Constructors

`public InvalidSourceException()`

Constructs an InvalidSourceException with no detail message.

`public InvalidSourceException(String detail)`

Constructs an InvalidSourceException with the specified detail message.

**Parameters:**

detail - the detail message

### E.7.5 NoDataAvailableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.davic.mpeg.sections.SectionFilterException
                           |
                           +----
org.davic.mpeg.sections.NoDataAvailableException
```

public class **NoDataAvailableException**

extends SectionFilterException

Signals that no data is available from a Section object.

## Constructors

`public NoDataAvailableException()`

Constructs a NoDataAvailableException.

`public NoDataAvailableException(String s)`

Constructs a NoDataAvailableException with the specified detail message

**Parameters:**

s - the detail message

### E.7.6 IllegalFilterDefinitionException

```
java.lang.Object
   |
   +----java.lang.Throwable
```

```
            |
            +----java.lang.Exception
                    |
                    +----org.davic.mpeg.sections.SectionFilterException
                            |
                            +----
org.davic.mpeg.sections.IllegalFilterDefinitionException
```

public class **IllegalFilterDefinitionException**

extends SectionFilterException

Signals that a requested filter definition is not valid.

## Constructors

`public IllegalFilterDefinitionException()`

Constructs an IllegalFilterDefinitionException.

`public IllegalFilterDefinitionException(String s)`

Constructs a IllegalFilterDefinitionException with the specified detail message

**Parameters:**

s - the detail message

### E.7.7 FilteringInterruptedException

```
java.lang.Object
   |
   +----java.lang.Throwable
            |
            +----java.lang.Exception
                    |
                    +----org.davic.mpeg.sections.SectionFilterException
                            |
                            +----
org.davic.mpeg.sections.FilteringInterruptedException
```

public class **FilteringInterruptedException**

extends SectionFilterException

Signals that a filtering operation was interrupted before any data had been filtered.

## Constructors

`public FilteringInterruptedException()`

Constructs an FilteringInterruptedException.

`public FilteringInterruptedException(String s)`

Constructs a FilteringInterruptedException with the specified detail message

**Parameters:**

s - the detail message



## E.8 Section filtering

Most high level modules use the org.davic.mpeg.sections module. This module filters sections from an MPEG-2 Transport Stream. A section is a data package format commonly used to send data in an MPEG-2 TS.

### E.8.1 Filter options

The interfaces as presented in this document provide three kinds of interfaces:

- pure PID filtering
- PID filtering with a Positive filter
- PID filtering with a Negative and a Positive filter

When the sections are filtered in the pure PID filtering method all sections sent within a certain PID trigger this filter:

TS_PID = SF_PID

In this equation TS_PID is the value of the PID field of the TS-packets, the SF_PID is the PID value as defined in the section filter parameters.

The second filter provides the option to filter on the PID and on the header of the section. The filtering on the section header is done using a mask and a value parameter. The mask defines which bits are filtered on. The value holds the value these bits should have. If all the bits in the header set in the mask equal the value as set in the value parameter the filter is triggered. This kind of filter is called a Positive Filter

So this Positive Filter is triggered when the following situation occurs:

Value & Mask = SectionHeader & Mask

The third and final option adds another step to the filter process. In many situations it is needed to trigger a filter when a part of the header changes. For example when of a certain section such as the version_number field changes. To be able to do this another filter step is necessary. This filter is called a Negative Filter.

The Negative Filter also uses the mask and value parameters, but the Negative Filter triggers when:

value & mask ≠ header & mask

### E.8.2 MPEG2 section format (informative)

The format of long headers of MPEG-2 sections is presented in Figure E-4.

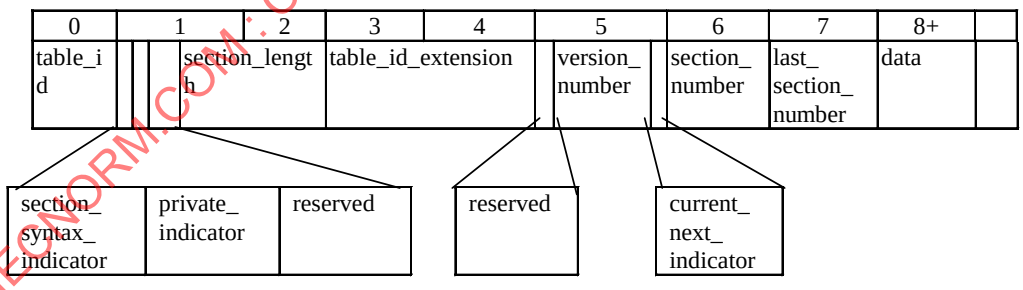



**Figure E-4 — Format of long header of MPEG-2 section**

The first row represents the byte order, the second row the most important fields. The length of the header equals 8 bytes. Of these 8 bytes, byte 0 holds the table_id and byte 1 and 2 hold no critical filter information. So a section filter with length 7, excluding the first three bytes, covers the complete section header and gives some freedom to the user for additional filter fields (for example CA uses datafield 1 as address).

This means for mask and value of length 7 the mapping as given in Figure E-5 is used:



| Mask or Value array | Section header byte |
|---|---|
| 0 | 3 |
| 1 | 4 |
| 2 | 5 |
| 3 | 6 |
| 4 | 7 |
| 5 | 8 |
| 6 | 9 |
| 7 | 10 |

**Figure E-5. — Mapping of Mask and Value on Section Header**



**(Blank page)**

# Annex F
(normative)
# Resource Notification API

## F.1 Objective

Any system with limited resources needs some way of managing those resources and ensuring that any contention is resolved fairly and without an adverse impact on application reliability. This applies even more in a set-top box where resources can be very limited and where application robustness is of vital importance. If such a system can run multiple applications (either interoperable or native, or a combination of the two), then some form of resource management is essential.

The API (org.davic.resources) described here provides a Java interface to either an existing resource manager or any resource management functionality implemented in the DAVIC environment. For this reason, the scope of this API is limited to those applications which have a Java component.

Many necessary methods are left undefined because the security requirements for those methods cannot be met by methods defined in a Java interface, or because the exact format of these methods and their parameters is best specified in any API using the resource notification API.

## F.2 Interfaces

### F.2.1 ResourceProxy

public interface **ResourceProxy**

The resource proxy interface is implemented by objects which represent a scarce resource to an application but where the actual scarce resource may be a lower level object to which the application does not have direct access. The indirection provided by ResourceProxy allows the retaining of state regardless of availability of the actual resource. Objects implementing the ResourceProxy interface are created by the application program and may have a lifetime longer than the time a resource is actually held by the application. A resource may be acquired and released multiple times using the same ResourceProxy object. All interaction between applications and objects abstracting over the resources themselves is carried out via an object implementing the ResourceProxy interface.

**Methods**

**getClient**

```
public abstract ResourceClient getClient()
```

**Returns:**

the object which asked to be notified about withdrawal of the underlying physical resource from a resource proxy.

### F.2.2 ResourceServer

public interface **ResourceServer**

The resource server interface is implemented by objects which manage low level scarce resources and inform applications of changes in their status which may have happened due to factors beyond the control of the application. Any application wishing to use a resource controlled by an object implementing the ResourceServer interface must request access to that resource via an object implementing the ResourceProxy interface, and should release the resource via the same object when exclusive access to the resource is no longer needed.



## Methods

**addResourceStatusEventListener**

```
public abstract void addResourceStatusEventListener(ResourceStatusListener
listener)
```

This method informs a resource server that a particular object should be informed of changes in the state of the resources managed by that server.

**Parameters:**

listener - the object to be informed of state changes

**removeResourceStatusEventListener**

```
public abstract void removeResourceStatusEventListener(ResourceStatusListener
listener)
```

This method informs a resource server that a particular object is no longer interested in being informed about changes in state of resources managed by that server. If the object had not registered its interest initially then this method has no effect.

**Parameters:**

listener - the object which is no longer interested

### F.2.3 ResourceStatusListener

public interface **ResourceStatusListener**

This interface should be implemented by objects wishing to be informed about changes in status of particular resources.

## Methods

**statusChanged**

```
public abstract void statusChanged(ResourceStatusEvent event)
```

This method is called by a ResourceServer when a resource changes status.

**Parameters:**

event - the change in status which happened.

### F.2.4 ResourceClient

public interface **ResourceClient**

This interface should be implemented by objects that use a scarce resource.

## Methods

**requestRelease**

```
public abstract boolean requestRelease(ResourceProxy proxy,
                                       Object requestData)
```

A call to this operation informs the ResourceClient that another application has requested the resource accessed via the proxy parameter. If the ResourceClient decides to give up the resource as a result of this, it should terminate its usage of proxy and return True, otherwise False. requestData may be used to pass more data to the

ResourceClient so that it can decide whether or not to give up the resource, using semantics specified outside this framework; for conformance to this framework, requestData can be ignored by the ResourceClient.

**Parameters:**

proxy - the ResourceProxy representing the scarce resource to the application

requestData - application specific data

**Returns:**

If the ResourceClient decides to give up the resource following this call, it should terminate its usage of proxy and return True, otherwise False.

**release**

```
public abstract void release(ResourceProxy proxy)
```

A call to this operation informs the ResourceClient that proxy is about to lose access to a resource. The ResourceClient shall complete any clean-up that is needed before the resource is lost before it returns from this operation. This operation is not guaranteed to be allowed to complete before notifyRelease() is called.

**Parameters:**

proxy - the ResourceProxy representing the scarce resource to the application

**notifyRelease**

```
public abstract void notifyRelease(ResourceProxy proxy)
```

A call to this operation notifies the ResourceClient that proxy has lost access to a resource. This can happen for two reasons: either the resource is unavailable for some reason beyond the control of the environment (e.g. hardware failure) or because the client has been too long in dealing with a ResourceClient.release() call.

**Parameters:**

proxy - the ResourceProxy representing the scarce resource to the application

## F.3 Classes

### F.3.1 ResourceStatusEvent

```
java.lang.Object
   |
   +----org.davic.resources.ResourceStatusEvent
```

public class **ResourceStatusEvent**

extends Object

This class is the parent class for events reporting changes in the status of resources.

**Constructors**

```
public ResourceStatusEvent(Object source)
```

This constructs a resource status event relating to the specified resource. The precise class of the object will depend on the individual API using the resource notification API.

**Parameters:**