ETSI TS 102 809 V1.2.1 (2013-07)

Digital Video Broadcasting (DVB);
Signalling and carriage of interactive applications and services in Hybrid broadcast/broadband environments

Reference
RTS/JTC-DVB-326

Keywords
broadcasting, DVB, internet, IP, multimedia

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

The Digital Video Broadcasting Project (DVB) is an industry-led consortium of broadcasters, manufacturers, network operators, software developers, regulatory bodies, content owners and others committed to designing global standards for the delivery of digital television and data services. DVB fosters market driven solutions that meet the needs and economic circumstances of broadcast industry stakeholders and consumers. DVB standards cover all aspects of digital television from transmission through interfacing, conditional access and interactivity for digital video, audio and data. The consortium came together in 1993 to provide global standardization, interoperability and future proof specifications.

The consortium came together in 1993 to provide global standardization, interoperability and future proof specifications.

The normative XML schemas referenced by the present document are attached as separate files contained in archive ts_102809v010201p0.zip which accompanies the present document. The XML schemas included in the present document are informative.

1 Scope

The present document defines a framework for the signalling and carriage of interactive applications or services in broadcast and broadband networks. This framework covers:

- Signalling interactive applications or services in both classical broadcast networks and broadband networks.
- Distributing the files of interactive applications or services through either classical broadcast networks or broadband networks.
- Synchronizing interactive applications or services to video or audio content distributed through classical broadcast networks or broadband networks.
- Referencing video, audio or subtitle content distributed through classical broadcast networks or broadband networks from interactive applications or services.

The present document is independent of any particular technology for interactive applications or services. It is intended to be referenced by organizations defining how interactive applications or services are to be deployed and not used as a stand-alone document in its own right. It is expected that those organizations will make a selection appropriate for their market or deployment from among the functionality defined here. The use of "shall", "should" and similar terms in the present document is intended to apply only if the particular feature is used and not to imply that the feature itself is mandatory.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

[2] ETSI EN 301 192 (V1.3.1): "Digital Video Broadcasting (DVB); DVB specification for data broadcasting".

[3] ISO/IEC 13818-1: "Information technology - Generic coding of moving pictures and associated audio information: Systems".

[4] ISO/IEC 13818-6: "Information technology - Generic coding of moving pictures and associated audio information: Part 6: Extensions for DSM-CC".

[5] IETF RFC 2616: "Hypertext Transfer Protocol - HTTP/1.1".

[6] ETSI TS 102 034: "Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks".

[7] ISO 639-2:1998: "Codes for the representation of names of languages - Part 2: Alpha-3 code".

[8] ISO/IEC 8859-1: "Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1".

[9] Object Management Group: "The Common Object Request broker: Architecture and Specification".

NOTE: Available at http://www.omg.org/cgi-bin/doc?formal/97-09-01.pdf.

[10] IETF RFC 2045: "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies".

[11] IETF RFC 1950: "ZLIB Compressed Data Format Specification version 3.3".

[12] IETF RFC 1951: "DEFLATE Compressed Data Format Specification version 1.3".

[13] ETSI TS 102 823 (V1.1.1): "Digital Video Broadcasting (DVB); Specification for the carriage of synchronized auxiliary data in DVB transport streams".

[14] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

[15] ETSI TS 101 162: "Digital Video Broadcasting (DVB); Allocation of identifiers and codes for Digital Video Broadcasting (DVB) systems".

[16] ATSC A/100-5: "DTV application software environment level 1 (DASE-1); Part 5: ZIP archive resource format".

[17] W3C: "XML Schema Part 0: Primer Second Edition".

NOTE: Available at http://www.w3.org/TR/xmlschema-0/

[18] IETF RFC 1945: "Hypertext Transfer Protocol - HTTP/1.0".

[19] IETF RFC 2818: "HTTP over TLS".

[20] ETSI TS 102 851: "Digital Video Broadcasting (DVB); Uniform Resource Identifiers (URI) for DVB Systems".

[21] IETF RFC 1035: "Domain names - implementation and specification".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 102 727: "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.2.2".

[i.2] ETSI TR 101 202 (V1.1.1): "Digital Video Broadcasting (DVB); Implementation guidelines for Data Broadcasting".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

abstract service: mechanism to group a set of related unbound applications where some aggregator has taken the responsibility to ensure that the set of related applications work together

NOTE: This is a generalization of a broadcast service to support applications not related to any broadcast TV or radio service. A set of resident applications which an network operator has packaged together (e.g. chat, email, WWW browser) could comprise one abstract service.

application: collection of assets and logic that together provide a self-contained interactive service to the user

application lifecycle: various states in which an application may exist and the transitions between them, including starting and stopping

Application Programming Interface (API): interface between an application and a particular feature, function or resource of the terminal

classical broadcast network: network using classical broadcast technologies based on MPEG-2 transport streams carried over a physical layer such as DVB-T, DVB-S or DVB-C

platform specification: document which references the present document and defines which parts of the present document are applicable in a particular market or deployment

selected service: TV or radio service which is currently being presented by the receiver and whose application signalling is being monitored by the receiver

service: sequence of programmes under the control of a broadcaster which can be broadcast as part of a schedule

unbound application: application which is not associated with a broadcast service

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ADF	Application Description File
AFI	Authority and Format Identifier
AIT	Application Information Table
API	Application Programming Interface
AV	Audio Video
BIOP	Broadcast Inter-ORB Protocol
CORBA	Common Object Request Broker Architecture
CRC	Cyclic Redundancy Check
DDB	Download Data Block
DII	Download Info Indication
DNS	Domain Name System
DSI	Download Server Initiate
DSM-CC	Digital Storage Media - Command and Control
DSMCC	Digital Storage Media Command and Control
DTD	Document Type Definition
DVB	Digital Video Broadcasting
EIT	Event Information Table
EPG	Electronic Programme Guide
ETSI	European Telecommunications Standards Institute
HTML	Hyper Text Mark-up Language
HTTP	Hyper Text Transport Protocol
IEC	International Electrotechnical Commission
IOR	Interoperable Object References
IP	Internet Protocol
IPR	Intellectual Property Rights
ISO	International Organization for Standardization
LSB	Least Significant Byte
MHP	Multimedia Home Platform
MPEG	Moving Picture Expert Group
NPT	Normal Play Time
NSAP	Network Service Access Point
OC	Object Carousel
ORB	Object Request Broker
OUI	Organizationally Unique Identifier
PAT	Program Association Table
PID	Packet IDentifier
PMT	Program Map Table
PNG	Portable Network Graphics
PSI	Program Specific Information

PVR	Personal Video Recorder
SD&S	Service Discovery and Selection
SDT	Service Description Table
SI	Service Information
STC	System Time Clock
TS	Transport Stream
TV	TeleVision
UI	User Interface
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
WWW	World Wide Web
XML	eXtensible Markup Language

4 Application models

4.1 Introduction

The present document can enable a wide range of different application models depending on which of the optional features are selected. Below is a list of some of these, sorted from simplest to more complex.

- Applications bound to exactly one broadcast service which are started when that service is selected and stopped when that service is de-selected.
- Applications bound to more than one broadcast service which are started when any such service is selected, stopped when that service is de-selected, even if the de-selection is part of changing to a new service to which the application is also bound.
- Applications which persist across service changes are applications bound to more than one broadcast service that are started when any service to which they are bound is selected, run without interruption while any service to which they are bound remains selected and stopped when no longer bound to any currently selected service.
- Applications bound to a content item that is part of a broadcast service (for example an individual programme or adverts) will be started when that content item starts (if the service is selected at that time) and terminated when the content item finishes (if the service remains selected at that time).
- Applications bound to a content on demand item will either be handled identically to applications bound to parts of a broadcast service (including the possibility for dynamic changes during the content on demand item) or will be valid for the entire duration of the content item.
- Applications which are valid while the terminal is connected to a network operator or service platform provider. In some deployments, this may be permanent.

4.2 Starting and stopping applications

4.2.1 Applications bound to broadcast services

When a broadcast service is selected, the following shall apply:

- The terminal shall determine if there are any applications signalled as being related to the service as defined by clause 5.3 or clause 5.4 of the present document.
- Applications which are part of that service and which are signalled with a control code of AUTOSTART (see table 3) and which are not still running from a previously presented service shall be started.
- Applications which are part of that service, which are signalled with a control code of AUTOSTART and which are already running from a previously presented broadcast service shall continue to run uninterrupted. A second instance of the application shall not be started.

- Applications which are part of that broadcast service and which are signalled with a control code of PRESENT shall continue to run if already running but shall not be started if not already running.
- Running applications from any previously presented broadcast service which are not part of the new broadcast shall be stopped as part of the change of presented service.

While a broadcast service continues to be presented as defined above, the following apply:

- Applications which are added to the service with a control code of AUTOSTART shall be automatically started when their addition is detected by the terminal. Applications added to the service with any other control code shall not be automatically started.
- Applications which are part of the service whose control code changes to AUTOSTART from some other value shall be automatically started unless already running.
- Applications which are part of the service whose control code changes from AUTOSTART to PRESENT and which are already running shall continue to run.
- Applications whose control code changes to KILL or DESTROY shall be stopped as defined by the semantics which the application technology specification defines for those control codes.

When a broadcast service stops being selected, the following apply:

- Applications where the serviceBound element of the application_descriptor (see clauses 5.3.5.3 and 5.4.4.4) in their signalling has value true shall be stopped.
- When an application continues running after change of broadcast service, it shall run as signalled in the new service and not the former service.

4.2.2 Applications bound to a content on demand item

Applications bound to a content on demand item will either be handled identically to applications bound to parts of a broadcast service (including the possibility for dynamic changes during the content on demand item) or will be valid for the entire duration of the content item. In the latter case, applications whose control code is AUTOSTART shall be started when the content item starts being presented and shall be stopped when the content item stops being presented. Changes to application control codes are not possible.

4.2.3 Applications bound to a network operator

Applications bound to a network operator may run at any time when a terminal is connected to that operator's network. Applications with a control code of AUTOSTART shall be automatically started when the terminal is first connected to that network or when the application(s) are added to the signalling. Where terminals can change from one network operator to another, as part of this process, the former network operator's applications shall be stopped and the new operator's applications started.

5 Signalling interactive applications and services

5.1 Semantics

This clause covers the following topics:

- How the receiver identifies the applications associated with a service and finds the locations from which to retrieve them.
- The signalling that enables the broadcast to manage the lifecycles of applications.
- How the receiver can identify the sources of broadcast data required by the applications of a service.

Much of the signalling is generic. For example, the application_descriptor is independent of the application representation. Other signalling may be defined by the platform specification. URIs used in this signalling shall comply with the format and restrictions defined in [20].

5.2 Application metadata

5.2.1 Introduction

Applications may have a number of items of metadata associated with them. These are as follows:

- Type: Identifies the platform needed to run or present the application.
- Identifier: Identifies the application.
- Control code: Defines the lifecycle state of the application.
- Profile: Defines the minimum profile of terminal needed for this application.
- Visibility: Identifies whether the application is visible to the user or to other applications via an application listing API (where such an API is supported).
- Priority: Defines the priority of the application relative to other signalled applications.
- Icons: Identifies the location of icons for this application.
- Graphics constraints: Identifies any constraints on this application with respect to changes in graphics configuration or presented video.
- Storage information: Defines whether an application should be stored, and which application files should be stored.

In this clause, each sub-clause first defines the semantics for that item of information and then the MPEG-2 and XML based encodings of the item.

5.2.2 Application types

5.2.2.1 Semantics

All applications have an associated type in order that a terminal can discard applications whose types it does not support.

NOTE: The application type is not sufficient to guarantee that a terminal can run an application. For more information, see the metadata relating to application profiles in clause 5.2.5.

5.2.2.2 MPEG-2 Encoding

In the MPEG-2 encoding, application types are identified by a 15-bit number. This enables receivers to filter out signalling for unsupported application types. Defined application types are registered with DVB [15]. For historical reasons, application types are registered with the ID MHP_Application_Type_ID.

5.2.2.3 XML Encoding

In the XML encoding, application types are strings, typically a MIME type. See clause 5.4.4.11.

5.2.3 Application identification

5.2.3.1 Semantics

Each application has an associated application identifier. This consists of two parts, the organisation_id and the application_id as follows:

organisation_id: This field is a globally unique value identifying the organization that is responsible for the application. These values are registered with DVB [15]. Values of zero shall not be encoded. For compatibility with clause B.2.10 the most significant 8 bits of the organisation_id shall be zero.

Where applications are authenticated using X.509 certificates, this field is reproduced in the organisationName field of the subject name in the "leaf" certificate of an authenticated application.

NOTE: The inclusion of this field in the leaf certificate provides authentication of the value.

application_id: This field uniquely identifies the application. This is allocated by the organization registered with the organisation_id who decides the policy for allocation within the organization. Values of zero shall not be encoded.

The application_id values are divided into three ranges: one for unsigned applications, one for signed applications and one for specially privileged applications. This is for security reasons. Applications transmitted as unsigned shall use an application_id from the unsigned applications range and applications transmitted as signed shall use an application_id from the signed applications range. Applications transmitted as privileged shall use an application_id from the privileged applications range.

Table 1: Value ranges for application_id

application_id values	Use
0x0000	Shall not be used
0x0001 to 0x3fff	Application_ids for unsigned applications
0x4000 to 0x7fff	Application_ids for signed applications
0x8000 to 0x9fff	Application_ids for privileged applications
0xa000 to 0xfffd	Reserved for future use by DVB
0xfffe	Special wildcard value for signed applications of an organization
0xffff	Special wildcard value for all applications of an organization

Application_id values 0xffff and 0xfffe are wild cards. They shall not be used to identify an application but, for example, are allowed for use in the external_application_authorization_descriptor (see clause 5.3.5.7). The value 0xffff matches all applications with the same organisation_id. The value 0xfffe matches all signed applications with the same organisation_id.

The same application identifier may be used in different application types for applications performing essentially the same function.

5.2.3.2 MPEG-2 encoding

This is a 6 byte field with the following structure:

Table 2: Application identifier syntax

	No. of bits
application_identifier {	
organisation_id	32
application_id	16
}	

The same application_identifier() shall appear only once within the set of applications signalled for the same application type.

5.2.3.3 XML encoding

The application identifier is defined by the following XML fragment where the elements are defined above.

```
<xsd:complexType name="ApplicationIdentifier">
    <xsd:sequence>
        <xsd:element name="orgId" type="xsd:unsignedInt"/>
        <xsd:element name="appId" type="xsd:unsignedShort"/>
    </xsd:sequence>
</xsd:complexType>
```

5.2.4 Application control codes

The broadcast signalling provides a mechanism for broadcasters to control the lifecycle of standard application types.

5.2.4.1 Semantics

This control code allows the broadcaster to signal to the receiver what to do with the application with regard to its lifecycle. The set of codes have some differences between application types and precise semantics are defined on an application type specific basis.

If the receiver receives a code that it does not recognize, the application shall continue in its current state.

Table 3: Application control code values

MPEG-2 encoding	Identifier	Semantics
0x00		reserved_future_use
0x01	AUTOSTART	The application shall be started when the service is selected, unless the application is already running.
0x02	PRESENT	The application is allowed to run while the service is selected, however it shall not start automatically when the service becomes selected.
0x03	DESTROY	The application shall be stopped but may be permitted the opportunity to close down gracefully. Attempts to start the application shall fail.
0x04	KILL	The application shall be stopped as soon as possible. Attempts to start the application shall fail.
0x05	PREFETCH	Application files should be cached by the receiver, if possible. The application shall not be started and attempts to start it shall fail.
0x06	REMOTE	This identifies an application that is not available on the current transport stream and hence only available after tuning to a new transport stream or if cached and signalled as launchable completely from cache.
0x07	DISABLED	The application shall not be started and attempts to start it shall fail.
0x08	PLAYBACK_AUTOSTART	The application shall not be run, neither direct from broadcast nor when in timeshift mode. When a recording is being played back from storage, the application shall be presented as if it was autostart.
0x09 to 0xFF		reserved_future_use

Platform specifications should define which of these control codes are applicable.

5.2.4.2 MPEG-2 encoding

The application control code is signalled through the application_control_code field for the application in the AIT (see clause 5.3.4.6). The values are shown in table 3.

5.2.4.3 XML encoding

The XML encoding of the application control code is defined by the following fragment where the values are as defined in table 3:

```
<xsd:simpleType name="ApplicationControlCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AUTOSTART"/>
        <xsd:enumeration value="PRESENT"/>
```

```
        <xsd:enumeration value="DESTROY"/>
        <xsd:enumeration value="KILL"/>
        <xsd:enumeration value="PREFETCH"/>
        <xsd:enumeration value="REMOTE"/>
        <xsd:enumeration value="DISABLED"/>
        <xsd:enumeration value="PLAYBACK_AUTOSTART"/>
    </xsd:restriction>
</xsd:simpleType>
```

5.2.5 Platform profiles

5.2.5.1 Semantics

Some platform specifications may define a number of different profiles, and potentially different versions of those profiles. These fields define the minimum platform profile and version of that profile required by the application. Platform specifications that use this information need to define what it means in their context.

application_profile: This field is an integer value which represents the platform profile required by the application. This indicates that a receiver implementing one of the profiles listed in this loop is capable of executing the application.

version.major: This field carries the numeric value of the major sub-field of the profile version number.

version.minor: This field carries the numeric value of the minor sub-field of the profile version number.

version.micro: This field carries the numeric value of the micro sub-field of the profile version number.

The four above fields indicate the minimum profile on which an application will run. Applications may test for features found in higher (backwards compatible) profiles and exploit them. The terminal shall only launch applications if the following expression is true for at least one of the signalled profiles:

$$
\begin{aligned}
&(\text{application\_profile} \in \text{terminal\_profiles\_set}) \\
&\quad \wedge \{(\text{application\_version.major} < \text{terminal\_version.major(application\_profile)}) \\
&\quad\quad \vee [(\text{application\_version.major} = \text{terminal\_version.major(application\_profile)}) \\
&\quad\quad\quad \wedge (\{\text{application\_version.minor} < \text{terminal\_version.minor(application\_profile)}\} \\
&\quad\quad\quad \vee \{[\text{application\_version.minor} = \text{terminal\_version.minor(application\_profile)}] \\
&\quad\quad\quad\quad \wedge [\text{application\_version\_micro} \leq \text{terminal - version.micro(application\_profile)}]\})]\}
\end{aligned}
$$

Where:

$\in$ represents "belongs to the set of"

$\wedge$ represents "logical AND"

$\vee$ represents "logical OR"

NOTE: The encoding of these values may vary between application types and is defined by the interactive services technology specification.

5.2.5.2 MPEG-2 encoding

Profiles are encoded as part of the application descriptor as follows:

Table 4: Application profile encoding

	No. of Bits	Identifier	Value
application_profiles_length	8	uimsbf	
for(i=0; i<N; i++) {			
application_profile	16	uimsbf	
version.major	8	uimsbf	
version.minor	8	uimsbf	
version.micro	8	uimsbf	
}			

5.2.5.3 XML encoding

The XML encoding of the profiles is as follows:

```
<xsd:complexType name="MhpVersion">
    <xsd:sequence minOccurs="1">
        <xsd:element name="profile" type="ipi:Hexadecimal16bit "/>
        <xsd:element name="versionMajor" type="ipi:Hexadecimal8bit "/>
        <xsd:element name="versionMinor" type="ipi:Hexadecimal8bit "/>
        <xsd:element name="versionMicro" type="ipi:Hexadecimal8bit "/>
    </xsd:sequence>
</xsd:complexType>
```

NOTE: The name of the type is historical.

5.2.6 Application visibility

5.2.6.1 Semantics

The visibility field specifies whether the application is suitable to be offered to the end-user for them to decide if the application should be launched. Table 5 lists the allowed values of this field.

NOTE: This applies equally to any generic launching menu application provided by the content or service provider or the terminal manufacturer.

Table 5: Definition of visibility states for applications

MPEG-2 encoding	XML Encoding	Description
00	NOT_VISIBLE_ALL	This application shall not be visible either to applications via an application listing API (if such an API is supported by the terminal) or to users via the navigator with the exception of any error reporting or logging facility, etc.
01	NOT_VISIBLE_USERS	This application shall not be visible to users but shall be visible to applications via an application listing API (if such an API is supported by the terminal).
10		reserved_future_use
11	VISIBLE_ALL	This application can be visible to users and shall be visible to applications via an application listing API (if such an API is supported by the terminal).

This field is optional.

5.2.6.2 MPEG-2 encoding

Application visibility is encoded in the visibility field of the application descriptor in the AIT (see clause 5.3.5.3). Possible values for this field are given in table 5.

5.2.6.3 XML encoding

The XML encoding of the visibility is as follows:

```
<xsd:simpleType name="VisibilityDescriptor">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="NOT_VISIBLE_ALL"/>
        <xsd:enumeration value="NOT_VISIBLE_USERS "/>
        <xsd:enumeration value="VISIBLE_ALL"/>
    </xsd:restriction>
</xsd:simpleType>
```

See table 5 in clause 5.2.6.1 for the definition of these values.

5.2.7 Application priority

5.2.7.1 Semantics

The application priority identifies a relative priority between the applications signalled in a service:

- Where there is more than one application with the same application identification in a service, this priority shall be used to determine which application is started.
- Where there are insufficient resources to continue running a set of applications, this priority shall be used to determine which applications to stop or pause.
- A larger integer value indicates higher priority.
- If two applications have the same application identification and the same priority, the terminal may make an implementation-dependent choice on which to start.

NOTE: Platform specifications may define special semantics for specific priority values.

5.2.7.2 MPEG-2 encoding

Application priority is encoded in the application_priority field of the application_descriptor (see clause 5.3.5.3).

5.2.7.3 XML encoding

Application priority is encoded in the priority field of the application descriptor (see clause 5.4.4.4).

5.2.8 Application icons

5.2.8.1 Semantics

One or more icons may be associated with an application. The content format for these possible icons shall be PNG. Platform specifications may impose additional restrictions on the content format of icons.

Each icon has an icon locator and a set of flags that identify the size and aspect ratio of the icon.

The icon locator is the first part of the string that specifies the location of the icon files. It is relative to a location that depends on the application type. The icon locator shall not end with a "/" slash character.

The file names for the icon files are encoded in a standard way:

```
filename        = icon_locator "/dvb.icon." hex_string
hex_string      = 4*4hex
hex             = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"
digit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

An icon file shall contain exactly one icon. The icon contained in the icon file shall have the format specified by the 4 hexadecimal digit postscript of its file name. The value of this postscript is given by the corresponding MPEG-2 encoding of the icon flags (see table 7).

5.2.8.2 MPEG-2 encoding

Information relating to the application icons is encoded in the application_icons_descriptor.

Table 6: Application icons descriptor syntax

	No.of Bits	Identifier	Value
application_icons_descriptor() {			
descriptor_tag	8	uimsbf	0x0B
descriptor_length	8	uimsbf	
icon_locator_length	8	uimsbf	
for (i=0; i<N; i++) {			
icon_locator_byte	8	uimsbf	
}			
icon_flags	16	bslbf	
for (i=0; i<N; i++) {			
reserved_future_use	8	bslbf	
}			
}			

Possible values for the icon_flags field are given in table 7.

Table 7: Definition of different icon flags

Icon flag bits	Description of icon size and pixel aspect ratio
0000 0000 0000 0001	32 x 32 for square pixel display
0000 0000 0000 0010	32 x 32 for broadcast pixels on 4:3 display (See note)
0000 0000 0000 0100	24 x 32 for broadcast pixels on 16:9 display
0000 0000 0000 1000	64 x 64 for square pixel display
0000 0000 0001 0000	64 x 64 for broadcast pixels on 4:3 display (See note)
0000 0000 0010 0000	48 x 64 for broadcast pixels on 16:9 display
0000 0000 0100 0000	128 x 128 for square pixel display
0000 0000 1000 0000	128 x 128 for broadcast pixels on 4:3 display (See note)
0000 0001 0000 0000	96 x 128 for broadcast pixels on 16:9 display
0000 0010 0000 0000	256 x 256 for square pixel display
0000 0100 0000 0000	256 x 256 for broadcast pixels on 4:3 display (See note)
0000 1000 0000 0000	192 x 256 for broadcast pixels on 16:9 display
xxxx 0000 0000 0000	reserved_future_use
NOTE: Approximatively 15/16 pixel aspect ratio on 50 Hz system.	

If the icon_flags field of the application icons descriptor were to have a value indicating the presence of multiple icons, each of the indicated icons would have its own icon file. For example, if icon_flags has a value of 0x0005, the directory specified by icon_locator would contain two files named dvb.icon.0004 (for 24 x 32 square pixel rendering) and dvb.icon.0001 (for 32 x 32 square pixel rendering).

5.2.8.3 XML encoding

Icon information is encoded in one or more IconDescriptor elements:

```
<xsd:complexType name="IconDescriptor">
    <xsd:attribute name="filename" type="xsd:string" use="required"/>
    <xsd:attribute name="size" type="xsd:unsignedShort" use="optional"/>
    <xsd:attribute name="aspectRatio" type="mhp:AspectRatio" use="optional"/>
</xsd:complexType>
```

NOTE 1: The MPEG-2 and XML encodings are intentionally different. The MPEG-2 encoding only carries the icon_locator prefix and the remainder of the filename is computed. The XML encoding carries the complete URL.

NOTE 2: The mhp:AspectRatio type is defined in clause 5.4.4.7 of the present document.

The size and aspectRatio attributes are defined as optional since they can be determined from the 4 hexadecimal digit postscript of its file name, as defined in table 7.

5.2.9 Graphics constraints

5.2.9.1 Semantics

Applications may be constrained in the graphics resolutions they support, or in their ability to handle changes in the graphics or video configuration.

Constraints on the graphical capabilities of an application can be specified using a number of fields, defined below.

Applications where this information is not signalled shall be assumed to have the following graphics constraints:

- Supports standard definition video.
- Cannot run without a visible UI.
- Cannot handle changed graphics configurations.
- Cannot handle externally controlled video.

Applications where the set of signalled graphics configurations is empty shall be assumed to not care about a default graphics configuration. Either they do not use graphics or they are written to support the full range of graphics configurations defined in the present document and tested accordingly.

5.2.9.1.1 Supported graphics configurations

Supported graphics configurations for an application are given by a list of one or more of supported configurations (listed in table 9). The full screen configurations are sorted from most preferred to least preferred.

5.2.9.1.2 Running without a visible UI

The can_run_without_visible_ui flag indicates whether the application needs to display a user interface. If this flag is set then the application can usefully run with no user interface visible. If this flag is not set then the application can only usefully run with a user interface visible. Applications signalled with this flag set are responsible for detecting when it would be reasonable to show their user interface again and requesting this as defined in the platform specification.

5.2.9.1.3 Handling changed graphics configurations

The handles_configuration_changed flag indicates whether the application is capable of handling changes in the graphics configuration. If this flag is set then the application can handle changes in the graphics configuration between the supported graphics configurations for this applications (see clause 5.2.9.1.1). If this flag is not set then once the default graphics configuration has been set for an application instance, it will only correctly display under that graphics configuration.

5.2.9.1.4 Handling externally controlled video

The handles_externally_controlled_video flag indicates whether an application can usefully run when the presentation of the video is under the control of a second application external to its service. If this flag is set then the application can handle being displayed under these circumstances. Examples include picture in picture and picture outside picture.

5.2.9.2 MPEG-2 encoding

The graphics_constraints_descriptor signals which constraints apply to the application.

Table 8: Graphics constraints descriptor syntax

	No. of bits	Identifier	Value
graphics_constraints_descriptor() {			
descriptor_tag	8	uimsbf	0x14
descriptor_length	8	uimsbf	
reserved_future_use	5	bslbf	
can_run_without_visible_ui	1	bslbf	
handles_configuration_changed	1	bslbf	
handles_externally_controlled_video	1	bslbf	
for(i=0;i<N;i++) {			
graphics_configuration_byte	8	uimsbf	
}			
}			

Supported graphics configurations for an application are given by a list of one or more of the values listed in table 9.

Table 9: Graphics configuration byte values

Value	Meaning
0	Reserved
1	Full screen standard definition
2	Full screen 960x540
3	Full screen 1 280x720
4	Full screen 1 920x1080
5 to 31	Reserved for future use by DVB project
32 to 255	Reserved for future use

5.2.9.3 XML encoding

The XML encoding is not defined in the present document.

5.2.10 Application usage

5.2.10.1 Semantics

This identifies that the application provides a specific, well-known, service; for example teletext, EPG or chat. Terminals may include a shortcut to start these services, for example a remote control key. Terminals may also include a native UI offering access to these services.

5.2.10.2 MPEG-2 encoding

Table 10: Application usage descriptor

	No. of bits	Identifier	Value
application_usage_descriptor() {			
descriptor_tag	8	uimsbf	0x16
descriptor_length	8	uimsbf	
usage_type	8	uimsbf	
}			

descriptor_tag: This 8 bit field with value 0x16 identifies the descriptor.

usage_type: This 8 bit field indicates which service is provided by the application. It shall be coded according to table 11.

Table 11: MPEG-2 encoding of application usage types

Type Value	Description
0x00	reserved
0x01	Digital Text application
0x02 to 0x7F	reserved for future use
0x80 to 0xFF	usable by platform specifications (see note)
NOTE: Platform specification should define the domain in which these values are applicable, e.g. using a specific data broadcast ID.	

Platform specifications should define which of these usage types are applicable.

If no application_usage_descriptor is present then an application does not provide a specific well-known service.

5.2.10.3 XML encoding

The XML encoding of the application usage type is given by the ApplicationUsageDescriptor element:

```
<xsd:complexType name="ApplicationUsageDescriptor">
    <xsd:sequence>
        <xsd:element name="ApplicationUsage" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

The ApplicationUsage element indicates which service is provided by the application. It shall be coded according to table 12.

Table 12: XML encoding of application usage types

ApplicationUsage Value	Description
urn:dvb:mhp:2009:digitalText	Digital Text application

Platform specifications extending the usage types should prefix the value with the relevant namespace identifier for the platform specification.

5.2.11 Stored applications

5.2.11.1 Semantics

Storable applications follow the standard application signalling model defined in the present document. A terminal that does not provide storage or does not recognize these extensions will perceive storable applications as viable applications that potentially can be run from the broadcast connection. The signalling described in this clause augments the signalling described elsewhere in the present document.

The primary focus of stored applications is to improve the startup behaviour of applications delivered over broadcast connections. This feature is less relevant for applications delivered over a broadband connection.

5.2.11.1.1 Lifecycle of stored applications

Stored applications may be broadcast related or stand alone.

If an application is broadcast related then the application's life cycle is controlled by the broadcast service. Such applications are suitable for caching but not for running as a stand alone application. For stored broadcast related applications the broadcast signalling shall be used when launching the stored application. So, little information from the signalling needs to be stored with the application.

- If the application is stand alone then the application can usefully be launched by the user independently of a broadcast service. Applications with this property may also be launched as if broadcast related if the currently selected service lists them in its signalling. For stored stand-alone applications, the signalling used when launching the application shall come from a stored representation of the AIT.

5.2.11.1.2 Application versioning

The version field provides the version number of the application. This number starts at zero and increments by one each time any of the files listed in the Application Description File (see clause 5.2.12) change or the contents of the Application Description File itself change. Used values shall never be reused. In the event that the number range is exhausted a new application_id shall be used.

The is_launchable_with_older_version flag indicates whether an older cached version of an application may be run even though a higher version is signalled in the broadcast. When set, the terminal shall start the cached application where the version number of the cached application is lower than or equal to the version number of the broadcast application. If the version number of the cached application is higher than the version number of the broadcast application, the cached application shall not be started. If the flag is not set, the cached application shall not be started.

NOTE: If this flag is set, the cached application is responsible to handle version conflicts between cached application code and broadcast application data.

5.2.11.1.3 Launching applications from the cache

The launchable_completely_from_cache flag indicates whether a connection to a transport protocol is required. When set, this indicates that this application can be run entirely from cache, without connecting to the transport protocol signalled in the application's signalling, assuming that all the critical files have been cached. If the flag is not set, a connection to the transport protocol needs to be made in order to run this application as a broadcast-related application. This flag only applies if the application is being run as a broadcast related application; it is ignored when storing an application into a stored service, as all applications signalled as stand-alone can run without a connection to the transport protocol when run as part of a stored service.

This flag shall only be set when the not_launchable_from_broadcast flag is also set.

NOTE: This flag should be set only for applications where the object carousel is not present at all.

The not_launchable_from_broadcast flag indicates whether an application can be usefully launched before it is completely cached. When set, this indicates that the delivery characteristics of this application are such that it is not useful to launch the application unless it has been completely cached/stored. If the flag is not set, then caching provides some benefit but is not essential.

Applications in stored services and applications where the not_launchable_from_broadcast flag is set shall only be launched where the terminal has stored the complete set of files which are listed in the Application Description File as being critical.

Table 13: Storage descriptor flag combinations

not_launchable_from_broadcast	launchable_completely_from_cache	is_launchable with_older_version	Description
0	0	0	Normal case.
0	0	1	Shall not be signalled.
0	1	0	Shall not be signalled.
0	1	1	Shall not be signalled.
1	0	0	Runs if signalled version is stored.
1	0	1	Runs if signalled or older version is stored.
1	1	0	Runs completely from cache if signalled version is stored. The application cannot be stored due to unavailability of the object carousel for the current service.
1	1	1	Runs if signalled or older version is stored. The application cannot be stored due to unavailability of the object carousel for the current service.
When set, flag indicates that files are present but bitrate is too low.	When set, flag indicates that files are not present in current broadcast at all.		

5.2.11.1.4 Storage priority

The storage priority of an application indicates the priority of this application for storage relative to the other applications signalled in this service. It is only meaningful for applications which have been proactively cached by the terminal implementation and shall be ignored otherwise.

Higher values indicate more important applications to store. The behaviour when applications have the same priority is implementation dependent.

5.2.11.2 MPEG-2 encoding

Information about the storage capabilities for an application is carried in the application storage descriptor.

Table 14: Syntax of application storage descriptor

	No.of Bits	Identifier	
application_storage_descriptor() {			
descriptor_tag	8	uimsbf	0x10
descriptor_length	8	uimsbf	
storage_property	8	uimsbf	
not_launchable_from_broadcast	1	bslbf	
launchable_completely_from_cache	1	bslbf	
is_launchable_with_older_version	1	bslbf	
Reserved	5	bslbf	
Reserved	1	bslbf	
Version	31	uimsbf	
Priority	8	uimsbf	
}			

The storage_property field is encoded as follows:

Table 15: Semantics of storage property values

storage_property	Property
0	broadcast related
1	stand alone
2 to 255	reserved

5.2.11.3 XML encoding

The XML encoding of the storage capabilities for an application is given by the StorageCapabilities element:

```
<xsd:complexType name="StorageCapabilities">
    <xsd:sequence minOccurs="0">
        <xsd:element name="storageProperty" type="mhp:StorageType"/>
        <xsd:element name="isStorable" type="xsd:boolean"/>
        <xsd:element name="canCache" type="xsd:boolean"/>
    </xsd:sequence>
    <xsd:attribute name="launchableFromBroadcast" type="xsd:boolean" use="required"/>
    <xsd:attribute name="launchableCompletelyFromCache" type="xsd:boolean" use="required"/>
    <xsd:attribute name="launchableWithOlderVersion" type="xsd:boolean" use="required"/>
</xsd:complexType>
```

NOTE 1: The mhp:StorageType type is defined in clause 5.4.4.10 of the present document.

NOTE 2: The storage priority is not defined in the XML encoding.

5.2.12 Application Description File

5.2.12.1 Description

The Application Description File (ADF) provides the list of files that need to be stored for an application as well as other related necessary information. The notation uses an XML-based syntax.

For those applications that can be stored, an Application Description File shall be placed in the same carousel as the application.

NOTE: The Application Description File does not duplicate all the information needed to run the application. The terminal also needs to use the information in the application signalling when installing the application.

Where a file is listed in the Application Description File of more than one application and is stored, the terminal shall ensure that each application sees the correct version of the file for that application. The version of the file visible to one application shall not be changed by any changes in the version of the file visible to any other application which may share that same file.

5.2.12.2 Application description file name and location

The location of an ADF should be defined in a platform specification. By convention, the name of an ADF is:

```
'dvb.storage.oooooooo.aaaa'
```

where:

oooooooo is the organisation_id of the application as a 8 character hexadecimal string

aaaa is the application_id as a 4 character hexadecimal string

The organisation_id and application_id shall be padded with leading zeros to the specified length.

Lowercase hex digits shall be used to encode the organisation_id and application_id.

5.2.12.3 Syntax

The syntax of the Application Description File is defined by the following XML DTD.

The PublicLiteral to be used for specifying this DTD in document type declarations of the XML files is:

```
"-//DVB//DTD Application Description File 1.0//EN"
```

and the URL for the SystemLiteral is:

```
"http://www.dvb.org/mhp/dtd/applicationdescriptionfile-1-0.dtd"
```

```
<!ENTITY % object "(dir|file)">

<!-- the main element for the application description -->
<!ELEMENT applicationdescription (%object;)+>

<!ATTLIST applicationdescription version NMTOKEN #REQUIRED>

<!ELEMENT dir (%object;)*>
<!ATTLIST dir
    name CDATA #REQUIRED
    priority NMTOKEN #IMPLIED
>

<!ELEMENT file EMPTY>
<!ATTLIST file
    name CDATA #REQUIRED
    priority NMTOKEN #IMPLIED
    size NMTOKEN #REQUIRED
>
```

5.2.12.4 Semantics

version: A decimal integer denoting the version number of this application.

The value of this attribute shall not contain leading zeros (unless it is "0"). The value of this attribute shall also match the version number signalled in the version field of the application storage descriptor in the AIT entry of this application; if it does not, the application description file is invalid. (This field allows application authors to ensure that the version number signalled in the AIT is correct. If it is wrong then this prevents any files being stored.)

Name: This attribute provides the name of a file system object (directory or file) that is storable. This is the name of the object within its enclosing directory and hence does not include any directory path information. For the name attribute of a file element only, the last character of the name can be the wild-card character "*". This character will match any string including an empty string.

If name is "." or "..", contains the path separator character "/", or contains the character NUL (U+0000), then receivers shall reject the ADF as invalid.

NOTE 1: No elements are provided for naming object types such as Stream or StreamEvent which are carried in an object carousel, therefore there is no mechanism to specify that Stream and StreamEvent objects are required to be stored.

NOTE 2: Listing a directory object in the file does not imply anything about those contents of the directory which are not themselves listed in the file.

NOTE 3: Specifications which reference the present document may impose restrictions on file names which may be used.

Paths are relative to the directory containing the application description file, i.e. <file> and <dir> elements immediately inside the <applicationdescription> element refer to files and directories in the same directory as the application description file.

Priority: This attribute describes how important it is to store this object. The value shall be between 0 and 255, inclusive. If it is outside this range then the application description file is invalid. The value zero indicates that it is critical to store the object (i.e. there is no benefit in storing any objects unless this part is stored). Higher values indicate lower storage priority.

The default value for the priority attribute is zero (i.e. critical).

The priority for an object inherits from the immediately enclosing directory.

Size: This attribute defines the size in bytes of the file, or files where the name attribute includes a wild-card.

5.3 MPEG-2 table and section syntax

5.3.1 Summary

5.3.1.1 Summary of common signalling

The minimum signalling requirements for any applications are summarized as follows:

- PMT with application signalling descriptor to identify the service component carrying the Application Information Table.
- Application Information Table with the following information in its common descriptor loop:
  - transport_protocol_descriptor (all applications descriptions shall be within the scope of at least one transport_protocol_descriptor. These can be placed in either or both of the descriptor loops).
- Application Information Table with the following information in its application information descriptor loop:
  - application_descriptor;
  - application_name_descriptor.

5.3.1.2 Summary of additional signalling for applications carried via OC

In either the "common" (first) descriptor loop or the "application" (inner) descriptor loop:

- transport_protocol_descriptor, with the selector bytes containing the OC specific information as defined in table 31.

5.3.1.3 How to add a new scheme (informative)

The signalling scheme is intended to be extensible with regard to the application representations and transport protocols that are supported. The areas that need to be addressed when doing this are summarized below.

To add further transport protocols:

- Extend table 30 "Semantics of selector bytes".
- Possibly define further specialist descriptors such as the IP_signalling_descriptor.

To add further application representations:

- Define further specialist descriptors if needed (see clause 10.9 "DVB-J specific descriptors" in the MHP specification [i.1] for examples).
- Define the application type specific life cycle control codes in clause 5.2.4 "Application control codes".

Where constant values are registered by the present document extend the table 38 "Registry of constant values".

5.3.2 Program specific information

The elementary stream (inner) loop of the PMT for a DVB service supporting one or more applications shall reference streams for the following:

- Location of the stream transporting the Application Information Table.
- Location of the stream(s) transporting the application code and data.

5.3.2.1 Application signalling stream

The elementary stream information for the PMT entry describing the elementary stream carrying the Application Information Table has the following characteristics:

- The stream_type is set to 0x05 (ISO/IEC 13818-1 [3], private sections).
- An application_signalling_descriptor (see clause 5.3.5.1).

There may be more than one elementary stream carrying application signalling information for a service.

5.3.2.2 Data broadcast streams

The minimum signalling in the PMT associated with data broadcast components is the value of the PMT stream_type field required by the DVB data broadcasting specification (EN 301 192 [2]) for the transport protocol. The full details of the data broadcast protocol, the location of its "principal" component etc. are provided in the AIT (see clause 5.3.4 "Application Information Table").

Optionally, the PMT may include data_broadcast_id_descriptors.

NOTE: Inclusion of data_broadcast_id_descriptors enables receivers to start mounting the file system that delivers applications concurrently with acquiring the AIT that identifies which applications are of interest. Enabling this concurrent operation may allow receivers to accelerate their activation of an interactive application. See clause B.2.8 "Mounting an object carousel".

The data_broadcast_id_descriptor identifies the "principal" component of the data broadcast. The detailed semantics of this optional signalling reflects the transport protocol. For example, in the case of a DVB object carousel it identifies the component carrying the DSI.

There may also be certain protocol specific descriptors in the PMT. For example, the object carousel requires the inclusion of the carousel_identifier_descriptor (see clause B.2.8 "Mounting an Object Carousel").

In its minimum form (with no selector information) a data broadcast id descriptor just identifies the "principal" component. This optionally may be extended with selector information that identifies the application types of the autostart applications delivered by that data broadcast. See clause 5.3.5.2 "Data broadcast id descriptor".

5.3.3 Notation

5.3.3.1 reserved

The term "reserved" when used in the clause defining the coded bit stream, indicates that the value may be used in the future for ISO defined extensions. Unless otherwise specified within the present clause all "reserved" bits shall be set to "1".

5.3.3.2 reserved_future_use

The term "reserved_future_use", when used in the clause defining the coded bit stream, indicates that the value may be used in the future for ETSI defined extensions. Unless otherwise specified within the present clause all "reserved_future_use" bits shall be set to "1".

5.3.4 Application Information Table

The Application Information Table (AIT) provides full information on the data broadcast, the required activation state of applications carried by it, etc. The AIT comprises the set of AIT sub-tables (see clause 5.3.4.5) within the selected service which have an application_type that the receiver can decode.

Data in the AIT allows the broadcaster to request that the receiver change the activation state of an application.

5.3.4.1 Data errors

AITs which contain errors shall be processed as follows:

- An error in a descriptor shall result in that descriptor being silently discarded. Processing of that descriptor loop shall continue with the next descriptor (if any). The scope of error detection of a descriptor should be limited to the application information section in which it is carried.
- An error in an application loop outside a descriptor shall result in that entry in the application loop being silently discarded. Processing of that application loop shall continue with the next entry (if any).

NOTE: The consequence of the above is that an error in a mandatory descriptor which results in that descriptor being silently ignored may then result in an application loop which is missing such a mandatory descriptor. Hence that application loop is silently ignored.

- An error in an application information section outside of an application loop shall result in that entire application information section being silently discarded. Processing of the AIT shall continue with the next application information section (if any).

5.3.4.2 AIT transmission and monitoring

Terminals shall monitor the PMT for changes in the number of AIT elementary streams present. The time within which changes shall be detected is application type dependent. Terminals shall monitor all AIT elementary streams within the selected service, as described in more detail below.

The minimum repetition rate for each AIT sub-table should be defined by the platform specification.

Provided that AITs for the selected service are delivered on 3 or fewer elementary streams then the maximum time interval between the moment the AIT is updated and the moment the new version is detected by the terminal should be defined by the platform specification.

NOTE: If broadcasts use more than 3 elementary streams to deliver AITs then receiver response time may degrade in an unpredictable way.

The terminal is only required to monitor AIT sections for application types that it can decode. In this case, the application signalling may only be passed on for a subset of the application types being broadcast, in the case where the broadcast carries a superset of the terminal's capabilities.

Applications removed from the AIT sub-table which was signalling them but where that AIT sub-table remains present in the network, shall be stopped as if they had been signalled with a DESTROY control code.

If the AIT sub-table signalling an application vanishes from the network completely, that application shall continue to run. The terminal shall monitor for the re-appearance of the AIT sub-table as defined for the appearance of new AIT sub-tables above while that service remains selected.

5.3.4.3 Optimized AIT signalling

The optional AIT_version_number carried by the application_signalling_descriptor allows a possible optimization of receiver burden as it allows receivers to acquire the AIT only after they see changes in the AIT version advertised in the PMT.

See clause 5.3.5.1 "Application signalling descriptor".

5.3.4.4 Visibility of AIT

If an application tunes away from a transport stream where its signalling is carried without selecting a new service, it will continue running although the AIT is not visible.

In terminals with multiple network interfaces, if the AIT of the selected service is visible via any of them, then the AIT signalling is used as normal.

5.3.4.5 Definition of sub-table for the AIT

All sections on the same PID with the AIT table_id and the same value of application_type are members of the same sub-table.

5.3.4.6 Syntax of the AIT

The Application Information Section describes applications and their associated information. Each Application Information Section includes one "common" descriptor loop at the top level for descriptors that are shared between applications of that sub table and a loop of applications. Each application in the application loop has an "application" descriptor loop containing the descriptors associated with that application.

Like DVB SI tables, the scope of common loop descriptors is the sub-table. So, any descriptors present in the common descriptor loop apply to all sections of the sub-table. Typically, common descriptors would normally only be present in section 0 of a sub-table, unless there was not enough space.

Like other DVB SI tables, any strings contained in these tables shall not have null terminations.

Table 16: Application Information Section syntax

	No. of bits	Identifier
application_information_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
test_application_flag	1	bslbf
application_type	15	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_use	4	bslbf
common_descriptors_length	12	uimsbf
for(i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
application_loop_length	12	uimsbf
for(i=0;i<N;i++){		
application_identifier()		
application_control_code	8	uimsbf
reserved_future_use	4	bslbf
application_descriptors_loop_length	12	uimsbf
for(j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

table_id: This 8 bit integer with value 0x74 identifies this table.

section_syntax_indicator: The section_syntax_indicator is a 1-bit field which shall be set to "1".

section_length: This is a 12-bit field, the first two bits of which shall be "00". The remaining 10 bits specify the number of bytes of the section starting immediately following the section_length field, and including the CRC_32. The value in this field shall not exceed 1 021 (0x3FD).

test_application_flag: This 1-bit field when set indicates an application which is transmitted for the purposes of receiver testing and which shall not be started or listed in any API or displayed in any user interface by receivers under normal operational conditions. The means (if any) by which a receiver is put into a mode where applications signalled with this bit set are treated as if this field is set to zero is implementation dependent but should not be one which typical end-users might discover on their own.

application_type: This is a 15-bit field which identifies the type of the applications described in this AIT sub_table. See clause 5.2.2.2.

version_number: This 5-bit field is the version number of the sub_table. The version_number shall be incremented by 1 when a change in the information carried within the sub_table occurs. When it reaches value "31", it wraps around to "0".

current_next_indicator: This 1-bit indicator shall be set to "1".

section_number: This 8-bit field gives the number of the section. The section_number of the first section in the sub_table shall be "0x00". The section_number shall be incremented by 1 with each additional section with the same table_id, and application_type.

last_section_number: This 8-bit field specifies the number of the last section (that is, the section with the highest section_number) of the sub_table of which this section is part.

common_descriptors_length: This 12-bit field gives the total length in bytes of the following descriptors. The descriptors in this descriptor loop apply for all of the applications contained in this AIT sub_table.

application_control_code: This 8-bit field controls the state of the application. The semantics of this field is application type dependant. See clause 5.2.4 "Application control codes".

application_loop_length: This 12-bit field gives the total length in bytes of the following loop containing application information.

application_identifier(): This 48 bit field identifies the application. The structure of this field is defined in clause 5.2.3 "Application identification".

application_descriptors_loop_length: This 12-bit field gives the total length in bytes of the following descriptors. The descriptors in this loop apply to the specific application.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in annex B of EN 300 468 [1] after processing the entire section.

5.3.4.7 Use of private descriptors in the AIT

Private descriptors may be included in the AIT provided that they are in the scope of a DVB-SI EN 300 468 [1] private data specifier descriptor. The scope rules for the private data specifier descriptor are as follows:

- If this descriptor is located within any descriptor loop of the AIT, then any specifier identified within this descriptor loop applies to all following descriptors and user-defined values in the particular descriptor loop until the end of the descriptor loop, or until another occurrence of a private_data_specifier_descriptor.
- The use of the descriptor in the common (first) descriptor loop does not apply to descriptors or user-defined values in the application (second) descriptor loop.

5.3.4.8 Text encoding in AIT

Unless otherwise specified, all fields interpreted as text strings in the AIT shall be encoded as UTF8, but shall not include the null character.

5.3.4.9 Access to an MPEG-2 format AIT via a broadband connection

The AIT file contains the MPEG-2 encoding of an AIT in a form that may be loaded via HTTP and is used to group applications that are not associated with a broadcast service.

NOTE: Platform specifications which include support for the AIT file should define how this is used; for example, passing an HTTP URL which refers to this file to a platform-defined API call.

Platform specifications should define any requirements for monitoring or polling an AIT file for changes.

5.3.4.9.1 Syntax

The interaction channel encoding of the AIT into the AIT file is as follows:

- A single file shall contain all of the data.
- The file shall contain a concatenation of Application Information Sections (specified in clause 5.3 of the present document).
- The possibly multiple sections shall be ordered as follows:
  - Ascending order of application_type.
  - Within a single value of application_type in ascending order of section_number.
- All sections shall have current_next_indicator set to "1".
- Only the AUTOSTART and PRESENT application control codes (see table 3) are appropriate.

5.3.4.9.2 Syntactic restrictions

5.3.4.9.2.1 Transport protocols

The only allowed protocol_id has the value 0x0003. See table 29 "Protocol_id".

5.3.4.9.3 MIME type

The MIME type for an AIT file shall be "application/vnd.dvb.ait". The file extension shall be ".ait". Implementations may also encounter the MIME type "application/dvb.ai" for the AIT used for backwards compatibility. Use of this MIME type is not recommended for new applications, deployments of services.

5.3.5 Generic descriptors

5.3.5.1 Application signalling descriptor

The application_signalling_descriptor is defined for use in the elementary stream loop of the PMT where the stream_type of the elementary stream is 0x05. It identifies that the elementary stream carries an Application Information Table.

The application_signalling_descriptor optionally carries a loop of application_type and AIT_version_number pairs. These allow the descriptor to optionally reproduce the current version number state of the associated Application Information Table. This allows the receiver to be informed of the version of the AIT as a side effect of monitoring the PMT (which is expected to be monitored closely, under normal conditions). See clause 5.3.4.3 "Optimized AIT signalling".

When the receiver detects a change of the content of the application_signalling_descriptor, it shall acquire the new version of the AIT and respond accordingly.

The presence of the application_type and AIT_version_number subfields is optional. If not present then the AIT transmission and monitoring applies, see clause 5.3.4.2 "AIT transmission and monitoring".

Table 17: Application signalling descriptor syntax

	No. of bits	Identifier
application_signalling_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for(i=0; i<N; i++){		
reserved_future_use	1	
application_type	15	uimsbf
reserved_future_use	3	bslbf
AIT_version_number	5	uimsbf
}		
}		

descriptor_tag: This 8 bit integer with value 0x6F identifies this descriptor.

descriptor_length: This 8 bit field indicates the number of bytes following the descriptor length field.

application_type: This 15 bit field identifies the application type of an Application Information Table sub-table that is on this elementary stream.

AIT_version_number: This 5 bit field provides the "current" version number of the Application Information Table sub-table identified by the application_type field.

5.3.5.2 Data broadcast id descriptor

The data_broadcast_id_descriptor is defined for use in the elementary stream information of the PMT. The descriptor identifies:

- The transport format of the data broadcast whose "principal component" is on this elementary stream.

  The semantics of "principal component" is transport protocol specific.

- The set of application types for any autostart applications delivered by the data broadcast.

A single elementary stream may have more than one data_broadcast_id_descriptor to indicate conformance with more than one data broadcast specification. In addition, more than one data_broadcast_id_descriptor may be used to list additional application types within the scope of a particular data broadcast id.

More than one elementary stream may have a data_broadcast_id_descriptor indicating that auto start applications are carried by more than one delivery mechanism (for example a single service may have more than one object carousel delivering auto start applications).

5.3.5.2.1 Generic descriptor

The data_broadcast_id_descriptor is defined in a generic form by EN 300 468 [1] (illustrated in table 18). Where no "id specific data" is provided the descriptor just identifies the "principal" component of a data broadcast.

Table 18: Generic data broadcast id descriptor syntax

	No.of Bits	Identifier	Value
data_broadcast_id_descriptor() {			
descriptor_tag	8	uimsbf	0x66
descriptor_length	8	uimsbf	
data_broadcast_id	16	uimsbf	
for (i=0; i<N; i++) {			
id specific data	8	bslbf	
}			
}			

5.3.5.2.2 Data broadcast id descriptor for interactive application

When the data_broadcast_id is 0x00F0 or 0x00F1, (see table 38) the syntax of the data_broadcast_id_descriptor is as shown in table 19. This extends the generic descriptor with an optional list of application types for which autostart applications may exist within the data broadcast. This list provides a hint to allow the terminal to prioritize connection to a data broadcast when several are provided by the service. If no list is provided then the data_broadcast_id_descriptor is silent on the types of autostart applications that may be carried by the data broadcast. If the application list is not empty, then the data broadcast shall not include autostart applications of application types other than those in the list. It is not required that the data broadcast always include autostart applications of all types in the list.

Table 19: data_broadcast_id_descriptor syntax for interactive applications

	No.of Bits	Identifier	Value
data_broadcast_id_descriptor() {			
descriptor_tag	8	uimsbf	0x66
descriptor_length	8	uimsbf	
data_broadcast_id	16	uimsbf	
for (i=0; i<N; i++) {			
reserved_future_use	1		
application_type	15	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x66 identifies this descriptor.

data_broadcast_id: This 16 bit field indicates the format of the data broadcast transport protocol. These values are registered at http://www.dvb.org.

application_type: This 15 bit field indicates the type of the application. See clause 5.2.2 of the present document.

5.3.5.3 Application descriptor

Exactly one instance of the application_descriptor shall be contained in every "application" (inner) descriptor loop of the AIT.

Table 20: Application descriptor syntax

	No.of Bits	Identifier	Value
application_descriptor() {			
descriptor_tag	8	uimsbf	0x00
descriptor_length	8	uimsbf	
application_profiles_length	8	uimsbf	
for(i=0; i<N; i++) {			
application_profile	16	uimsbf	
version.major	8	uimsbf	
version.minor	8	uimsbf	
version.micro	8	uimsbf	
}			
service_bound_flag	1	bslbf	
visibility	2	bslbf	
reserved_future_use	5	bslbf	
application_priority	8	uimsbf	
for(i=0; i<N; i++) {			
transport_protocol_label	8	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x00 identifies this descriptor.

application_profiles_length: This 8-bit field indicates the length of the application_profile loop in bytes.

application_profile: This 16-bit field identifies which application type specific profile is required by this application. See clause 5.2.5 "Platform profiles".

version.major: This 8-bit field indicates the major version number of the profile. See clause 5.2.5 "Platform profiles".

version.minor: This 8-bit field indicates the minor version number of the profile. See clause 5.2.5 "Platform profiles".

version.micro: This 8-bit field indicates the micro version number of the profile. See clause 5.2.5 "Platform profiles".

service_bound_flag: If this flag is set to "1", the application is only associated with the current service and so the process of killing the application shall start at the beginning of the service change regardless of the contents of the destination AIT.

visibility: This 2-bit field indicates whether the application is visible to other applications via an application listing API (if supported by the platform) or to users. See clause 5.2.6 "Application visibility".

application_priority: This 8-bit field identifies the priority of the application relative to other signalled applications. See clause 5.2.7 "Application priority".

transport_protocol_label: This 8-bit field identifies a transport protocol that delivers the application. See transport_protocol_label in clause 5.3.6 "Transport protocol descriptors".

If more than one protocol is signalled then each protocol is an alternative delivery mechanism. The ordering indicates the broadcaster's view of which transport connection will provide the best user experience (first is best). This may be used as a hint by terminal implementations. It shall be evaluated only once during the life time of the application.

The protocol selection by the terminal may depend on a variety of factors including user preferences and the performance of the transport connections to the terminal.

5.3.5.4 Application recording descriptor

The application_recording_descriptor can be signalled in the application descriptor loop of the AIT. This descriptor contains extra information on application life cycle indicating in particular if an application is appropriate to use in conditions of trick-mode playback. It indicates whether this application shall or shall not be recorded, when a program, along with which this application is signalled, is recorded. It provides a means to specify the locations of data resources that shall be recorded along with the application, as well as the labels of the object carousel modules of the application that shall, should or should not be recorded.

Table 21: Application recording descriptor syntax

Syntax	No. of bits	Identifier	Comments / Value
application_recording_descriptor (){			
descriptor_tag	8	uimsbf	0x06
descriptor_length	8	uimsbf	
scheduled_recording_flag	1	bslbf	
trick_mode_aware_flag	1	bslbf	
time_shift_flag	1	bslbf	
dynamic_flag	1	bslbf	
av_synced_flag	1	bslbf	
initiating_replay_flag	1	bslbf	
reserved	2	bslbf	
label_count	8	uimsbf	N0
for(i=0;i<N0;i++){			
label_length	8	uimsbf	N1
for(j=0; j<N1; j++) {			
label_char	8	uimsbf	
}			
storage_properties	2	uimsbf	
reserved	6		
}			
component_tag_list_length	8	uimsbf	N2
for(i=0;i<N2;i++){			
component_tag	8	uimsbf	
}			
private_length	8	uimsbf	N3
for(i=0;i<N3;i++){			
private	8	uimsbf	
}			
for(i=0;i<N4;i++){			
reserved_future_use	8	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x06 identifies this descriptor.

scheduled_recording_flag: This single bit flag, when set to '1', indicates that the application is appropriate to record when the service in which it is signalled is recorded by a scheduled recording. When set to '0', it indicates that the application is inappropriate to record by a scheduled recording. Examples of why an application would be inappropriate to record include the application not having been tested in a PVR environment or that the application is closely related to the time of transmission and would be meaningless to the end-user if played back from a recording (e.g. an application tied to a live event).

trick_mode_aware_flag: This single bit flag, if set to '1', indicates that the application is trick-mode aware. If set to '0', the application is not aware of trick-modes.

time_shift_flag: This single bit flag, when set to '1', indicates that the application is appropriate to record when the service in which it is signalled is recorded in time-shift recording mode. When set to '0', it indicates that the application is inappropriate to record in time-shift recording mode.

dynamic_flag: This flag indicates whether the application relies on the use of dynamic data from the broadcast during its execution. When set to '1', it indicates that the application relies on the presence of files (either code or data) or application signalling (e.g. application control code) which change during the lifetime of the piece of content. When set to '0', it indicates that the application does not rely on dynamic data from the broadcast.

NOTE 1: The present document does not define behaviour for terminals that is conditional upon the value of this flag. Platform specifications may use this flag in their determination of whether or not an application is recordable.

av_synced_flag: This flag indicates whether the application requires use of stream events. If set to '1', this is required.

NOTE 2: The present document does not define behaviour for terminals that is conditional upon the value of this flag. Platform specifications may use this flag in their determination of whether or not an application is recordable.

initiating_replay_flag: This single bit flag, if set to '1', indicates that the terminal shall not initiate the playback of the streams located in the same recording as the application. The application is responsible for starting this playback. If set to '0', the implementation shall initiate this playback in parallel with starting the application as would conventionally be the case. This flag shall only be considered when playback of a recording is first started. After this time, the value of this flag shall be ignored.

label_count: This 8-bit field identifies the number of labels that have been used.

label_length: This 8-bit field identifies the number of bytes in the label.

label_char: These 8-bit fields carry an array of bytes that label a part of the application within its transport protocol.

NOTE 3: The present document does not define which parts of applications can be labelled or the form of the label (if any). Platform specifications that wish to use this mechanism need to define the format of labels.

storage_properties: A field indicating the importance of storing the labelled part of the application. Values for this field are defined in table 22.

Table 22: Values for the storage_properties field

storage_properties value	Definition
0	should not be stored
1	critical to store
2	optional to store
3	reserved

component_tag_list_length: This integer specifies the length in number of bytes of the list of component tags.

component_tag: This field identifies a service component that delivers data that is required by the application at playback time and that shall be recorded along with the application and the audio, video and subtitle streams to be recorded.

private: These bytes may be used for private extensions.

reserved_future_use: These reserved bytes may be used for future DVB extensions.

5.3.5.5 Application usage descriptor

The application_usage_descriptor identifies that the application provides a specific, well-known, service; for example teletext, EPG or chat. Terminals may include a shortcut to start these services, for example a remote control key. Terminals may also include a native UI offering access to these services. If no application_usage_descriptor is present then an application does not provide a specific well-known service.

Table 23: Application usage descriptor

	No. of bits	Identifier	Value
application_usage_descriptor() {			
descriptor_tag	8	uimsbf	0x16
descriptor_length	8	uimsbf	
usage_type	8	uimsbf	
}			

descriptor_tag: This 8 bit field with value 0x16 identifies the descriptor

usage_type: This 8 bit field indicates which service is provided by the application. It shall be coded according to table 11.

5.3.5.6 User information descriptors

The user information descriptors complement the application_descriptor by providing information suitable for presentation to the user (where the application_descriptor provides technical information for automatic use by the receiver).

These descriptors are defined for use in the "application" (inner) descriptor loop of the AIT.

5.3.5.6.1 Application name descriptor

Exactly one instance of this descriptor shall be included in the "application" (inner) descriptor loop. The application name shall distinguish the application and shall be informative to the user.

Table 24: Application name descriptor syntax

	No.of Bits	Identifier	Value
application_name_descriptor() {			
descriptor_tag	8	uimsbf	0x01
descriptor_length	8	uimsbf	
for (i=0; i<N; i++) {			
ISO_639_language_code	24	bslbf	
application_name_length	8	uimsbf	
for (i=0; i<N; i++) {			
application_name_char	8	uimsbf	
}			
}			
}			

descriptor_tag: This 8 bit integer with value 0x01 identifies this descriptor.

ISO_639_language_code: This 24-bit field contains the ISO 639-2 2 [7] three character language code of the language of the following application name. Both ISO 639.2/B and ISO 639.2/T may be used.

Each character is coded into 8 bits according to ISO 8859 1 [8] and inserted in order into the 24-bit field.

application_name_length: This 8 bit unsigned integer specifies the number of bytes in the application name.

application_name_char: This field carries one character of a string (not null terminated) of characters encoded in accordance with annex A of EN 300 468 [1]. The string names the application in a manner intended to be informative to the user. Specific application types may impose additional restrictions on the encoding of this value.

5.3.5.6.2 Application icons descriptor

Zero or one instance of this descriptor shall be included in the "application" (inner) descriptor loop. It allows icons to be associated with the application.

Table 25: Application icons descriptor syntax

	No.of Bits	Identifier	Value
application_icons_descriptor() {			
descriptor_tag	8	uimsbf	0x0B
descriptor_length	8	uimsbf	
icon_locator_length	8	uimsbf	
for (i=0; i<N; i++) {			
icon_locator_byte	8	uimsbf	
}			
icon_flags	16	bslbf	
for (i=0; i<N; i++) {			
reserved_future_use	8	bslbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x0B identifies this descriptor.

icon_locator_length: This 8 bit integer specifies the number of bytes in the string that prefixes standard icon file name.

icon_locator_byte: This 8 bit value is one byte of the icon locator string. See clause 5.2.8 "Application icons".

icon_flags: This 16-bit field identifies the size and aspect ratio of icons available for this application. See clause 5.2.8 "Application icons".

5.3.5.7 External application authorization descriptor

The "common" (first) descriptor loop of the Application Information Table may contain zero or more external_application_authorization_descriptors. Each descriptor contains information about external applications that are allowed to continue to run with the applications listed in this Application Information Table sub-table but cannot be launched from this service. The external authorization applies to applications with the identified application_identifier() that are of the application_type identified by the AIT subtable where this descriptor is contained.

Table 26: External application authorization descriptor syntax

	No.of Bits	Identifier	Value
external_application_authorisation_descriptor() {			
descriptor_tag	8	uimsbf	0x05
descriptor_length	8	uimsbf	
for(i=0; i<N; i++) {			
application_identifier()			
application_priority	8	uimsbf	
}			
}			

descriptor_tag: This 8-bit integer with value 0x05 identifies this descriptor.

application_identifier(): This 48-bit field identifies an application. The structure of this field is defined in clause 5.2.3 "Application identification".

application_priority: This 8-bit integer specifies the priority that this application assumes in the context of the current service.

If the 0xffff or 0xfffe wildcard is used for the application_id within the application_identifier() and there are applications from the same organisation_id explicitly signalled in the application loop of the AIT, the priority for those applications shall be the one signalled in the application_descriptor (see clause 5.3.5.3).

See application_priority under clause 5.3.5.3 "Application descriptor".

5.3.5.8 Graphics constraints descriptor

The graphics_constraints_descriptor defines the circumstances under which an application can work (or has been tested to work). These circumstances are:

- which full screen graphics resolutions an application supports.
- whether an application can work when its video is controlled (e.g. scaled to less than full screen size) by another application not signalled as part of the current service (e.g. an EPG, a navigator, or an unbound application running as part of an abstract service).

This descriptor may be present either in the "application" (inner) loop of an AIT in which case it applies to only that application or the "common" (outer) loop of an AIT in which case it applies to all applications signalled in that AIT sub-table.

Table 27: Graphics constraints descriptor syntax

	No. of bits	Identifier	Value
graphics_constraints_descriptor() {			
descriptor_tag	8	uimsbf	0x14
descriptor_length	8	uimsbf	
reserved_future_use	5	bslbf	
can_run_without_visible_ui	1	bslbf	
handles_configuration_changed	1	bslbf	
handles_externally_controlled_video	1	bslbf	
for(i=0;i<N;i++) {			
graphics_configuration_byte	8	uimsbf	
}			
}			

Where the fields have the following meanings;

descriptor_tag: This 8 bit integer with value 0x14 identifies this descriptor.

descriptor_length: This 8 bit field indicates the number of bytes following the descriptor length field.

can_run_without_visible_ui: This single bit flag indicates whether the application can run without a visible UI. See clause 5.2.9.1.2 "Running without a visible UI".

handles_configuration_changed: This single bit flag indicates whether the application can support changes in the terminal's graphics configuration. See clause 5.2.9.1.3 "Handling changed graphics configurations".

handles_externally_controlled_video: This single bit flag indicates whether the application requires control over the presentation of video in the same service. See clause 5.2.9.1.4 "Handling externally controlled video".

graphics_configuration_byte: These 8 bit fields contains a value specified in clause 5.2.9.1.1 "Supported graphics configurations".

5.3.6 Transport protocol descriptors

The transport_protocol_descriptor identifies the transport protocol associated with a service component and possibly provides protocol dependent information.

The descriptor may be used in either the "common" (outer) descriptor loop or the "application" (inner) descriptor loop. When in the "common" loop it applies to all of the applications in that sub-table. Any such descriptors in the "application" loop describe additional transport protocols available to a specific application.

Each application shall be in the scope of at least one transport_protocol_descriptor.

Table 28: Transport protocol descriptor syntax

	No.of Bits	Identifier	Value
transport_protocol_descriptor() {			
descriptor_tag	8	uimsbf	0x02
descriptor_length	8	uimsbf	
protocol_id	16	uimsbf	
transport_protocol_label	8	uimsbf	
for(i=0; i<N; i++) {			
selector_byte	8	uimsbf	N1
}			
}			

descriptor_tag: This 8 bit integer with value 0x02 identifies this descriptor.

protocol_id: An identifier of the protocol used for carrying the applications. The values of the protocol_id are registered in the present document and at http://www.dvb.org.

Table 29: Protocol_id

protocol_id	Description
0x0000	reserved_future_use
0x0001	Object Carousel as defined in annex B of the present document.
0x0002	reserved
0x0003	Transport via HTTP over the interaction channel as defined in clause 7.2.
0x0004 to 0x00FF	Reserved for use by DVB
0x0100 to 0xFFFF	Subject to registration at http://www.dvb.org.

transport_protocol_label: This 8 bit field uniquely identifies a transport protocol within this AIT section. The application_descriptor refers to this value to identify a transport connection that carries the application.

selector_byte: Additional protocol specific information.

Table 30: Semantics of selector bytes

protocol_id	Selector byte data
0x0000	reserved_future_use
0x0001	See clause 5.3.6.1, "Syntax of selector bytes for OC transport".
0x0002	reserved
0x0003	See clause 5.3.6.2, "Syntax of selector bytes for interaction channel transport"
0x0004 to 0xFFFF	Not defined in this version of the present document

5.3.6.1 Syntax of selector bytes for OC transport

When the protocol ID is 0x0001 the selector bytes in the transport_protocol_descriptor shall be as shown in table 31.

Table 31: Syntax of selector bytes for OC transport

Syntax	Bits	Identifier
remote_connection	1	bslbf
reserved_future_use	7	bslbf
if(remote_connection == "1") {		
original_network_id	16	uimsbf
transport_stream_id	16	uimsbf
service_id	16	uimsbf
}		
component_tag	8	uimsbf

remote_connection: This single bit flag if set to "1" indicates that the transport connection is provided by a broadcast service that is different to the one carrying the AIT. Such applications shall not be autostarted by receivers but are visible (subject to the visibility field of the application descriptor and the availability of an API for discovering signalled applications) for possible launching by service selection (but not via an application launching API). When this bit is set, the following 3 fields (original_network_id, transport_stream_id and service_id) are included in the selector bytes. This flag shall be set to "0" when the transport connection is provided by the current service.

Applications with this flag set shall either have their application control code set to REMOTE (see table 3), or they shall have an application_storage_descriptor with "launchable_completely_from_cache" set to "1" (see clause 5.2.11.1.3).

Applications where remote_connection is "1" that also have an application_storage_descriptor with "launchable_completely_from_cache" set to "1" (see clause 5.2.11.1.3) are a special case. If such an application is cached on the terminal, it can be launched in the usual way. There are no special restrictions on the control code for an application signalled in this way - e.g. it could be PRESENT or even AUTOSTART. If the application is not cached on the terminal, it cannot be launched and the signalled control code will be ignored - it will always be treated as if it was REMOTE.

Remote applications can be cached and stored in the usual way if an application first tunes the network interface to the appropriate transport stream.

original_network_id: This 16 bit field identifies the DVB SI original network id of the transport stream that provides the transport connection.

transport_stream_id: This 16 bit field identifies the MPEG transport stream id of the transport stream that provides the transport connection.

service_id: This 16 bit field identifies the DVB-SI service id of the service that provides the transport connection.

component_tag: Identifies the "principal" service component that delivers the application. The identified component is the elementary stream that carries the DSI of the object carousel.

5.3.6.2 Syntax of selector bytes for interaction channel transport

When the protocol ID is 0x0003 the selector bytes in the transport_protocol_descriptor shall be as shown in table 32 "Syntax of selector bytes for interaction transport". This allows encoding of a number of URLs. The descriptor can also be used in a simplified form where only one URL is encoded.

For efficiency when encoding possibly many similar URLs the encoding divides the URL into a shared base part and a set of URL extensions. The set of URLs can identify ZIP [16] files, or base URLs ending in the "/" character, that encapsulate portions of the file system.

Multiple transport protocol descriptors with the protocol ID value 0x0003 and the same transport protocol label may be provided to define a larger set of URLs to describe the file system.

Table 32: Syntax of selector bytes for interaction transport

Syntax	Bits	Identifier
for(i=0; i<N; i++){		
URL_base_length	8	uimsbf
for(j=0; j<N; j++){		
URL_base_byte	8	uimsbf
}		
URL_extension_count	8	uimsbf
for(j=0; j<URL_extension_count; j++){		
URL_extension_length	8	uimsbf
for(k=0; k<URL_length; k++){		
URL_extension_byte	8	uimsbf
}		
}		
}		

URL_base_length

This 8-bit field provides the number of bytes in the base part of the URL.

URL_base_byte

These bytes form the first part of a HTTP URL conforming to HTTP 1.0 (see RFC 1945 [18]), or the first part of an HTTPS URL conforming to RFC 2818 [19] or the first part of another URL conforming to RFC 3986 [14].

URL_extension_count

This 8-bit field indicates the number of URL extensions conveyed by this descriptor.

URL_extension_length

This 8-bit field indicates the number of bytes in the extension part of the URL.

URL_extension_byte

These bytes form the later part of an HTTP URL conforming to HTTP 1.0 (see RFC 1945 [18]), or the later part of an HTTPS URL conforming to RFC 2818 [19] or else a URL whose scheme is supported by a registered interaction channel transport service provider implementation.

URLs are formed by concatenating the URL extension with the preceding URL base. The URL so formed either identifies a file system directory or a specific ZIP file.

In the simplified form, the following apply:

- Exactly one base URL shall be encoded.
- The URL formed by URL_base_byte shall be a URL ending with a slash ("/") character. References to ZIP files are not permitted.
- URL_extension_count shall be zero.
- Only one transport_protocol_descriptor with protocol_id 0x0003 shall be present in the scope of the application.

5.3.7 Simple application location descriptor

One instance of this descriptor shall be contained in the "application" (inner) descriptor loop of the AIT for each application.

Table 33: Simple application location descriptor syntax

	No.of Bits	Identifier	Value
simple_application_location_descriptor () {			
descriptor_tag	8	uimsbf	0x15
descriptor_length	8	uimsbf	
for(i=0; i<N; i++) {			
initial_path_bytes	8	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x15 identifies this descriptor.

initial_path_bytes: These bytes contain a string specifying the URL path component to the entry point document.

5.3.7.1 Example

The following example describes the usage of the simple_application_location_descriptor.

An application author designs an application in the following manner:

- The application data is distributed among several directories, for instance an "image" directory and a "main" directory.
- The application entry point is a document called "index.foo" and stored in the "main" directory.

From the application author's point of view, the application entry point is specified by the path "main/index.foo". This path is stored in the initial_path_bytes string of the location descriptor.

Table 34: Examples showing application entry point signalling for different protocol_id values

protocol_id value	Selector	Resulting application entry point
0x0001	Component tag, e.g. 0xb4	dvb://1.2.3.b4/main/index.foo
0x0003	Base URL, e.g. "http://www.example.com/apps"	http://www.example.com/apps/main/index.foo

If the broadcaster chooses to insert this application in a file system sub-directory called "application", the initial_path_bytes shall be prefixed with the string "application/", i.e. initial_path_bytes shall have the value "application/main/index.foo".

5.3.8 Simple application boundary descriptor

This descriptor is defined for use in the application loop of the AIT. It provides a set of prefixes that describe the data elements that form the application.

This descriptor is optional. When absent, the application boundary defaults to the complete set of all content coming from the transport signalled in the transport_protocol_descriptor associated with the application. This can be overridden by the platform specification.

Multiple boundary descriptors can be used for the same application. In this case, the applicable set of extensions is the union of the set of extensions defined by the descriptors.

Table 35: Simple application boundary descriptor syntax

	No.of Bits	Identifier	Value
simple_application_boundary_descriptor {			
descriptor_tag	8	uimsbf	0x17
descriptor_length	8	uimsbf	
boundary_extension_count	8	uimsbf	
for(j=0; j<boundary_extension_count; j++){			
boundary_extension_length	8	uimsbf	
for(k=0; k<boundary_extension_length; k++){			
boundary_extension_byte	8	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x17 identifies this descriptor.

boundary_extension_count: This 8-bit field indicates the number of boundary extensions conveyed by this descriptor.

boundary_extension_length: This 8-bit field indicates the number of bytes in the boundary extension.

boundary_extension_byte: These bytes form a URL prefix. Any URLs which match this prefix are considered to be within the application boundary. Note that the URL prefix is a strict prefix (e.g. 'http://www.example.com' instead of 'www.example.com') and may include components of a path (e.g. 'http://www.example.com/epg/'). Platform specifications may define a minimum level of granularity given by the prefix.

5.3.9 Service information

5.3.9.1 Data broadcast descriptor for interactive application announcement

The generic data_broadcast_descriptor is defined in EN 300 468 [1]. This clause defines the syntax and semantics of the selector bytes when the data broadcast id has the value 0x00F2 (see table 36). In this case the selector bytes provide a list of interactive applications and information about each application. Zero or more instances of this descriptor may be listed in the SDT or the EIT to identify interactive applications associated with the service or the event where the descriptor is present. This descriptor only indicates the association between the service or event and the applications. The location of each listed application shall be resolved through the AIT. This descriptor shall not list applications where the test_application_flag is (or will be) set in the corresponding entry in the AIT.

Table 36: Syntax of extended data broadcast descriptor - broadcast id 0xF2

	No.of Bits	Identifier	Value
data_broadcast_descriptor(){			
descriptor_tag	8	uimsbf	
descriptor_length	8	uimsbf	
data_broadcast_id	16	uimsbf	
component_tag	8	uimsbf	
selector_length	8	uimsbf	
for(i=0; i<selector_length; i++){			
organization_id	32	uimsbf	
application_id	16	uimsbf	
reserved_future_use	1	bslbf	
application_type	15	uimsbf	
application_profile_length	8	uimsbf	
for (j=0; j<N; j++){			
application_profile	16	uimsbf	
version.major	8	uimsbf	
version.minor	8	uimsbf	
version.micro	8	uimsbf	
}			
application_names_length	8	uimsbf	
for(j=0; j<N2;j++){			
ISO_639_language_code	24	bslbf	
application_name_length	8	uimsbf	
for(l=0; l<N3; l++){			
application_name_char	8	bslbf	
}			
}			
reserved_length	8	uimsbf	
for(j=0; i<N4; i++){			
reserved_future_use	8	bslbf	
}			
private_data_length	8	uimsbf	
for(j=0; j<N5; j++){			
private_data_byte	8	bslbf	
}			
}			
ISO_639_language_code	24	bslbf	
text_length	8	uimsbf	
for (i=0; i<text_length; i++){			
text_char	8	uimsbf	
}			
}			

Semantics of the data broadcast descriptor:

The semantics for the following elements of the syntax are defined in EN 300 468 [1]:

descriptor_tag: For this 8-bit field see EN 300 468 [1].

descriptor_length: For this 8-bit field see EN 300 468 [1].

data_broadcast_id: For this 16-bit field see EN 300 468 [1]. This field has the value 0x00F2 (see table 38) when announcing interactive applications (regardless of the transport method(s) used for the interactive application and data).

component_tag: For this 8-bit field see EN 300 468 [1].

selector_length: For this 8-bit field see EN 300 468 [1].

The semantics for the following elements of the syntax are defined in the present document:

organization_id: This is 32-bit field encodes the organisation_id of the application. See clause 5.2.3 "Application identification".

application_id: This is 16-bit field encodes the ID of the application. See clause 5.2.3 "Application identification".

application_type: This is 15-bit field encodes the type of the application. See clause 5.2.2 "Application types".

application_profile_length: This 8-bit field indicates the length of the application profile loop in bytes.

application_profile: This 16-bit field identifies which application type specific profile is required by this application. See clause 5.2.5 "Platform profiles".

version.major: This 8-bit field indicates the major version number of the profile. See clause 5.2.5 " Platform profiles".

version.minor: This 8-bit field indicates the minor version number of the profile. See clause 5.2.5 " Platform profiles".

version.micro: This 8-bit field indicates the micro version number of the profile. See clause 5.2.5 " Platform profiles".

application_names_length: This 8-bit unsigned integer specifies the number of bytes in the following multilingual application names.

ISO_639_language_code: This is 24-bit field encodes the ISO_639_language_code of the application name. See clause 5.3.5.6.1 "Application name descriptor".

application_name_length: This is 8-bit field encodes the length of the application name. See clause 5.3.5.6.1 "Application name descriptor".

application_name_char: See application_name_char in clause 5.3.5.6.1 "Application name descriptor".

reserved_length: This 8-bit unsigned integer specifies the number of reserved bytes that follow.

reserved_future_use: This is an 8-bit field.

private_data_length: This 8-bit unsigned integer specifies the number of private data bytes that follow.

private_data_byte: This is an 8-bit field.

The semantics for the following elements of the syntax are defined in EN 300 468 [1]:

ISO_639_language_code: For this 24-bit field see EN 300 468 [1].

text_length: For this 8-bit field see EN 300 468 [1].

text_char: For this 8-bit field see EN 300 468 [1].

5.3.10 Stored applications

5.3.10.1 Application storage descriptor

This application_storage_descriptor advertises that an application can be stored and provides some indications of its properties. The presence of this descriptor indicates that an Application Description File is provided for the application (see clause 5.2.11). For a storable application a single application_storage_descriptor shall be placed in either the "common" (outer) descriptor loop or "application" (inner) descriptor loop of the AIT.

This descriptor, and the implied Application Description File, also supports receivers that implement speculative caching.

Table 37: Syntax of application storage descriptor

	No.of Bits	Identifier	Value
application_storage_descriptor() {			
descriptor_tag	8	uimsbf	0x10
descriptor_length	8	uimsbf	
Storage_property	8	uimsbf	
not_launchable_from_broadcast	1	bslbf	
launchable_completely_from_cache	1	bslbf	
is_launchable_with_older_version	1	bslbf	
reserved	5	bslbf	
reserved	1	bslbf	
version	31	uimsbf	
priority	8	uimsbf	
}			

descriptor_tag: This 8 bit integer with value 0x10 identifies this descriptor.

descriptor_length: This 8 bit field indicates the number of bytes following the descriptor length field.

storage_property: This 8 bit field indicates whether the application is broadcast related or stand alone.

launchable_completely_from_cache: See clause 5.2.11.1.3 "Launching applications from the cache".

is_launchable_with_older_version: See clause 5.2.11.1.2 "Application versioning".

not_launchable_from_broadcast: See clause 5.2.11.1.3 "Launching applications from the cache ".

version: See clause 5.2.11.1.2 "Application versioning".

priority: See clause 5.2.11.1.4 "Storage priority".

5.4 XML-based syntax

This clause defines an XML encoding for the AIT in addition to the MPEG-2 table and section based encoding defined in clause 5.3 of the present document. Since the intended use for this XML encoding is in conjunction with the SD&S defined in TS 102 034 [6], this encoding follows the same format and re-uses already defined elements and types.

Of the features in the MPEG-2 encoding, the signalling of graphics constraints is not supported in the XML encoding.

The semantics of the fields defined in this clause shall be identical to those of the corresponding fields in the existing MPEG-2 table and section based encoding as defined by the present document.

Monitoring for changes in the XML-based AIT shall be performed as defined in clause 5.4.3 of TS 102 034 [6].

The MIME type used for the XML encoding of the AIT shall be application/vnd.dvb.ait+xml. The file extension shall be ".aitx".

5.4.1 Service bound application signalling

Service bound applications shall be signalled by including an ApplicationList element in either the IPService or the Package elements of SD&S (see TS 102 034 [6]). This is fully specified in clauses 5.4.3.2 and 5.4.3.1 respectively of the present document.

Applications are either defined inline within the ApplicationList element or in an ApplicationDiscovery record as defined in clause 5.4.5. In the latter case the ApplicationIdentifier is used in the ApplicationList to reference the application in the ApplicationDiscovery record.

Alternatively, a service may include an MPEG-2 format AIT in-band in the stream. Announcement that the transport stream includes this AIT is signalled in the IPService element. However, inclusion of an MPEG-2 format AIT in-band in the stream prohibits the use of the XML encoding of the AIT.

5.4.2 Signalling of unbound applications

Unbound applications (i.e. applications which are not associated with a specific service) shall be signalled by including one or more AbstractIPService elements in the SD&S service provider discovery record (see TS 102 034 [6]). This is fully specified in clause 5.4.3.3.

Applications are either defined inline within the AbstractIPService's ApplicationList element or in an ApplicationDiscovery record as defined in clause 5.4.5. In the latter case the ApplicationIdentifier is used in the AbstractIPService's ApplicationList element to reference the application in the ApplicationDiscovery Record.

5.4.3 Extensions to defined SD&S elements

5.4.3.1 Package

The application list signalled in the package contains a set of applications which are available for all the IPServices signalled in the package. The ApplicationList is added as an extension to the Package type defined in TS 102 034 [6].

```
<xsd:complexType name="PackageType">
    <xsd:complexContent>
        <xsd:extension base="ipi:Package">
            <xsd:sequence>
                <xsd:element name="ApplicationList" type="mhp:ApplicationList" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

NOTE: Adding an application to a package is semantically the same as adding the application to all services of that package.

5.4.3.2 IP Service

Service bound applications for single services are signalled by an extension of the IPService type defined in TS 102 034 [6].

```
<xsd:complexType name="IPServiceType">
    <xsd:complexContent>
        <xsd:extension base="ipi:IPService">
            <xsd:choice>
                <xsd:element name="ApplicationList" type="mhp:ApplicationList" minOccurs="0"/>
                <xsd:element name="AITDescriptor" type="mhp:AITDescriptorType" minOccurs="0"/>
            </xsd:choice>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AITDescriptorType" />
```

This extension adds an ApplicationList element to the end of the IPService element. The ApplicationList element is an instantiation of the ApplicationList type defined in clause 5.4.4.1 of the present document.

The AIT of the service may be included inline within the transport stream using the MPEG2 syntax. This may be signalled by an AITDescriptor element. If an inline AIT is signalled the ApplicationList element shall not be present.

5.4.3.3 ServiceProvider

Unbound applications are signalled by an extension of the ServiceProviderType type as defined in TS 102 034 [6].

```
<xsd:complexType name="ServiceProviderType">
    <xsd:complexContent>
        <xsd:extension base="ipi:ServiceProviderType">
            <xsd:sequence>
                <xsd:element name="AbstractService" type="mhp:AbstractIPService"
```

```
                    maxOccurs="unbounded" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The extension adds an AbstractService element at the end of the service provider definition. The AbstractService element is an instantiation of the AbstractIPService type defined in clause 5.4.4.15 of the present document.

5.4.4 New XML element definitions

5.4.4.1 ApplicationList

```
<xsd:complexType name="ApplicationList">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="application" type="mhp:Application" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="ApplicationReference" type="mhp:ApplicationIdentifier" minOccurs="0"
            maxOccurs="unbounded">
    </xsd:sequence>
</xsd:complexType>
```

An ApplicationList is a list of Application and/or ApplicationReference elements. An ApplicationReference is an instantiation of an ApplicationIdentifier type defined in clause 5.4.4.3 of the present document. A reference can be resolved by looking for the ApplicationIdentifier in the ApplicationDiscovery records of the same service provider.

5.4.4.2 Application

```
<xsd:complexType name="Application">
        <xsd:sequence>
            <xsd:element name="appName" type="ipi:MultilingualType" maxOccurs="unbounded"/>
            <xsd:element name="applicationIdentifier" type="mhp:ApplicationIdentifier"/>
            <xsd:element name="applicationDescriptor" type="mhp:ApplicationDescriptor"/>
            <xsd:element name="applicationSpecificDescriptor"
                type=" mhp:ApplicationSpecificDescriptor" minOccurs="0"/>
            <xsd:element name="applicationUsageDescriptor"
                type="mhp:ApplicationUsageDescriptor" minOccurs="0"/>
            <xsd:element name="applicationBoundary"
                type="mhp:SimpleApplicationBoundaryDescriptorType"
                minOccurs="0"
                maxOccurs="1" />
            <xsd:element name="applicationTransport"
                type="mhp:TransportProtocolDescriptorType"
                minOccurs="1"
                maxOccurs="unbounded" />
            <xsd:element name="applicationLocation"
                type="mhp:SimpleApplicationLocationDescriptorType"
                minOccurs="1"
                maxOccurs="1" />
        </xsd:sequence>
</xsd:complexType>
```

An application can be completely described by:

- An application name which can be multilingual (appName).
- A unique identification (applicationIdentifier).
- A generic descriptor which is common and mandatory for all types of application (applicationDescriptor).
- An application specific descriptor which will depend upon the type of the signalled application.
- An application usage descriptor which is optional.
- An application boundary descriptor which is optional.
- One or more application transport descriptors.
- A simple application location descriptor.

5.4.4.2.1 Application Specific Information (informative)

Some platform specifications may choose to describe, or require, extra information to be communicated to the terminal that is outside the scope of information currently carried in the Application element, for example additional transport protocol information or security related information. To enable this, the Application type defined in clause 5.4.4.2 can be extended, following the XML extensibility recommendations defined in [17].

5.4.4.3 ApplicationIdentifier

```
<xsd:complexType name="ApplicationIdentifier">
    <xsd:sequence>
        <xsd:element name="orgId" type="xsd:unsignedInt"/>
        <xsd:element name="appId" type="xsd:unsignedShort"/>
    </xsd:sequence>
</xsd:complexType>
```

As defined in clause 5.2.3 "Application identification", an application is uniquely identified by:

- OrgId, a globally unique organization identifier that identifies the organization that is responsible for the application.
- AppId, an application identifier allocated by the organization registered with the organization identifier who decides the policy for allocation within the organization.

5.4.4.4 ApplicationDescriptor

```
<xsd:complexType name="ApplicationDescriptor">
    <xsd:sequence>
        <xsd:element name="type" type="mhp:ApplicationType"/>
        <xsd:element name="controlCode" type="mhp:ApplicationControlCode"/>
        <xsd:element name="visibility" type="mhp:VisibilityDescriptor" minOccurs="0"/>
        <xsd:element name="serviceBound" type="xsd:boolean" default="true" minOccurs="0"/>
        <xsd:element name="priority" type="ipi:Hexadecimal8bit"/>
        <xsd:element name="version" type="ipi:Version"/>
        <xsd:element name="mhpVersion" type="mhp:MhpVersion" minOccurs="0"/>
        <xsd:element name="icon" type="mhp:IconDescriptor" minOccurs="0"/>
        <xsd:element name="storageCapabilities" type="mhp:StorageCapabilities" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

The contents of this complex type are mostly those defined in clause 5.3.5.3 "Application descriptor".

The simple type elements are defined as follows:

visibility: This optional element specifies whether the application is suitable to be offered to the end-user for them to decide if the application should be launched. See clause 5.2.6 "Application visibility".

serviceBound: Whether the application is bound to a service or not as defined by the service_bound_flag. See clause 5.3.5.3 "Application descriptor".

priority: This field identifies a relative priority between the applications signalled in this service. See clause 5.2.7 "Application priority".

version: See clause 5.2.11.1.2 "Application Versioning".

mhpVersion: See clause 5.4.4.8.

icon: Signals the presence of an icon representing the application.

storageCapabilities: This optional element shall be added for giving the receiver the required information to store / cache the application.

5.4.4.5 VisibilityDescriptor

```
<xsd:simpleType name="VisibilityDescriptor">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="NOT_VISIBLE_ALL"/>
        <xsd:enumeration value="NOT_VISIBLE_USERS "/>
        <xsd:enumeration value="VISIBLE_ALL"/>
    </xsd:restriction>
</xsd:simpleType>
```

These values are defined in table 5 in clause 5.2.6.1 "Semantics".

5.4.4.6 IconDescriptor

```
<xsd:complexType name="IconDescriptor">
    <xsd:attribute name="filename" type="xsd:string" use="required"/>
    <xsd:attribute name="size" type="xsd:unsignedShort" use="optional"/>
    <xsd:attribute name="aspectRatio" type="mhp:AspectRatio" use="optional"/>
</xsd:complexType>
```

As defined in clause 5.2.8 "Application icons" the IconDescriptor element serves to signal the presence of an icon representing the application. The size and aspectRatio attributes are defined as optional since they can be determined as defined in table 7.

E.g.

```
<icon filename="dvb.icon.1"/> , size = 32x32 pixel square
```

5.4.4.7 AspectRatio

```
<xsd:simpleType name="AspectRatio">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="4_3"/>
        <xsd:enumeration value="16_9"/>
        <xsd:enumeration value="1_1"/>
    </xsd:restriction>
</xsd:simpleType>
```

These aspect ratios are the set of aspect ratios used in table 7 in clause 5.2.8 "Application icons".

5.4.4.8 MhpVersion

```
<xsd:complexType name="MhpVersion">
    <xsd:sequence minOccurs="1">
        <xsd:element name="profile" type="ipi:Hexadecimal16bit "/>
        <xsd:element name="versionMajor" type="ipi:Hexadecimal8bit "/>
        <xsd:element name="versionMinor" type="ipi:Hexadecimal8bit "/>
        <xsd:element name="versionMicro" type="ipi:Hexadecimal8bit "/>
    </xsd:sequence>
</xsd:complexType>
```

These elements are defined as follows:

profile: See application_profile in clause 5.2.5 " Platform profiles".

versionMajor: See version.major in clause 5.2.5 " Platform profiles".

versionMinor: See version.minor in clause 5.2.5 " Platform profiles".

versionMicro: See version.micro in clause 5.2.5 " Platform profiles".

NOTE: This type is named for historical reasons.

5.4.4.9 StorageCapabilities

```
<xsd:complexType name="StorageCapabilities">
    <xsd:sequence minOccurs="0">
        <xsd:element name="storageProperty" type="mhp:StorageType"/>
    </xsd:sequence>
    <xsd:attribute name="launchableFromBroadcast" type="xsd:boolean" use="required"/>
    <xsd:attribute name="launchableCompletelyFromCache" type="xsd:boolean" use="required"/>
    <xsd:attribute name="launchableWithOlderVersion" type="xsd:boolean" use="required"/>
</xsd:complexType>
```

This descriptor, if present, serves to state whether the application can be stored or cached in the receiver as defined in clause 5.2.11 "Stored applications".

The attributes launchableFromBroadcast, launchableCompletelyFromCache, launchableWithOlderVersion have exactly the same meaning as the flags as defined in clauses 5.2.11.1.2 and 5.2.11.1.3.

5.4.4.10 StorageType

```
<xsd:simpleType name="StorageType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="BROADCAST-RELATED"/>
        <xsd:enumeration value="STANDALONE"/>
    </xsd:restriction>
</xsd:simpleType>
```

See clause 5.2.11.1 "Lifecycle of stored applications".

5.4.4.11 ApplicationType

```
<xsd:complexType name="ApplicationType">
    <xsd:choice>
        <xsd:element name="DvbApp" type="mhp:DvbApplicationType"/>
        <xsd:element name="OtherApp" type="mpeg7:mimeType"/>
    </xsd:choice>
</xsd:complexType>
```

See clause 5.2.2 "Application types".

5.4.4.12 DvbApplicationType

```
<xsd:simpleType name="DvbApplicationType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="DVB-J"/>
        <xsd:enumeration value="DVB-HTML"/>
    </xsd:restriction>
</xsd:simpleType>
```

5.4.4.13 ApplicationControlCode

```
<xsd:simpleType name="ApplicationControlCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AUTOSTART"/>
        <xsd:enumeration value="PRESENT"/>
        <xsd:enumeration value="DESTROY"/>
        <xsd:enumeration value="KILL"/>
        <xsd:enumeration value="PREFETCH"/>
        <xsd:enumeration value="REMOTE"/>
        <xsd:enumeration value="DISABLED"/>
        <xsd:enumeration value="PLAYBACK_AUTOSTART"/>
    </xsd:restriction>
</xsd:simpleType>
```

This descriptor serves to dynamically control application life cycle. The meaning of each one of the enumeration elements, as well as the expected behaviour in the receiver, is fully defined in clause 5.2.4 "Application control codes".

5.4.4.14 ApplicationSpecificDescriptor

```
<xsd:complexType name="ApplicationSpecificDescriptor">
    <xsd:choice>
        <xsd:element name="dvbjDescriptor" type="mhp:DVBJDescriptor"/>
        <xsd:element name="htmlDescriptor" type="mhp:DVBHtmlDescriptor"/>
        <xsd:element name="otherDescriptor" type="mhp:OtherDescriptor"/>
    </xsd:choice>
</xsd:complexType>

<xsd:complexType name="OtherDescriptor" abstract="true"/>
```

This descriptor contains the specific descriptor depending upon the type of application. As well as descriptors defined in the present document, it may also include externally defined descriptors.

NOTE: DVBJDescriptor and DVBHTMLDescriptor are outside the scope of the present document.

5.4.4.15 AbstractIPService

```
<xsd:complexType name="AbstractIPService">
    <xsd:sequence>
        <xsd:element name="svcName" type="ipi:MultilingualType" maxOccurs="unbounded"/>
        <xsd:element name="svcId" type="mhp:Hexadecimal24bit"/>
        <xsd:element name="isAutoSelect" type="xsd:boolean"/>
        <xsd:element name="ApplicationList" type="mhp:ApplicationList" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="Hexadecimal24bit">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9a-fA-F]{1}[1-9a-fA-F]{1}[0-9a-fA-F]{4}"/>
    </xsd:restriction>
</xsd:simpleType>
```

These elements have the following definitions:

svcName: The name of the abstract service.

svcId: An identifier for the abstract service. This shall be unique within the abstract services signalled for a service provider.

isAutoSelect: Flag indicating if the service should be automatically started. If the value of this element is true then the service shall be automatically started when the service provider is selected. If false, it shall not.

5.4.4.16 ApplicationOfferingType

```
<xsd:complexType name="ApplicationOfferingType">
    <xsd:complexContent>
        <xsd:extension base="ipi:OfferingBase">
            <xsd:sequence>
                <xsd:element name="ApplicationList" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Application" type="mhp:Application"
                                maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

This element may be used by a service provider to list Application offerings

5.4.4.17 ServiceDiscovery

```
<xsd:element name="ServiceDiscovery">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element name="ApplicationDiscovery" type="mhp:ApplicationOfferingType"
                maxOccurs="unbounded"/>
```

```
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
```

NOTE: This element forms the root element of an SD&S XML instance document defining one or more Application Discovery records. Other SD&S Offering records, as defined in TS 102 034 [6], are contained under the ServiceDiscovery root element defined in TS 102 034 [6]. The appropriate ServiceDiscovery element is identified in an XML instance document through its namespace prefix.

5.4.4.18 ApplicationUsageDescriptor

```
<xsd:complexType name="ApplicationUsageDescriptor">
    <xsd:sequence>
        <xsd:element name="ApplicationUsage" type="xsd:anyURI" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

See clause 5.2.10.3 for the definition of the semantics of this element.

5.4.4.19 TransportProtocolDescriptorType

```
<xsd:complexType name="TransportProtocolDescriptorType" abstract="true" />
```

This type defines the base class for a transport protocol descriptor. This is an abstract type - subclasses of this type are defined to support specific transport protocols. See clause 5.3.6 for a wider discussion of transport protocol descriptors.

5.4.4.20 HTTPTransportType

```
<xsd:complexType name="HTTPTransportType">
        <xsd:complexContent>
            <xsd:extension base="mhp:TransportProtocolDescriptorType">
                <xsd:sequence>
                    <xsd:element name="URLBase" type="xsd:anyURI"
                        minOccurs="1"
                        maxOccurs="1" />
                    <xsd:element name="URLExtension" type="xsd:anyURI"
                        minOccurs="0"
                        maxOccurs="unbounded" />
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

The HTTP transport protocol descriptor shall be used when an entry point to an application is accessed via HTTP. The actual URL of the entry point is composed from the base URL concatenated with the path provided by the simple application location descriptor.

For URLBase and URLExtension the same semantics shall apply as for URL_base_byte and URL_extension_byte respectively in clause 5.3.6.2 of the present document.

5.4.4.21 OCTransportType

```
<xsd:complexType name="OCTransportType">
  <xsd:complexContent>
    <xsd:extension base="mhp:TransportProtocolDescriptorType">
      <xsd:sequence>
        <xsd:choice maxOccurs="1" minOccurs="0">
          <xsd:element name="DvbTriplet" type="ipi:DVBTriplet">
          </xsd:element>
          <xsd:element name="TextualId" type="ipi:TextualIdentifier">
          </xsd:element>
        </xsd:choice>
        <xsd:element name="ComponentTag" minOccurs="1" maxOccurs="1"
type="mhp:ComponentTagType">
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

An ApplicationType element of type OCTransportType shall be present if the application is delivered via DSMCC Object Carousel. If this application is linked to a service and the carousel is part of that service then the service identifier may be omitted.

For applications not linked to a service, e.g. service provider applications, either the DvbTriplet or the TextualId of a service shall be present.

5.4.4.22 ComponentTagType

```
<xsd:complexType name="ComponentTagType">
  <xsd:attribute name="ComponentTag" type="ipi:Hexadecimal8bit">
  </xsd:attribute>
</xsd:complexType>
```

This type defines the representation of an DVB component tag.

5.4.4.23 SimpleApplicationLocationDescriptorType

```
<xsd:simpleType name="SimpleApplicationLocationDescriptorType">
        <xsd:restriction base="xsd:anyURI" /> </xsd:simpleType>
```

This descriptor is defined in clause 5.3.7.

5.4.4.24 SimpleApplicationBoundaryDescriptorType

```
<xsd:complexType name="SimpleApplicationBoundaryDescriptorType">
        <xsd:sequence>
            <xsd:element name="BoundaryExtension" type="xsd:anyURI"
                minOccurs="1"
                maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>
```

This descriptor provides a set of prefixes that describe the data elements that form the application. See clause 5.3.8 for the semantics of this descriptor.

5.4.5 ApplicationDiscovery record

An ApplicationDiscovery record is an instantiation of the ApplicationOfferingType carried in ServiceDiscovery element as specified by clauses 5.4.4.16 and 5.4.4.17.

The presence and retrieval location of an ApplicationDiscovery record is signalled from the ServiceProviderDiscovery record using the OfferingListType (see clause 5.2.5 of TS 102 034 [6]). The PayloadId of the ApplicationDiscovery record shall be 0xC1.

5.5 Constant values

Table 38: Registry of constant values

Where used	Type	Value	Where Defined	Scope
private data specifier descriptor	descriptor tag	0x5F	PSI and SI tables	SI
Data broadcast id descriptor		0x66	PMT	
Application Signalling Descriptor		0x6F	PMT	
Service identifier descriptor		0x71	SDT	
Caching priority descriptor	descriptor tag	0x71	DII moduleInfo userInfo	EN 301 192 [2] - DVB specification for data broadcasting
Content type descriptor		0x72	BIOP objectInfo (note 1)	
reserved to DVB for future OC descriptors		0x73 to 0x7F	OC	
reserved to DVB for future use	table ID on AIT PID	0x00 to 0x73		The present document
Application Information Table		0x74		
reserved to DVB for future use		0x75 to 0x7F		
reserved for private use		0x80 to 0xFF		
Application descriptor	descriptor tag	0x00	AIT	The present document
Application name descriptor		0x01		
Transport protocol descriptor		0x02		
reserved to DVB for future use		0x03, 0x04		
External application authorization descriptor		0x05		
Application recording descriptor		0x06		
reserved to DVB for future use		0x07 – 0x0A		
Application icons descriptor		0x0B		
reserved to DVB for future use		0x0C – 0x0F		
Application storage descriptor		0x10		
reserved to DVB for future use		0x11 to 0x13		
graphics constraints descriptor (see clause 5.3.5.6 "Graphics constraints descriptor)		0x14		
Simple application location descriptor		0x15		
Application usage descriptor		0x16		
Simple application boundary descriptor		0x17		
reserved to DVB for future use		0x18 to 0x5E		
private data specifier descriptor (note 2)		0x5F		
Subject to registration at http://www.dvb.org		0x60 to 0x7F		
User defined (note 3)		0x80 to 0xFE		

Where used	Type	Value	Where Defined	Scope
DVB Object Carousel	data broadcast id	0x00F0	PMT, AIT	SI
reserved		0x00F1		
DVB application presence		0x00F2	EIT, SDT	SI
reserved to DVB for future use		0x00F3 - 0x00FE	PMT, AIT	SI

NOTE 1: Strictly MessageSubHeader::ObjectInfo in the file message and the bound object info in a file binding of a directory or service gateway message.
NOTE 2: The DVB SI private data specifier descriptor is defined for use in the Application Information Table to introduce private descriptors.
NOTE 3: All user defined descriptors shall be within the scope of a private data specifier descriptor (see clauses 5.3.4.7 "Use of private descriptors in the AIT").

6 Referencing DVB services

6.1 DVB URL syntax and semantics

The syntax and semantics of the "dvb:" URL scheme are defined in [20].

6.2 DVB URL resolution

6.2.1 Service identifier descriptor

Zero or more service_identifier_descriptors may be included in the SDT description of a service. Each such descriptor defines a single textual identifier for the service. The syntax of the textual service identifier is:

```
<service_name> "." <service_provider_domain_name>
```

where:

```
<service_name>
```

is a unique name for the service within the service provider's domain.

```
<service_provider_domain_name>
```

is an Internet DNS domain name that the service provider has rights to control. The organization's administrating the Internet DNS domain names are used as a globally unique registration mechanism that allows these textual service identifiers to be globally unique names.

The <service_name> field shall follow the rules defined for Internet DNS names so that the whole textual service identifier is a valid host name to be used in the Internet DNS as defined in RFC 1035 [21].

An example of a textual service identifier is:

```
movie-channel-1.broadcaster-b.com
```

where "broadcaster-b.com" is an Internet DNS domain owned by the broadcaster and "movie-channel-1" is a unique name for the service assigned by the service provider

NOTE 1: The textual service identifier has the same syntax as an Internet host name and it and it has to be assigned in a domain that the service provider has the rights to control. However, the textual service name for a service is not required to resolve to any IP address using the Internet DNS service and if it does, this version of the present document does not specify any specific services that this host should provide if contacted using the IP protocols.

A single service identifier can be assigned to services in different physical networks even if they have different original_network_id and service_id. A given service identifier shall only be associated with services that are considered to be the same service.

NOTE 2: It is up to the service provider to decide which services are "same" and which are not. For example, two services in two different networks where the service have the same programme content but different regional adverts could be generally considered to be the "same" service. However, this decision is entirely up to the service provider.

More than one service identifier may be allocated to a service instance.

Table 39: Service identifier descriptor

	No.of Bits	Identifier	Value
service_identifier_descriptor () {			
descriptor_tag	8	uimsbf	0x71
descriptor_length	8	uimsbf	
for (i = 0; i < descriptor_length; i++) {			
textual_service_identifier_bytes	8	uimsbf	
}			
}			

descriptor_tag: This 8 bit integer with value 0x71 identifies this descriptor.

textual_service_identifier_bytes: These bytes contain the unique identifier for a service encoded using the normal encoding for text strings in DVB SI.

7 Application transport

7.1 Object carousel

This clause describes the protocol used when broadcast applications are transmitted using the DSM-CC User-to-User Object Carousels.

The present document is based on the following specifications:

- ISO/IEC 13818-1 [3] - MPEG 2 systems.
- ISO/IEC 13818-6 [4] - DSM-CC.
- EN 301 192 [2] - DVB specification for data broadcasting.
- TR 101 202 [i.2] - Implementation Guidelines for Data broadcasting.

With the constraints and extensions described in annex B.

7.2 HTTP

When applications are downloaded using the HTTP protocol, the HTTP 1.1 protocol shall be supported as defined in RFC 2616 [5].

8 Synchronization

8.1 Introduction

The present document supports synchronization to video or audio streams in a service using DSMCC stream events as defined in clause B.2.4.

These can be either:

- "Do-it-now" events as defined in clause B.2.4.2.2. These events are posted to the application as soon as they are received by the terminal.
- Events synchronized to a DVB timeline as defined in clause B.2.4.2.2. The events are posted to the application when the timeline reaches the time signalled for the event.

Platform specifications where synchronization is needed should define which of these are supported in their deployment.

8.2 Referencing

Two mechanisms are defined for referencing sources of stream events from applications:

- By referencing a DSMCC stream event object in an object carousel. This requires the service to contain an object carousel as well as the elementary stream carrying the stream event messages.
- By referencing an XML file containing equivalent information to the DSMCC stream event object as defined by the following schema. This enables synchronization to services carrying the stream event messages but not containing an object carousel. The MIME type "application/vnd.dvb.streamevent+xml" shall be used for these XML files when a MIME type is required.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--W3C Schema generated by XMLSpy v2006 sp2 U (http://www.altova.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:dsmcc="urn:dvb:mis:dsmcc:2009"
    targetNamespace="urn:dvb:mis:dsmcc:2009" elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <xs:complexType name="DsmccType">
        <xs:sequence>
            <xs:element name="dsmcc_object" type="dsmcc:DsmccObjectType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="dsmcc" type="dsmcc:DsmccType"/>
    <xs:complexType name="DsmccObjectType">
        <xs:sequence>
            <xs:element name="stream_event" type="dsmcc:StreamEventType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="component_tag" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="StreamEventType">
        <xs:attribute name="stream_event_id" type="xs:string" use="required"/>
        <xs:attribute name="stream_event_name" type="xs:string" use="required"/>
    </xs:complexType>
</xs:schema>
```

Annex A (informative): Elements defined by the platform specification

A.1 Introduction

This annex describes the elements defined in the present document whose semantics are defined by the platform specification.

A.2 Elements which are defined by the platform specification

Table A.1: Elements which are defined by the platform specification

Clause	Element
5.2.5	Application profiling and profile versioning
5.2.8	Location of application icon (where the app icon locator is relative to)
5.2.8	Encoding of application icons locator
5.2.12	Location of application description file
5.3.4.2	Time in which AIT changes will be detected
5.3.4.2	Minimum repetition rate for each AIT subtable
5.3.4.2	Time in which AIT update will be detected
5.3.5.6.1	Encoding of application name in MPEG-2 application name descriptor
B.2.4.6.1	Time in which updates to the set of timebases will be detected
B.2.4.6.2	Time in which changes to event fire times are detected.

Annex B (normative): Object carousel

B.1 Introduction

This annex describes the constraints and extensions to the specifications listed in clause 7.1 when using DSM-CC User-to-User Object Carousels for the carriage of broadcast applications.

B.1.1 Key to notation

Certain notations are used in the "value" columns of the syntax tables:

Table B.1: Key to notation

Symbol	
+	A value that is "allocated" e.g. configuration parameter of the object carousel server.
*	A value that is "calculated" e.g. a field whose value is calculated by the carousel server as a consequence of the number of bytes in other fields.

B.2 Object carousel profile

In the following clause, the message structures of the object carousels are introduced with associated additional restrictions. Each section contains a table specifying the restrictions on the usage of the fields. The table also indicates the source for these restrictions: the DSM-CC standard, DVB guidelines or a specific restriction for the present document.

For the object carousel messages, also the message syntax is included. **In the syntax tables grey shading indicates parts that the broadcaster may put in, but a terminal compliant with the present document may ignore.**

B.2.1 DSM-CC sections

All object carousels messages are transmitted using DSM-CC section format. The DSM-CC section format is defined in chapter 9.2 of the DSM-CC specification [4].

The DSM-CC standard provides an option to use either a CRC32 or a checksum for detecting bit errors. For the present document, we make the following restriction:

Table B.2: Restrictions on DSM-CC Section format

Field	Restrictions	Source
section_syntax_indicator	1 (indicating the use of the CRC32)	The present document
last_section_number	For sections transporting DownloadDataBlock fragments: - all modules intended to be retrieved shall have the last section number 0xFE: - if last section number = 0xFF receiver behaviour is undefined.	

The maximum section length is 4 096 bytes for all types of sections used in object carousels. The section overhead is 12 bytes, leaving a maximum of 4 084 bytes of payload per section.

B.2.1.1 Sections per TS packet

Parts of no more than four sections shall be delivered in a single TS packet.

B.2.2 Data carousel

This clause defines the content of the data carousel messages when used in the object carousel.

Usage of data carousel descriptors not listed below in a DVB object carousel is not defined by the present document.

B.2.2.1 General

The definitions in table B.3 apply to both the dsmccDownloadDataHeader and the similar dsmccMessageHeader.

Table B.3: Restrictions on DSM-CC DownloadData and Message headers

Field	Restrictions	Source
TransactionId	See clause B.2.5, "Assignment and use of transactionId values"	The present document
AdaptationLength	The terminal may ignore the possible contents of the dsmccAdaptationHeader field.	

B.2.2.2 DownloadInfoIndication

The DownloadInfoIndication is a message that describes a set of modules and gives the necessary parameters to locate the module and retrieve it.

Table B.4: Restrictions on the DII

Field	Restrictions	Source
blockSize	maximum size 4 066 (max. section payload - DDB-header size (18)) The recommended blockSize is 4 066.	DSM-CC (for the definition of blockSize), the present document (for the value)
windowSize	0 (not used for Object Carousels)	DSM-CC
ackPeriod	0 (not used for Object Carousels)	DSM-CC
tCDownloadWindow	0 (not used for Object Carousels)	DSM-CC
tCDownloadScenario	0 (not used for Object Carousels)	DSM-CC
compatibilityDescriptor(): compatibilityDescriptorLength	0 (no compatibility descriptor for Object Carousels)	DSM-CC
PrivateDataLength	The terminal may ignore the possible contents of the privateData field	DVB

B.2.2.3 DownloadServerInitiate

The DownloadServerInitiate is used in the case of object carousels to provide the object reference to the ServiceGateway (i.e. root directory) of the object carousel.

Table B.5: Restrictions on DSI

Field	Restrictions	Source
compatibilityDescriptor(): compatibilityDescriptorLength	0 (no compatibility descriptor for Object Carousels)	DSM-CC
privateData	Contains the ServiceGatewayInfo structure	DSM-CC
serverId	Shall be set to 20 bytes each with the value of 0xFF	DVB/The present document

B.2.2.4 ModuleInfo

The moduleInfo structure is placed in the moduleInfo field of the DownloadInfoIndication of the data carousel. It contains the information needed to locate the module.

Table B.6: Restrictions on the DII moduleInfo field

Field	Restrictions	Source
BIOP::ModuleInfo::Taps	The first tap shall have the "use" value 0x0017 (BIOP_OBJECT_USE). The id and selector fields are not used and the terminal may ignore them. The terminal may ignore possible other taps in the list.	DVB
BIOP::ModuleInfo::UserInfo	The userInfo field contains a loop of descriptors. These are specified in the DVB Data Broadcasting standard and/or the present document. The terminal shall support the compressed_module_descriptor (tag 0x09) used to signal that the module is transmitted in compressed form. The userInfo field may also contain a caching_priority_descriptor and one or more label_descriptors.	DVB/The present document
moduleTimeOut blockTimeOut minBlockTime	These fields are defined in units of µs. An appropriate value shall be explicitly encoded by carousel generation equipment. There is no default value that may be encoded, i.e. 0xFFFFFFFF has no special meaning. Receivers shall not employ an inbuilt default instead of the signalled value, as there is no way to define these without knowledge of the construction of a particular carousel.	The present document

Table B.7: BIOP::ModuleInfo syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOP::ModuleInfo() {				
moduleTimeOut	32	uimsbf	+	
blockTimeOut	32	uimsbf	+	
minBlockTime	32	uimsbf	+	
taps_count	8	uimsbf	N1	≥1
{				
id	16	uimsbf	0x0000	user private
use	16	uimsbf	0x0017	BIOP_OBJECT_USE
assocTag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	
}				
for (j=1; j<N1; j++) {				Possible additional taps that may be ignored by terminals.
id	16	uimsbf	+	
use	16	uimsbf	+	
assocTag	16	uimsbf	+	
selector_length	8	uimsbf	N2	
for (j=0; j<N2; j++) {				
selector_data	8	uimsbf	+	
}				
}				
userInfoLength	8	uimsbf	N3	
for (k=0; k<N3; j++) {				
userInfo_data	8	uimsbf	+	
}				
}				

B.2.2.4.1 Label descriptor

This clause is empty in the present document.

B.2.2.4.2 Caching priority descriptor

To indicate priorities for the objects, a caching_priority_descriptor may be included in the userInfo field of the moduleInfo in the DownloadInfoIndication message.

This descriptor provides a priority value for the caching. The same priority applies for each object in the module. The priority indicated in the descriptor is only a hint to the terminal and implementations may use that in combination with other caching strategies.

The descriptor includes also the transparency level (see clause B.5.2 "Transparency levels of caching") that shall be used by the terminal implementation if it caches objects in this module.

Table B.8: Caching priority descriptor syntax

Syntax	No. of bits	Identifier	Value	Comment
caching_priority_descriptor() {				
descriptor_tag	8	uimsbf	0x71	
descriptor_length	8	uimsbf		
priority_value	8	uimsbf		
transparency_level	8	uimsbf		
}				

descriptor_tag: This 8 bit integer value with 0x71 identifies this descriptor.

priority_value: Indicates the caching priority for the objects within this module. A higher value indicates more importance for caching.

transparency_level: Transparency level that shall be used by the terminal if it caches objects contained in this module. The possible values are listed in table B.9. The semantics of the policies are defined in clause B.5.2 "Transparency levels of caching".

Table B.9: Transparency level values

Value	Description
0	reserved
1	Transparent caching
2	Semi-transparent caching
3	Static caching.
4 to 255	reserved for future use

When this descriptor is not included in the userInfo field of the moduleInfo for a module, the default values that shall be assumed are:

- priority_value: 128.
- transparency_level: 1 (transparent caching).

B.2.2.5 ServiceGatewayInfo

The ServiceGatewayInfo structure is carried in the DownloadServerInitiate message and provides the object reference to the ServiceGateway object.

Table B.10: Restrictions on the ServiceGatewayInfo

Field	Restrictions	Source
BIOP::ServiceGatewayInfo::downloadTaps	The terminal may ignore the downloadTap list.	The present document
BIOP::ServiceGatewayInfo::serviceContextList	The terminal may ignore the service context list.	
BIOP::ServiceGatewayInfo::UserInfo	The terminal may ignore the user info.	

Table B.11: ServiceGatewayInfo() syntax

Syntax	No. of bits	Identifier	Value	Comment
ServiceGatewayInfo(){				
IOP::IOR()			+	See table B.21 "IOP::IOR syntax"
downloadTaps_count	8	uimsbf	N1	software download Taps
for (i=0; i<N1; i++) {				
DSM::Tap()				
}				
serviceContextList_count	8	uimsbf	N2	serviceContextList
for (i=0; i<N2; i++) {				
context_id	32	uimsbf		
context_data_length	16	uimsbf	N3	
for (j=0; j<N3; j++) {				
context_data_byte	8	uimsbf	+	
}				
}				
userInfoLength	16	uimsbf	N5	user info
for (i=0; i<N5; i++) {				
userInfo_data	8	uimsbf	+	
}				
}				

B.2.2.6 Download cancel

There is no semantic for this message in this profile. Receivers may ignore them.

B.2.2.7 DownloadDataBlock

Table B.12: Restrictions on the DDB

Field	Restrictions	Source
moduleId	Module ids are unique within the scope of the object carousel. See ISO/IEC 13818-6 [4], clause 11.2.3.	DSM-CC

B.2.3 The object carousel

B.2.3.1 BIOP Generic Object Message

The BIOP Generic Object Message is a common structure used by all the BIOP (Broadcast Inter-ORB Protocol) messages.

Table B.13: Restrictions on the BIOP Generic Object Message

Field	Restrictions	Source
MessageHeader::byte_order	0 (indicating big-endian byte order)	DVB
MessageSubHeader::objectKey	Maximum length of the key shall be four bytes.	DVB
MessageSubHeader::objectKind	The short three-letter aliases shall be used, plus the null-terminator.	DVB
Access attributes	Access attributes are not transmitted in object carousels	DSM-CC

B.2.3.2 CORBA strings

In a number of places object carousel messages include text strings. These are formatted in accordance with clause 12.3.2 of CORBA/IIOP [9] and using the so-called "CDR-Lite" encoding as described by ISO/IEC 13818-6 [4], clause 5.6.3.4. I.e. the text is preceded by an integer specifying the length of the string and followed by a null terminator. The size of this integer depends on the string concerned and can be seen clearly in the syntax tables that follow. However, for clarity CORBA format strings and the size of their length fields are summarized in table B.14.

Table B.14: Location of CORBA format strings

string	length field size (bits)	location
objectKind_data	32	Table B.16 "BIOP::FileMessage syntax"
objectKind_data id_data kind_data	32 8 8	Table B.19 "BIOP::DirectoryMessage syntax"
objectKind_data	32	Table B.28 "BIOP::StreamMessage syntax"
objectKind_data eventName_data	32 8	Table B.30 "BIOP::StreamEventMessage syntax"
type_id_byte	32	Table B.21 "IOP::IOR syntax"
id_data kind_data	32 32	Table B.25 "Syntax of Lite Options Profile Body with ServiceLocation component"

B.2.3.3 BIOP FileMessage

The BIOP FileMessage is used for carrying file objects.

Table B.15: Restrictions on the BIOP File Message

Field	Restrictions	Source
MessageSubHeader::ObjectInfo	The ObjectInfo may be empty (have a length of zero). If not empty the first 8 bytes of the ObjectInfo shall contain the DSM::File::ContentSize attribute. This is optionally followed by a loop of descriptors. The descriptors defined for possible use in this location are: Content type descriptor	The present document
MessageSubHeader::ServiceContextList	The terminal may skip the possible serviceContextList structures.	

Table B.16: BIOP::FileMessage syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOP::FileMessage() {				
magic	4 x 8	uimsbf	0x42494F50	"BIOP"
biop_version.major	8	uimsbf	0x01	BIOP major version 1
biop_version.minor	8	uimsbf	0x00	BIOP minor version 0
byte_order	8	uimsbf	0x00	Big endian byte ordering
message_type	8	uimsbf	0x00	
message_size	32	uimsbf	*	
objectKey_length	8	uimsbf	N1	1 to 4
for (i=0; i<N1; i++) {				
objectKey_data	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4 x 8	uimsbf	0x66696C00	"fil" type_id alias
objectInfo_length	16	uimsbf	N2	
DSM::File::ContentSize	64	uimsbf	+	objectInfo (note)
for (i=0; i<N2 - 8; i++) {				
descriptor()	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N3	serviceContextList
for (i=0; i<N3; i++) {				
context_id	32	uimsbf		
context_data_length	16	uimsbf	N4	
for (j=0; j<N4; j++) {				
context_data_byte	8	uimsbf	+	
}				
}				
messageBody_length	32	uimsbf	*	
content_length	32	uimsbf	N5	
for (i=0; i<N5; i++) {				
content_byte	8	uimsbf	+	actual file content
}				
}				
NOTE: If present and non-zero, this shall be the same as the content_length of the referenced FileMessage.				

B.2.3.4 Content type descriptor

Zero or one content_type_descriptors can be carried in the file MessageSubHeader::ObjectInfo or the BIOP::Binding::ObjectInfo. Where more than one content_type_descriptor is used they shall express the same content format. Also, the content type (if any) signalled in the directory binding shall be identical to that signalled in the bound file's header. This optional descriptor identifies the media type of the file.

This content type signalling only applies to objects of type file and is not appropriate for other object types.

The format of the content_type_descriptor is shown in table B.17.

Table B.17: Content type descriptor syntax

Syntax	No. of bits	Identifier	Value	Comment
content_type_descriptor() {				
descriptor_tag	8	uimsbf	0x72	
descriptor_length	8	uimsbf		
for (i=0; i<descriptor_length; i++) {				
content_type_data_byte	8	uimsbf		A MIME type
}				
}				

descriptor_tag: This 8-bit integer with value 0x72 identifies this descriptor.

descriptor_length: This 8-bit integer identifies the number of bytes following it.

content_type_data_byte: These bytes form a string that indicates the MIME content type of the object. The string is specified as follows:

content_type_data = type "/" subtype *(";" parameter)

Where type, subtype and parameter are as defined in section 5 of RFC 2045 [10] and hence content_type_data carries the payload of the Content-Type header defined in RFC 2045 [10].

B.2.3.5 BIOP DirectoryMessage

The BIOP DirectoryMessage is used for carrying the directory objects.

Table B.18: Restrictions on the BIOP Directory Message

Field	Restrictions	Source
MessageSubHeader::ObjectInfo	The terminal may skip the N2 possible bytes in the objectInfo field.	The present document
MessageSubHeader::ServiceContextList	The terminal may skip the N3 possible serviceContextList structures.	The present document
BIOP::Name	The name shall contain exactly one NameComponent. The id_length shall be 2 or greater. The id_data shall not be replicated for other name components within this directory.	The present document
BIOP::Binding::BindingType	Either "ncontext" (in the case of a Directory object) or "nobject" (in the case of a File or a Stream object). Binding type "composite" shall not be used.	DVB
BIOP::Binding::ObjectInfo	The ObjectInfo for bound objects may be empty (have a length of zero). If the bound object is a file and the ObjectInfo is not empty the first 8 bytes of the ObjectInfo shall contain the ContentSize attribute. This is optionally followed by a loop of descriptors. The descriptors defined for possible use in this location are: Content type descriptor.	The present document

Table B.19: BIOP::DirectoryMessage syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOP::DirectoryMessage() {				
magic	4 x 8	uimsbf	0x42494F50	"BIOP"
biop_version.major	8	uimsbf	0x01	BIOP major version 1
biop_version.minor	8	uimsbf	0x00	BIOP minor version 0
byte_order	8	uimsbf	0x00	big endian byte ordering
message_type	8	uimsbf	0x00	
message_size	32	uimsbf	*	
objectKey_length	8	uimsbf	N1	1 to 4
for (i=0; i<N1; i++) {				
objectKey_data	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4 x 8	uimsbf	0x64697200	"dir" type_id alias
objectInfo_length	16	uimsbf	N2 = 0 (note)	objectInfo
for (i=0; i<N2; i++) {				
objectInfo_data	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N3	serviceContextList
for (i=0; i<N3; i++) {				
context_id	32	uimsbf		
context_data_length	16	uimsbf	N4	
for (j=0; j<N4; j++) {				
context_data_byte	8	uimsbf	+	
}				
}				
messageBody_length	32	uimsbf	*	
bindings_count	16	uimsbf	N5	
for (i=0; i<N5; i++) {				Binding
BIOP::Name() {				
nameComponents_count	8	uimsbf	N6 = 1	See table B.16.
for (i=0; i<N6; i++) {				
id_length	8	uimsbf	N7	NameComponent id
for (j=0; j<N7; j++) {				
id_data	8	uimsbf	+	The "/" character shall not be used.
}				
kind_length	8	uimsbf	N8	NameComponent kind
for (j=0; j<N8; j++) {				
kind_data	8	uimsbf	+	as type_id (see table 4-4 in TR 101 202 [i.2])
}				
}				
}				
BindingType	8	uimsbf	+	0x01 for nobject 0x02 for ncontext
IOP::IOR()			+	objectRef see table B.21
objectInfo_length	16	uimsbf	N9	
if(kind_data == "fil"){				
DSM::File::ContentSize	64	uimsbf	+	0 means that file size is not signalled
for (j=0; j<N9 - 8; j++) {				
descriptor_byte	8	uimsbf	+	
}				
}				
else {				
for (j=0; j<N9; j++) {				
descriptor_byte	8	uimsbf	+	
}				
}				
}				
}				
NOTE: See item 2 under 11.3.2.2 "Directory Message Format" in the DSM-CC specification [4]: "the objectInfo field shall be empty".				

B.2.3.6 BIOP ServiceGateway message

The syntax of the BIOP ServiceGateway message is identical to that of the BIOP DirectoryMessage (described above) with the following exceptions:

- The object kind is "srg" rather than "dir".
- Use is made of the service context list.

B.2.3.7 BIOP Interoperable Object References

The Interoperable Object References (IOR) are references to objects and contain the necessary information to locate the object. The IOR structure may contain different options to be able to point to objects that can be reached via different types of connections. For the present document, the use of IORs is limited to references to objects carried in broadcast object carousels. For object carousels, there are two types of object references: one to be used to reference objects carried in the same object carousel and one to be used to reference objects in other object carousels.

Table B.20: Restrictions on the BIOP IOR

Field	Restrictions	Source
IOP::IOR::type_id	Contains the objectKind of the referenced object. A short three-letter aliases shall be used, plus a null-terminator.	The present document
IOP::IOR::taggedProfileList	There shall be at least 1 taggedProfile included in an IOR. For objects carried in a broadcast object carousel, the first taggedProfile shall be either a TAG_BIOP profile or a TAG_LITE_OPTIONS. If the first tagged profile is some other profile, the object is not carried in a broadcast object carousel and the terminal may ignore the object subject to its own capabilities.	The present document

Table B.21: IOP::IOR syntax

Syntax	No. of bits	Identifier	Value	Comment
IOP::IOR {				
type_id_length	32	uimsbf	N1	
for (i=0; i<N1; i++) {				
type_id_byte	8	uimsbf	+	Short alias type_id (e.g. "dir")
}				
taggedProfiles_count	32	uimsbf	N2	Profile bodies
IOP::taggedProfile()				For objects in broadcast carousels: either BIOPProfileBody or LiteOptionsProfileBody
for (n=0; n<N2 - 1;n++) {				
IOP::taggedProfile()				Terminal may ignore other profiles (2...N1) if present
}				
}				

B.2.3.7.1 BiopProfileBody

The BiopProfileBody is used for references to objects within the same object carousel.

Table B.22: Restrictions on the BIOP Profile Body

Field	Restrictions	Source
BiopProfileBody::byte_order	0 (indicating big-endian byte order)	DVB
BiopProfileBody::LiteComponent	The list shall contain exactly 1 BiopObjectLocation and exactly 1 DSM::ConnBinder as the first two components in that order. The terminal may ignore possible other components in the list.	The present document
DSM::ConnBinder	For objects carried in the broadcast object carousel, the first Tap shall be of type BIOP_DELIVERY_PARA_USE. If there is another type of tap in the first position, the terminal may ignore this object reference, as it is a reference for object accessed using another type of protocol (e.g. for return channel use). The terminal may ignore possible other taps in the list.	The present document
DSM::Tap	In the BIOP_DELIVERY_PARA_USE tap, the id field is not used and may be ignored by the terminal.	The present document
DSM::Tap::timeout	This field is defined in units of µs. An appropriate value shall be explicitly encoded by carousel generation equipment. There is no default value that may be encoded, i.e. 0xFFFFFFFF has no special meaning. Receivers shall not employ an in-built default instead of the signalled value, as there is no way to define these without knowledge of the construction of a particular carousel.	The present document

Table B.23: BIOP Profile Body syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOPProfileBody {				
profileId_tag	32	uimsbf	0x49534F06	TAG_BIOP (BIOP Profile Body)
profile_data_length	32	uimsbf	*	
profile_data_byte_order	8	uimsbf	0x00	big endian byte order
lite_component_count	8	uimsbf	N1	
BIOP::ObjectLocation {				
componentId_tag	32	uimsbf	0x49534F50	TAG_ObjectLocation
component_data_length	8	uimsbf	*	
carouselId	32	uimsbf	+	
moduleId	16	uimsbf	+	
version.major	8	uimsbf	0x01	BIOP protocol major version 1
version.minor	8	uimsbf	0x00	BIOP protocol minor version 0
objectKey_length	8	uimsbf	N2	1 to 4
for (k=0; k<N2; k++) {				
objectKey_data	8	uimsbf	+	
}				
}				
DSM::ConnBinder {				
componentId_tag	32	uimsbf	0x49534F40	TAG_ConnBinder
component_data_length	8	uimsbf	N4	
taps_count	8	uimsbf	N3	
DSM::Tap {				
id	16	uimsbf	0x0000	user private
use	16	uimsbf	0x0016	If BIOP_DELIVERY_PARA_USE is provided it shall be the first tap. If there is another type of tap in the first position, the terminal may ignore this object reference, as it is a reference for an object accessed using another type of protocol (e.g. for return channel use).
assocTag	16	uimsbf	+	
selector_length	8	uimsbf	0x0A	
selector_type	16	uimsbf	0x0001	
transactionId	32	uimsbf	*	
timeout	32	uimsbf	*	
}				
for (n=0; n<N4 - 18; n++) {				The terminal may skip over the possible additional taps
additional_tap_byte	8	uimsbf		
}				
}				
for (n=0;n<N6;n++) {			N6=N1 - 2	
BIOP::LiteComponent{				
componentId_tag	32	uimsbf	+	
component_data_length	8	uimsbf	N7	
for (i=0; i<N7; i++) {				
component_data_byte	8	uimsbf		
}				
}				
}				
}				

B.2.3.7.2 LiteOptionsProfileBody

The LiteOptionsProfileBody is used for making links to objects carried in other object carousels. The LiteOptionsProfileBody can be used to make references to objects carried in other carousels within the same Transport Streams or in other Transport Streams. The following constraints are put on the use of the LiteOptionsProfileBody:

- LiteOptionsProfileBody references shall never be used in an IOR which is in the DSI referencing the service gateway.
- The target carousel is never mounted automatically by the implementation.
- The platform specification may define rules for when a LiteOptionsProfileBody is encountered.

Table B.24: Restrictions on the Lite Options Profile Body

Field	Restrictions	Source
LiteOptionsProfileBody::profile_data_byte_order	0 (indicating big-endian byte order)	DVB
LiteOptionsProfileBody::LiteOptionComponents	The list shall contain a ServiceLocation component as the first component. The terminal may ignore possible other components in the list.	The present document
DSM::ServiceLocation	For objects carried in the broadcast object carousel, the service domain NSAP address shall follow the Carousel NSAP address format.	The present document
DSM::ServiceLocation::InitialContext	The terminal may ignore the initial context	The present document

Table B.25: Syntax of Lite Options Profile Body with ServiceLocation component

Syntax	No. of bits	Identifier	Value	Comment
LiteOptionsProfileBody {				
profileId_tag	32	uimsbf	0x49534F05	TAG_LITE_OPTIONS (Lite Options Profile Body)
profile_data_length	32	uimsbf	*	
profile_data_byte_order	8	uimsbf	0x00	big endian byte order
lite_component_count	8	uimsbf	N1	
DSM::ServiceLocation {				
componentId_tag	32	uimsbf	0x49534F46	TAG_ServiceLocation
component_data_length	8	uimsbf	*	
serviceDomain_length	8	uimsbf	0x14	Length of carousel NSAP address
serviceDomain_data()	160	uimsbf	+	See table B.26
CosNaming::Name() {				pathName
nameComponents_count	32	uimsbf	N2	
for (i=0; i<N2; i++) {				
id_length	32	uimsbf	N3	NameComponent id
for (j=0; j<N3 j++) {				
id_data	8	uimsbf	+	
}				
kind_length	32	uimsbf	N4	NameComponent kind
for (j=0; j<N4 j++) {				
kind_data	8	uimsbf	+	as type_id (see table 4.4 in TR 101 202 [i.2])
}				
}				
}				
initialContext_length	32	uimsbf	N5	
for (n=0; n<N5 n++) {				
InitialContext_data_byte	8	uimsbf		
}				
}				
for (n=0;n<N6;n++) {			N6=N1-1	
BIOP::LiteComponent{				
componentId_tag	32	uimsbf	+	
component_data_length	8	uimsbf	N7	
for (i=0; i<N7; i++) {				
component_data_byte	8	uimsbf		
}				
}				
}				
}				

Table B.26: DVB Carousel NSAP Address

Syntax	No. of bits	Identifier	Value	Comment
DVBcarouselNSAPaddress {				
AFI	8	uimsbf	0x00	NSAP for private use
type	8	uimsbf	0x00	Object carousel NSAP Address.
carouselId	32	uimsbf	+	To resolve this reference a carousel_identifier_ descriptor with the same carousel_id as indicated in this field shall be present in the PMT signalling for the service identified below.
specifierType	8	uimsbf	0x01	IEEE OUI
specifierData { IEEE OUI }	24	uimsbf	0x00015A	Constant for DVB OUI
dvb_service_location () {				
transport_stream_id	16	uimsbf	+	This may be set to 0x0000 which indicates that the terminal shall not use the transport_stream_id when locating the service. For any other value then this field shall be used.
original_network_id	16	uimsbf	+	
service_id	16	uimsbf	+	(= MPEG-2 program_number)
reserved	32	bslbf	0xFFFFFFFF	
}				
}				

B.2.3.8 BIOP StreamMessage

Table B.27: Restrictions on the BIOP Stream Message

Field	Restrictions	Source
MessageSubHeader::ObjectInfo	The ObjectInfo field contains the DSM::Stream::Info_T structure and optionally other data after the Stream Info structure. Terminals may ignore the aDescription_bytes in the DSM::Stream::Info_T structure and the possible other object info data following the structure. Broadcasts may set the duration field to zero to indicate undefined duration.	The present document
MessageSubHeader::ServiceContextList	The terminal may skip the possible serviceContextList structures.	The present document
MessageSubHeader::MessageBody	The MessageBody carries a sequence of taps. There shall be at most one tap of use BIOP_PROGRAM_USE. This tap identifies the service that provides the media stream associated with the Stream object (via a deferred_association_tags_descriptor in the PMT). The tap may only reference programs that are broadcast on the same multiplex (i.e. terminals shall not need to tune to a different multiplex in order to receive the referenced media stream). There shall also be at most one tap with use STR_NPT_USE or STR_DVBTIMEL_USE indicating a timebase to be associated with the Stream object. Taps with use STR_DVBTIMEL_USE shall be interpreted according to clause B.2.3.10. Terminals may ignore possible other Taps (such as BIOP_ES_USE).	The present document
NOTE: Use of NPT is obsolete and not defined in the present document.		

Table B.28: BIOP::StreamMessage syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOP::StreamMessage() {				
magic	4 x 8	uimsbf	0x42494F50	"BIOP"
biop_version.major	8	uimsbf	0x01	BIOP major version 1
biop_version.minor	8	uimsbf	0x00	BIOP minor version 0
byte_order	8	uimsbf	0x00	big endian byte ordering
message_type	8	uimsbf	0x00	
message_size	32	uimsbf	*	
objectKey_length	8	uimsbf	N1	1 to 4
for (i=0; i<N1; i++) {				
objectKey_data	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	8	uimsbf	0x73747200	"str" type_id alias
objectInfo_length	16	uimsbf	N2	
DSM::Stream::Info_T {		uimsbf		objectInfo
aDescription_length	8	uimsbf	N3	aDescription
for (i=0; i<N3; i++) {				
aDescription_bytes	8	uimsbf	+	
}				
duration.aSeconds	32	simsbf	+	may be set to 0 to indicate undefined
duration.aMicroSeconds	32	uimsbf	+	may be set to 0 to indicate undefined
audio	8	uimsbf	+	Flag: 0x00 = false, non-zero = true.
video	8	uimsbf	+	Flag: 0x00 = false, non-zero = true.
data	8	uimsbf	+	Flag: 0x00 = false, non-zero = true.
}				
for (i=0; i<N2-(N3+10); i++) {				
objectInfo_byte	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N4	serviceContextList
for (i=0; i<N4; i++) {				
context_id	32	uimsbf		
context_data_length	16	uimsbf	N5	
for (j=0; j<N5; j++) {				
context_data_byte	8	uimsbf	+	
}				
}				
messageBody_length	32	uimsbf	*	
taps_count	8	uimsbf	N6	
for (i=0; i<N6; i++) {				
id	16	uimsbf	(note)	see clause B.2.4.4
use	16	uimsbf	+	see clause B.2.3.10 and table 4.12 in DVB Guidelines for Data Broadcasting [i.2]
assocTag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	no selector
}				
}				

NOTE: If the tap use is STR_DVBTIMEL_USE then the value of this 16 bit integer corresponds to the value of the broadcast_timeline_id field of the DVB Timeline that defines the timebase for this stream. For other values of tap use the value of this field is undefined.

B.2.3.9 BIOP StreamEventMessage

Table B.29: Restrictions on the BIOP StreamEvent Message

Field	Restrictions	Source
MessageSubHeader::ObjectInfo	The ObjectInfo field contains the DSM::Stream::Info_T and DSM::Stream::EventList_T structures followed optionally by other object info data (which may be ignored by terminals). See table B.27 regarding the DSM::Stream::Info_T. Terminals may ignore the possible other data following the DSM::Stream::EventList_T. The EventList_T defines a sequence of event names that correlates to the sequence of event ids in the MessageBody. eventNames_count shall equal eventIds_count.	The present document
MessageSubHeader::ServiceContextList	The terminal may skip the possible serviceContextList structures.	The present document
MessageSubHeader::MessageBody	The MessageBody carries a sequence of taps followed by a sequence of event ids. The sequence of taps follows the following rules: • There shall be at most one tap of use BIOP_PROGRAM_USE. This tap identifies the service that provides the media stream associated with the Stream object (via a deferred_association_tags_descriptor in the PMT). The tap may only reference programs that are broadcast on the same multiplex (i.e. terminals shall not need to tune to a different multiplex in order to receive the referenced media stream). • There shall be at most one tap with use STR_NPT_USE or STR_DVBTIMEL_USE indicating a timebase to be associated with the StreamEvent object. Taps with use STR_NPT_USE shall be interpreted as described in ISO/IEC 13818-6 [4]. Taps with use STR_DVBTIMEL_USE shall be interpreted according to clause B.2.3.10. • There shall be at most one tap with use STR_EVENT_USE STR_STATUS_AND_EVENT_USE or STR_DVBEVENT_USE. This tap indicates the PID where event data relating to the StreamEvent object is broadcast. Terminals may ignore possible other taps (such as BIOP_ES_USE).	The present document
NOTE: The use of NPT is obsolete and is not defined in the present document.		

Table B.30: BIOP::StreamEventMessage syntax

Syntax	No. of bits	Identifier	Value	Comment
BIOP::StreamEventMessage() {				
magic	4 x 8	uimsbf	0x42494F50	"BIOP"
biop_version.major	8	uimsbf	0x01	BIOP major version 1
biop_version.minor	8	uimsbf	0x00	BIOP minor version 0
byte_order	8	uimsbf	0x00	big endian byte ordering
message_type	8	uimsbf	0x00	
message_size	32	uimsbf	*	
objectKey_length	8	uimsbf	N1	
for (i=0; i<N1; i++) {				
objectKey_data	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4 x 8	uimsbf	0x73746500	"ste" type_id alias
objectInfo_length	16	uimsbf	N2	
DSM::Stream::Info_T {		uimsbf		
aDescription_length	8	uimsbf	N3	aDescription
for (i=0; i<N3; i++) {				
aDescription_bytes	8	uimsbf	+	see BIOP StreamMessage
}				
duration.aSeconds	32	simsbf	+	see BIOP StreamMessage
duration.aMicroSeconds	32	uimsbf	+	see BIOP StreamMessage
audio	8	uimsbf	+	see BIOP StreamMessage
video	8	uimsbf	+	see BIOP StreamMessage
data	8	uimsbf	+	see BIOP StreamMessage
}				
DSM::Event::EventList_T {				
eventNames_count	16	uimsbf	N4	
for (i=0; i<N4; i++) {				
eventName_length	8	uimsbf	N5	
for (j=0; j<N5; j++) {				
eventName_data	8	uimsbf	+	(including zero terminator)
}				
}				
}				
for (i=0; i<N2 - (N3 + 14 + N4 + sum(N5)); i++) {				
objectInfo_byte	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N6	
for (i=0; i<N6; i++) {				
context_id	32	uimsbf		
context_data_length	16	uimsbf	N7	
for (j=0; j<N7; j++) {				
context_data_byte	8	uimsbf	+	
}				
}				
messageBody_length	32	uimsbf	*	
taps_count	8	uimsbf	N8	
for (i=0; i<N8; i++) {				
id	16	uimsbf	(note)	see B.2.4.4
use	16	uimsbf	+	see clause B.2.3.10 and table 4.12 in DVB Guidelines for Data Broadcasting [i.2]
assocTag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	no selector
}				
eventIds_count	8	uimsbf	N4	(= eventNames_count)
for (i=0; i<N4; i++) {				
eventId	16	uimsbf	+	
}				
}				

Syntax	No. of bits	Identifier	Value	Comment
NOTE: If the tap use is STR_DVBTIMEL_USE, the value of this 16-bit integer corresponds to the value of the broadcast_timeline_id field of the DVB Timeline that defines the timebase for this stream. If the tap use is STR_DVBEVENT_USE, the value of this 16-bit integer field corresponds to the value of the synchronised_event_context of all events relevant to this stream. For other values of tap use the value of this field is undefined.				

B.2.3.10 Additional tapUse values

Two additional tapUse values are defined for use in referencing DVB synchronized auxiliary data TS 102 823 [13] from an Object Carousel as shown in table B.31.

Table B.31: tapUse values for referencing DVB synchronized auxiliary data

tapUse	Value
STR_DVBTIMEL_USE	0x8000
STR_DVBEVENT_USE	0x8001

The semantics of the fields of a Tap pointing to a DVB broadcast_timeline_descriptor are described below:

- The use field indicates the use of the Tap. The value of this field shall be STR_DVBTIMEL_USE.
- The value of the id field shall specify the broadcast_timeline_id of the timeline to be referenced.
- The assocTag identifies the connection on which the broadcast_timeline_descriptor is broadcast.
- The selector field shall be empty.

The semantics of the fields of a Tap pointing to a DVB synchronised_event_descriptor are described below:

- The use field indicates the use of the Tap. The value of this field shall be STR_DVBEVENT_USE.
- The value of the id field shall specify the synchronised_event_context of the events to be referenced.
- The assocTag identifies the connection on which the synchronised_event_descriptors are broadcast.
- The selector field shall be empty.

B.2.4 Broadcast timebases and events

Terminals may support DVB Timeline for broadcast timebase delivery.

Terminals are also required to support broadcast events as follows:

- DSM-CC scheduled stream events. These are defined with reference to a broadcast timebase defined using either DSM-CC NPT or the DVB Timeline mechanism.
- DSM-CC "do it now" stream events. These are stand-alone events that are independent of a broadcast timebase.
- DVB synchronized events. These are stand-alone events that are independent of a broadcast timebase but which can provide much greater temporal accuracy than DSM-CC "do it now" events.

The BIOP StreamMessage and StreamEventMessage are used within a DSM-CC object carousel to reference timebases and to define events. A BIOP StreamMessage can only be used to reference a broadcast timebase. A BIOP StreamEventMessage can be used to define broadcast events with or without reference to a broadcast timebase.

NOTE 1: The NPT mechanism and scheduled stream events that depend on it are known to be vulnerable to disruption in many digital TV distribution networks. Existing deployed network equipment that re-generates the STC is unlikely to be aware of NPT and hence will not make the necessary corresponding modification to STC values inside NPT reference descriptors. This may cause stream events scheduled against NPT to fire at the wrong time or to never fire at all. Applications should only use scheduled stream events with NPT when they are confident that the network where they are to be used does not have this problem. DVB Timeline, DSM-CC "do it now" events and events carried within DVB synchronized auxiliary data offer more reliable alternatives to NPT.

NOTE 2: The use of NPT is obsolete and is not defined in the present document.

B.2.4.1 Stream and StreamEvent messages

B.2.4.1.1 Association with time bases

The id field of the STR_DVBTIMEL_USE tap of a StreamMessage or StreamEventMessage identifies the timebase associated with that Stream/StreamEvent object. Multiple StreamMessage or StreamEventMessage may be used at the same time to allow subscriptions to multiple timebases of the same service. See clause B.2.4.4 "Broadcast timebases".

B.2.4.1.2 Event names and event IDs

In StreamEventMessages the EventList_T defines a sequence of event names that correlates 1:1 to the sequence of event IDs in the MessageBody. Within each BIOP::StreamEventMessage the event names uniquely associate to event ID values.

- The eventNames_count shall equal eventIds_count.
- The names in the EventList_T are zero-terminated strings.
- The eventID values in the StreamEventMessage correspond to the eventID values carried in StreamEventDescriptors or the synchronised_event_id values in DVB synchronised_event descriptors.

B.2.4.1.3 Stream event life time

In StreamEventMessages the set of events described in the BIOP::StreamEvent message is possibly a subset of the events that may be used by the application during the course of a programme. Therefore, applications may need to accommodate the dynamic change of such messages. Cache transparency (see clause B.5.2.1 "Transparent caching") and version listener mechanisms provide applications with the means to do this.

Similarly the set of stream event descriptors being transmitted at any time may not correspond to the set of events described in the BIOP::StreamEventMessage.

The event ID for an event name shall not change while the name exists. If a name is removed it shall not be reintroduced within 60 seconds.

B.2.4.2 Stream descriptors

B.2.4.2.1 NPT reference descriptor

NOTE: The use of NPT is obsolete and is not defined in the present document.

B.2.4.2.2 Stream event descriptor

B.2.4.2.2.1 Association of event ids to event time

Where the timebase for a scheduled stream event is provided by NPT, the eventNPT field conveys the NPT value at which the event will occur (or has occurred). Where the timebase for an event is provided by DVB Timeline, the eventNPT field conveys the tick value at which the event will occur (or has occurred) and shall be interpreted in units of the tick_rate used to define the DVB Timeline.

Each StreamEventDescriptor provides a single association between an eventID and an eventNPT. If the terminal detects a change in the eventNPT associated with a value of eventID this redefines the time at which the event should fire.

Terminals shall ignore scheduled events where the eventNPT has passed.

See also clause B.2.4.2.2.3 "Signalling of "do it now events".

NOTE: The use of NPT is obsolete and is not defined in the present document.

B.2.4.2.2.2 Re-use of event ids

Event ID values may be re-used any number of times. For example, after an event has fired then stream event descriptors with the same eventID but different eventNPT may be broadcast.

B.2.4.2.2.3 Signalling of "do it now events"

ISO/IEC 13818-6 [4] is silent on the broadcast signalling of "do it now" events.

These events shall be identified by the value of eventID and hence table id extension (see clause B.2.4.3.5 "Encoding of table id extension").

Where the value of eventID identifies a "do it now" event then the value of eventNPT shall be ignored by the terminal.

B.2.4.2.2.4 Private data

The contents of the privateDataByte field do not need to be interpreted by the terminal. However, the application can access the privateDataByte field using the org.dvb.dsmcc.StreamEvent.getEventData method.

B.2.4.2.3 Unused descriptors

Terminals may ignore the following descriptors if present:

- NPT Endpoint descriptor.
- Stream Mode descriptor.

B.2.4.3 DSM-CC sections carrying stream descriptors

B.2.4.3.1 Section version number

The section version number field increments to reflect changes in stream descriptor(s) carried by sections with the same value of table_id (0x3D) and table_id_extension.

The version number shall increment for reasons including the change in value of eventNPT for a given eventId.

B.2.4.3.2 Single firing of "do it now" events

Terminals shall respond to the first instance of a "do it now" event detected under a particular combination of table id, table id extension and version number. Reception of subsequent copies of the particular event shall be ignored until a different version number is detected.

B.2.4.3.3 Section number

For the present document terminals shall only consider section number zero.

B.2.4.3.4 DSM-CC sections for DSMCC_descriptor_list()

If the table_id field equals 0x3D the current_next_indicator bit shall be set to "1".

B.2.4.3.5 Encoding of table id extension

The section's table id extension field provides information on the stream descriptor(s) carried by the section:

Table B.32: Encoding of table id extension for DSMCC_descriptor_lists

table_id_extension bits			Payload of DSM-CC section with table ID 0x3D
[15]	[14]	[13 to 0]	
0	0	eventID[13 to 0]	Section carries a single "do it now" event.
0	1	xx xxxx xxxx xxxx	Section carries NPT reference descriptors
1	0	xx xxxx xxxx xxxx	Section carries one or more other stream descriptors. I.e. - Stream event descriptor(s) with a future eventNPTs - Stream mode descriptor (can be ignored in the present document) - NPT endpoint descriptor (can be ignored in the present document)
1	1	reserved for future use	
NOTE: The use of NPT is obsolete and is not defined in the present document.			

The value of eventID for "do it now" events shall be in the range 0x0001 to 0x3FFF. The value of eventID for scheduled events shall be in the range 0x8000 to 0xBFFF. The value 0 is not allowed (see clause 5.5.2.2.1 in ISO/IEC 13818-6 [4]).

B.2.4.4 Broadcast timebases

Multiple concurrent timebases may be defined for a single MPEG program but only a single time base is allowed to progress at any instant (the other timebases shall be paused). Timebases can be provided by two means: DSM-CC NPT and DVB Timeline.

NOTE: Use of DSM-CC NPT is obsolete and not defined in the present document.

B.2.4.4.1 DVB Timeline (Optional)

The relationship between each DVB Timeline and the MPEG timebase (STC) is defined using the broadcast_timeline_descriptor carried in DVB Synchronized Auxiliary Data TS 102 823 [13]. The broadcast_timeline_id field of the broadcast_timeline_descriptor (an 8-bit unsigned integer) identifies the timebase.

The value of the id field of the STR_DVBTIMEL_USE tap (a 16-bit unsigned integer) of a StreamMessage or StreamEventMessage identifies the timebase associated with that Stream/StreamEvent object. Multiple StreamMessages or StreamEventMessages may be used at the same time to allow subscriptions to multiple timebases of the same service.

In this profile, broadcast_timeline_descriptors can indicate two states:

- Non-paused. The running_status field set to "Running".
- Paused. The running_status field set to "Paused".

The DVB Timeline referenced may be defined using either the direct or offset encoding method specified in TS 102 823 [13].

Descriptors other than the broadcast_timeline_descriptor may be present within the Synchronized Auxiliary Data stream. Terminals shall skip descriptors which they do not support and continue processing the next descriptor. Not supported descriptors include those defined but not supported (e.g. the TVA_id descriptor in devices not required to support that descriptor by another specification) and descriptors whose tag value is either reserved or user private. See also clause B.2.4.5.3.

The Synchronized Auxiliary Data specification TS 102 823 [13] describes the management of DVB Timeline discontinuities.

Terminals shall comply with the recommendations for receivers in clause D.5 of TS 102 823 [13] including those defined by "should" in that clause. Terminals shall not fire events synchronized to a DVB Timeline in circumstances where the state of the timeline is unknown or where the timeline has been removed from the PMT of the service in which it is carried.

B.2.4.5 Broadcast events

B.2.4.5.1 DSM-CC "do it now" stream events

Terminals shall support DSM-CC "do it now" stream events as defined in clause B.2.4.2.2. This provides a means for delivering stand-alone events without the need for a broadcast timebase.

As these events are delivered in the payload of an MPEG Section, they cannot be accurately synchronized with linear media streams.

B.2.4.5.2 DSM-CC scheduled stream events

Terminals shall support DSM-CC scheduled stream events as defined in clause B.2.4.2.2. These are defined with reference to a broadcast timebase defined using either the DVB Timeline mechanism.

For DSM-CC scheduled stream events, the following usage scenarios are envisaged:

- A single continuous timebase (i.e. a single progressing value of time) can be used. In this case, the broadcast is logically a single continuing interactive production, and the broadcaster is responsible for pre-processing the applications, etc. before broadcast to ensure that they are suitable.
- The signal received by the terminal can include a unique timebase for each programme needing one. This timebase could be suspended during any insertion into a programme and discontinued at the end of the programme.

B.2.4.5.3 DVB synchronized events

Terminals shall support DVB synchronized events as defined by TS 102 823 [13]. This mechanism allows for the accurate generation of events without the need for a timebase.

DVB synchronized events are defined using the synchronized_event_descriptor.

The synchronised_event_context field of this descriptor identifies the application-specific context for a set of events and is referenced using the id field of the STR_DVBEVENT_USE tap of a DSM-CC StreamEventMessage.

The contents of the synchronised_event_data_byte field does not need to be interpreted by the terminal. However, the application can access the synchronised_event_data_byte field using the org.dvb.dsmcc.StreamEvent.getEventData method.

Terminals shall support the cancellation of DVB synchronised events that have not yet fired using the synchronised_event_cancel_descriptor.

Cancellation shall be supported for individual events and for sets of events with a common synchronised_event_context.

Descriptors other than the synchronised_event_descriptor and synchronised_event_cancel_descriptor may be present within the Synchronised Auxiliary Data stream. Terminals shall skip descriptors which they do not support and continue processing the next descriptor. Not supported descriptors include those defined but not supported (e.g. the TVA_id descriptor in devices not required to support that descriptor by another specification) and descriptors whose tag value is either reserved or user private. See also clause B.2.4.4.2.

DVB synchronized events have a specified presentation time, determined by the presentation timestamps of the auxiliary data structures carrying synchronised_event_descriptors and the reference_offset_ticks field of the synchronised_event_descriptors. As such, the events are synchronised to a point in time within linear media streams such as video or audio. Receivers shall deliver events to applications timed so as to retain this synchronization with the linear media.

B.2.4.6 Monitoring broadcast timebases and events

B.2.4.6.1 Timebase reference monitoring

When applications have registered for timebase stimulated events, the terminal shall allocate resources sufficient to ensure that updates to the set of timebases is detected. The time within which updates will be detected is application type dependent.

B.2.4.6.2 Timebase stimulated event monitoring

When applications have registered for timebase stimulated events the terminal shall allocate resources sufficient to ensure that updates to the set of timebase stimulated events is detected. The time within which updates will be detected is application type dependent. So, if an event is introduced or the time at which it is specified to fire is changed then the terminal will respect this change within the period specified. If the fire time for an event changes less than this period before it was previously scheduled to fire then there is no guarantee that all receivers will detect the change in time.

The receiver shall deactivate any event listeners dependant on a timebase (and may free resources associated with those listeners) if:

- the timebase is an NPT timebase and it is deleted (reference to it is removed from the set of NPTReferenceDescriptors);
- the timebase is an NPT timebase and a discontinuity is detected in that timebase;
- a service selection operation changes the current service.

NOTE: The use of NPT is obsolete and is not defined in the present document.

B.2.4.6.3 DSM-CC "do it now" stream events

"do it now" events are single shot events, accordingly terminals need to make special efforts to ensure a high probability that they can be reliably received.

For each application, the terminal is not required to monitor more than a single component delivering "do it now" stream events. So, if events from more than one DSM-CC StreamEventMessage are subscribed to no more than one stream component shall be specified as the source of StreamEventDescriptors carrying "do it now" events (i.e. the taps with use STR_EVENTUSE or STR_STATUS_AND_EVENT_USE shall have the same value when referring to "do it now" events).

Terminals shall dedicate a section filter to monitoring the possible transmission of "do it now" events while there are any applications subscribed to these events.

B.2.4.6.4 DSM-CC scheduled stream events

The stream descriptors for scheduled events are transmitted several times in the period before the time that they should fire. This allows a high probability that they will be effective even if they are not monitored continuously by the terminal.

Any scheduled stream event descriptors shall be transmitted at least once each second.

Terminals shall raise an event in response to a scheduled stream event provided that the stream event descriptors are broadcast for at least 5 seconds before the scheduled time.

For each application, the terminal is not required to monitor more than a single component delivering scheduled stream events. So, if events from more than one DSM-CC StreamEventMessage are subscribed to no more than one stream component shall be specified as the source of StreamEventDescriptors carrying scheduled events (i.e. the taps with use STR_EVENT_USE or STR_STATUS_AND_EVENT_USE shall have the same value when referring to scheduled events).

NOTE: Scheduled and "do-it-now" stream events can be carried on different stream components. The terminal is required to be able to monitor one stream of each.

B.2.4.6.5 Number of timebase components

The terminal is only required to monitor a single timebase component. So, if events from more than one DSM-CC StreamEventMessages are subscribed, each StreamEventMessage that references a timebase shall reference the same type (NPT or DVB Timeline) and shall contain a STR_NPT_USE or STR_DVBTIMEL_USE tap specifying the same association tag.

NOTE: The use of NPT is obsolete and is not defined in the present document.

B.2.4.6.6 DVB synchronized events

DVB synchronized events are transient events (i.e. only broadcast for a short period of time). Accordingly, terminals need to make special efforts to ensure a high probability that they can be reliably received.

For each application, the terminal is not required to monitor more than a single component delivering DVB synchronized events. So, if events from more than one DSM-CC StreamEventMessage are subscribed to, no more than one stream component shall be specified as the source of DVB synchronized events (i.e. the taps with use STR_DVBEVENT_USE shall have the same value of assocTag).

Terminals shall dedicate a filter to monitoring the possible transmission of such events while there are any applications subscribed to them.

Terminals are not required to monitor DSM-CC "do it now" stream events at the same time as monitoring DVB synchronized events.

B.2.5 Assignment and use of transactionId values

B.2.5.1 Informative background

The use of the transactionId in the object carousel is inherited from its use as defined by the DSM-CC specification, and as such it can appear somewhat complex. The transactionId has a dual role, providing both identification and versioning mechanisms for control messages, i.e. DownloadInfoIndication and DownloadServerInitiate messages. The transactionId should uniquely identify a download control message within a data carousel, however it should be "incremented" whenever any field of the message is modified.

NOTE: The term "incremented" is used in the DSM-CC specification. Within the scope of the present document this should be interpreted as "changed".

The object carousel is carried on top of one or more data carousels. By a data carousel used below the object carousel, we mean in the present document a set of DownloadInfoIndication message transmitted on a single PID and the DownloadDataBlock messages carrying the modules described in the DownloadInfoIndication messages. The DownloadDataBlock messages may be spread on other elementary streams than the DownloadInfoIndication messages. The DownloadServerInitiate message in the context of object carousels is considered to be part of the top level of the object carousel and not associated with any data carousel.

When a module is changed, the version number of the module needs to be changed. This implies that the DownloadInfoIndication message that references the module needs to be also updated. Since the DownloadInfoIndication is updated, the transactionId needs to be also changed. However, the transactionId of the DownloadInfoIndication message is used in other messages also, but the need to change the other messages should specifically be avoided and the implications of updating a module should be limited to the module itself and the DownloadInfoIndication that references the module. Therefore, additional rules on the usage of the transactionId have been specified as follows.

B.2.5.2 DVB semantics of the transactionId field

The transactionId has been split up into a number of sub-fields defined in table B.33. This reflects the dual role of the transactionId (outlined above) and constraints imposed to reduce the effects of updating a module. However, to increase interoperability the assignment of the transactionId has been designed to be independent of the expected filtering in target terminals.

Table B.33: Sub-fields of the transactionId

Bits	Value	Sub-field	Description
0	User-defined	Updated flag	This shall be toggled every time the control message is updated
1 to 15	User-defined	Identification	This shall and can only be all zeros for the DownloadServerInitiate message. All other control messages shall have one or more non-zero bit(s).
16 to 29	User-defined	Version	This shall be incremented every time the control message is updated. The value by which it is incremented should be one.
30 to 31	Bit 30 – zero Bit 31 - non-zero	Originator	This is defined in the DSM-CC specification ISO/IEC 13818-6 [4] as 0x02 if the transactionId has been assigned by the network - in a broadcast scenario this is implicit.

Due to the role of the transactionId as a versioning mechanism, any change to a control message will cause the transactionId of that control message to be incremented. Any change to a module will necessitate incrementing its moduleVersion field. This change shall be reflected in the corresponding field in the description of the module in the DownloadInfoIndication message(s) that describes it. Since a field in the DownloadInfoIndication message is changed its transactionId shall be incremented to indicate a new version of the message. Also, any change in the DownloadServerInitiate message implies that its transactionId shall also be incremented. However, when the transactionId is divided into subfields as specified above, updating a message will change only the Version part of the transactionId while the Identification part remains the same.

Since the transactionId is used also for identifying the messages when referencing the messages in other structures, it is very desirable that these referenced would not need to be updated every time the control message is update. Therefore the following rule shall be applied when locating the messages based on the references:

When locating a message based on the transactionId value used for referencing the message, only the Identification part (bits 1 to 15) shall be matched.

Using this rule, the implications of updating a module can be limited to the module itself and the DownloadInfoIndication message describing the module. Also, this implies that if a terminal wants to find out if a particular module that it has retrieved earlier has changed, it needs to filter the DownloadInfoIndication message that described that module and check if it has been changed.

B.2.6 Mapping of objects to data carousel modules

DSM-CC object carousels allow one or more objects to be carried in one module of the data carousel. In order to optimize the performance and memory requirements three additional requirements are specified:

- When mapping objects to modules of a data carousel, only closely related objects should be put into one module. Objects that are not closely related should not be put into the same module. If in the process of retrieving an object from the carousel a terminal acquires a module containing multiple objects, it should attempt to cache these since the expectation should be that the other objects are related to the object requested and probably will be needed soon.

- The size of a module that contains multiple objects should not exceed 65 536 bytes when decompressed (i.e. when the file has been decompressed from the file transport but before the content decoding has started). Terminals complying to the present document are only required to handle modules containing multiple objects where the module size when decompressed is 65 536 bytes or less. Modules containing a single file message can exceed 65 536 bytes with upper size only limited by the memory resources in the terminal.

- In addition to the limitations imposed by the 65 536 byte limit, directory and service gateway messages are limited to 512 object bindings per message.

B.2.7 Compression of modules

The modules may be transmitted either in uncompressed or compressed form. If the module is transmitted in compressed form, this is signalled by including the compressed_module_descriptor in the userInfo field of the moduleInfo in the DownloadInfoIndication message.

Table B.34 shows the syntax of the compressed_module_descriptor:

Table B.34: compressed_module_descriptor

	No. of bits	Identifier	Value
compressed_module_descriptor(){			
descriptor_tag	8	uimsbf	0x09
descriptor_length	8	uimsbf	
compression_method	8	uimsbf	
original_size	32	uimsbf	
}			

Presence of the compressed_module_descriptor indicates that the data in the module has the "zlib" structure as defined in RFC 1950 [11].

The terminal shall support the Deflate compression algorithm as specified in RFC 1951 [12]. This is signalled by setting the least significant nibble of the compression_method to 0x8 (i.e. compression_method is xxxx1000). The terminal is not required to support other compression algorithms.

B.2.8 Mounting an object carousel

The ServiceGateway object is the root directory of the file system delivered by an object carousel and needs to be acquired before any other object can be downloaded. This may be achieved by two compatible mechanisms. The signalling of which mechanisms are being supported by a broadcast is provided by the carousel_identifier_descriptor.

In the present document the use of the carousel_identifier_descriptor for signalling is mandatory in the second descriptor loop of a PMT (corresponding to a PID on which the DSI message for an object carousel is broadcast, i.e. the boot-PID). The consequence is that if a PMT second descriptor loop contains a data_broadcast_id_descriptor that provides signalling for the present document, it shall also contain a carousel_identifier_descriptor.

NOTE: A single PID only contains messages from a single Object Carousel and so only one carousel_identifier_descriptor is present in any second descriptor loop. However, a single service may contain more than one object carousel. Consequently, the carousel_identifier_descriptor may appear more than once in any single PMT.

The acquisition of the ServiceGateway object may be via the standard DSI-DII mechanism. This shall be supported by all broadcasts regardless of signalling in the carousel_identifier_descriptor and shall be sufficient for all terminals.

See also clause 5.2.2 "Program Specific Information".

A broadcast may also contain additional information in the carousel_identifier_descriptor to support the "enhanced" boot mechanism. This is signalled by setting the formatId field for this descriptor to 0x01. This additional information is an aggregation of all the fields necessary to locate the ServiceGateway, also found in the DSI and DII messages. However, in such a case the module containing the ServiceGateway object shall be broadcast on the PID identified by the data_broadcast_id_descriptor. It is optional for both broadcasts and terminals to support this mechanism.

B.2.8.1 carousel_identifier_descriptor

This descriptor is defined by MPEG and in the present document may be included in the second descriptor loop of a PMT.

Table B.35: Carousel identifier descriptor syntax

Syntax	No. of bits	Identifier	Value
carousel_identifier_descriptor {			
descriptor_tag	8	uimsbf	0x13
descriptor_length	8	uimsbf	N1
carousel_id	32	uimsbf	
FormatID	8	uimsbf	
if(FormatID == 0x00) {			
for(i=0; i<N1 - 5; i++){			
private_data_byte	8		
}			
}			
if(FormatID == 0x01) {			
ModuleVersion	8	uimsbf	
ModuleId	16	uimsbf	
BlockSize	16	uimsbf	
ModuleSize	32	uimsbf	
CompressionMethod	8	uimsbf	
OriginalSize	32	uimsbf	
TimeOut	8	uimsbf	
ObjectKeyLength	8	uimsbf	N2 ≤ 4
for(i=0; i<N2; i++){			
ObjectKeyData	8	bslbf	
}			
for(i=0; i<N1 - N2 - 21; i++){			
private_data_byte	8		
}			
}			
}			

carousel_id: This 32 bit field identifies the object carousel with the corresponding carousel_id. If the carousel is intended to be shareable across multiple transport streams, the 24 most significant bits of the carousel_id shall carry the 24 least significant bits of the broadcaster's organisation_id. If the carousel is not intended to be shareable, these 24 bits shall be zero. The remaining 8 least significant bits can take any value.

The carousel_id shall be unique within the program.

FormatID: This 8 bit integer identifies whether the carousel supports the "enhanced boot" mechanism or not. The value 0x00 indicates "standard boot", 0x01 indicates that "enhanced boot" is possible.

ModuleVersion: This 8 bit integer is the version number of the module containing the service gateway. This is equivalent to moduleVersion in the DII.

ModuleId: This 16 bit integer is the identifier of the module in the carousel. This is equivalent to moduleId in the DII.

BlockSize: This 16 bit integer is the size in bytes of every block in the module (except for the last block which may be the same or smaller). This is equivalent to blockSize in the DII.

ModuleSize: This 32 bit integer is the size of the module in bytes. This is equivalent to moduleSize in the DII.

CompressionMethod: This 8 bit field identifies the compression algorithm defined in RFC 1950 [11] used to compress the module. It is equivalent to compression_method carried in the compressed_module_descriptor in the DII.

OriginalSize: This 32 bit integer is the size of the data (in bytes) carried by the module before it was compressed. It is equivalent to original_size carried in the compressed_module_descriptor in the DII.

If the module has not been compressed the values of OriginalSize and ModuleSize shall be equal and the value of CompressionMethod is not defined.

TimeOut: This 8 bit integer specifies the timeout in seconds for acquisition of all blocks of the module.

ObjectKeyLength: This 8 bit integer specifies the number of bytes of ObjectKeyData.

ObjectKeyData: These 8 bit values form an octet string that identifies the BIOP message that is the ServiceGateway message.

B.2.9 Unavailability of a carousel

Broadcast carousels become permanently unavailable due to changes in the signalling including the following:

- The component signalled as carrying the DSI is removed from the PMT.
- The value of carousel ID associated with the carousel changes.
- The program disappears from the PAT.
- After an implementation dependent time general failure of the signalling (e.g. non-transmission of the PMT).

Additionally, carousels also become permanently unavailable when loss of connection to a temporarily disconnected carousel becomes permanent.

B.2.10 Delivery of carousels within multiple services

Carousels shall be considered identical if, in the PMTs of the services, all the following hold:

Either:

a) Both services are delivered within the same transport stream; and

- Both services list the boot component of the carousel on the same PID.
- The carousel_identifier_descriptor for the carousel are identical in both services (so the carousels have the same carousel_Id and boot parameters).
- All association tags used in the carousel map to the same PIDs in both services.

In this case, the carousel is transmitted over a single path, but the services are allowed to reference the carousel via a number of routes, including deferral to a second PMT via deferred association tags.

Or:

b) Both services are delivered over multiple transport streams; and

- The carousel_id in the carousel_identifier_descriptor is in the range of 0x100 - 0xffffffff (containing the broadcaster's organisation_id in the most significant 24 msbs of carousel_id).
- The carousel_identifier_descriptor for the carousel are identical in both services (so the carousels have the same carousel Id and boot parameters).

B.3 AssociationTag mapping

B.3.1 Decision algorithm for association tag mapping

B.3.1.1 TapUse is **not** BIOP_PROGRAM_USE

Figure B.1 illustrates the decision tree for identifying the elementary stream(s) by which the object carousel is distributed:

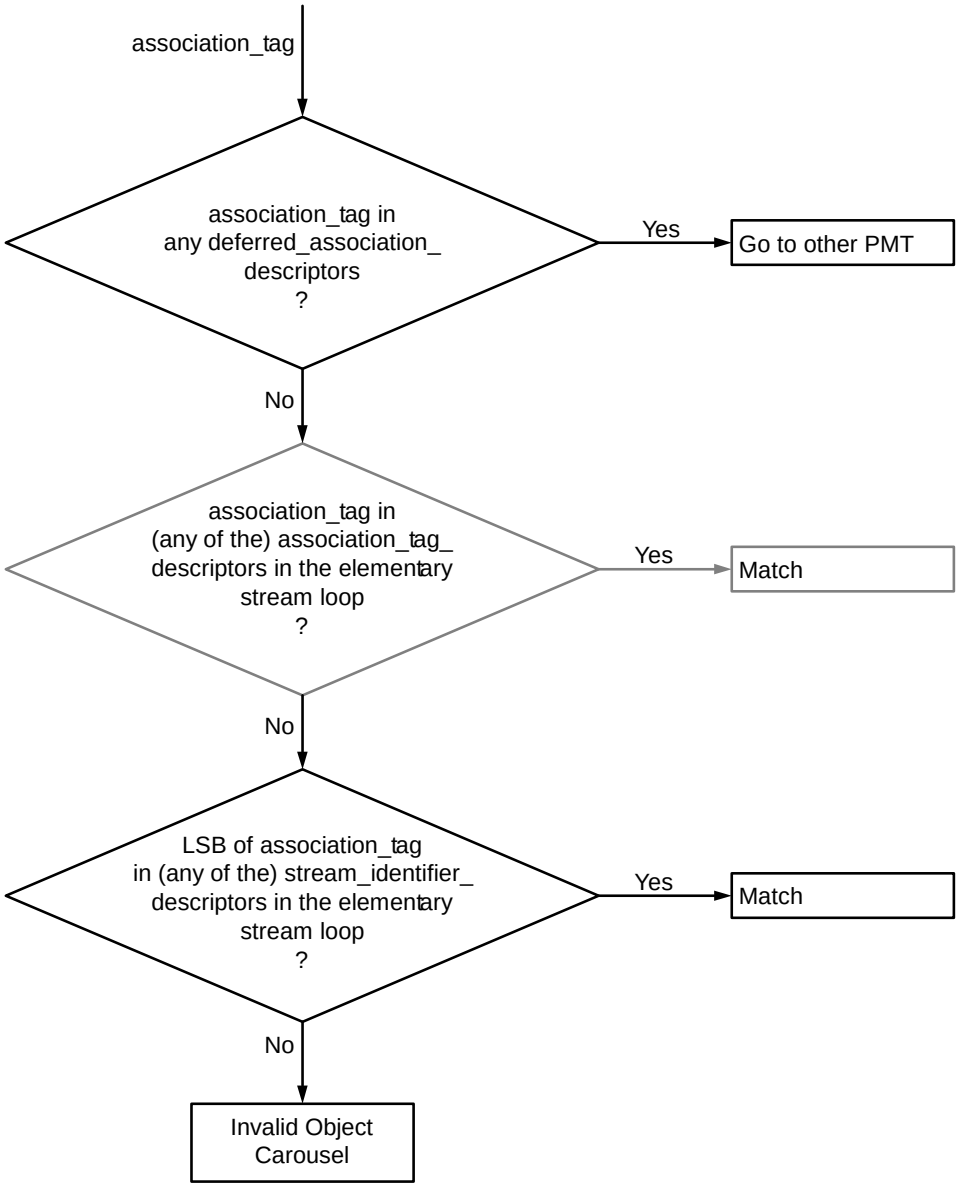

Figure B.1: Object Carousel ES identification decision tree

In the present document, the stream_identifier_descriptor shall always be used for assigning a component_tag for the elementary streams. Use of association_tag_descriptors as defined in DSM-CC [4] is not required. If the association_tag_descriptor is optionally used, a stream_identifier_descriptor (as defined in EN 300 468 [1]) shall still be present and the tag values shall be set consistently in each descriptor. This restriction simplifies the decision tree above so that the second decision can be skipped.

B.3.1.2 TapUse **is** BIOP_PROGRAM_USE

The decision tree in clause B.1 is not followed when resolving a BIOP_PROGRAM_USE tap as the only valid broadcast encoding is for a tap of use BIOP_PROGRAM_USE to resolve to deferred_association_tags_descriptor in the PMT even if the deferred_association_tags_descriptor identify the current service (i.e. stream or streamEvent reference itself). If this resolution fails then the service from which the object carousel is mounted shall be returned as the referenced service.

B.3.2 DSM-CC association_tags to DVB component_tags

The component_tag in a PMT's stream_identifier_descriptor (as defined in EN 300 468 [1]) is used to relate SI service component information with an elementary stream without directly referring to a PID value. Likewise, assocation_tags are used by DSM-CC in order to refer to an elementary stream without directly referencing a PID value. An assocation_tag value is mapped to an elementary stream by matching the LSB of the assocation_tag with a component_tag. The stream_identifier_descriptor is mandatory for all components referenced by an application and/or object carousel.

Broadcasters may choose to use assocation_tag_descriptors (as defined by ISO/IEC 13818-6 [4]) which should (theoretically) be tested for a match before trying component_tags. However, the LSB of the assocation_tag value in an assocation_tag_descriptor has to be equal to the component_tag for that PID. Since the component_tag is unique within a PMT this removes the need to match against assocation_tag_descriptors.

The deferrered_assocation_tags_descriptor required by the present document is the adaptation of the ISO/IEC 13818-6 [4] descriptor defined in TR 101 202 [i.2]. This latter definition standardizes a mechanism to signal the original network id.

When attempting to map an assocation_tag to an elementary stream the assocation_tag shall first be checked against any deferred_association_tags_descriptors in the current PMT (current in this context means the PMT of the service within which the assocation_tag is being mapped). If the association_tag matches any of the association_tags present in a deferred_assocation_tags_descriptor then the matching process proceeds to the service indicated in that descriptor. The terminal is not required to continue its search beyond this second service.

If the transport_stream_id field in the deferred_association_tags_descriptor is set to 0x0000 then it shall be ignored and the terminal is free to choose which transport stream ID it selects when obtaining a service.

B.3.3 deferred_association_tags_descriptor

The transport_stream_id field of the deferred_association_tags_descriptor (as defined in DSMCC [4]) may take value 0x0000 in which case it shall be ignored in resolving the reference.

B.4 Example of an object carousel (informative)

Figure B.2 illustrates an object carousel that is distributed over three elementary streams belonging to the same service.

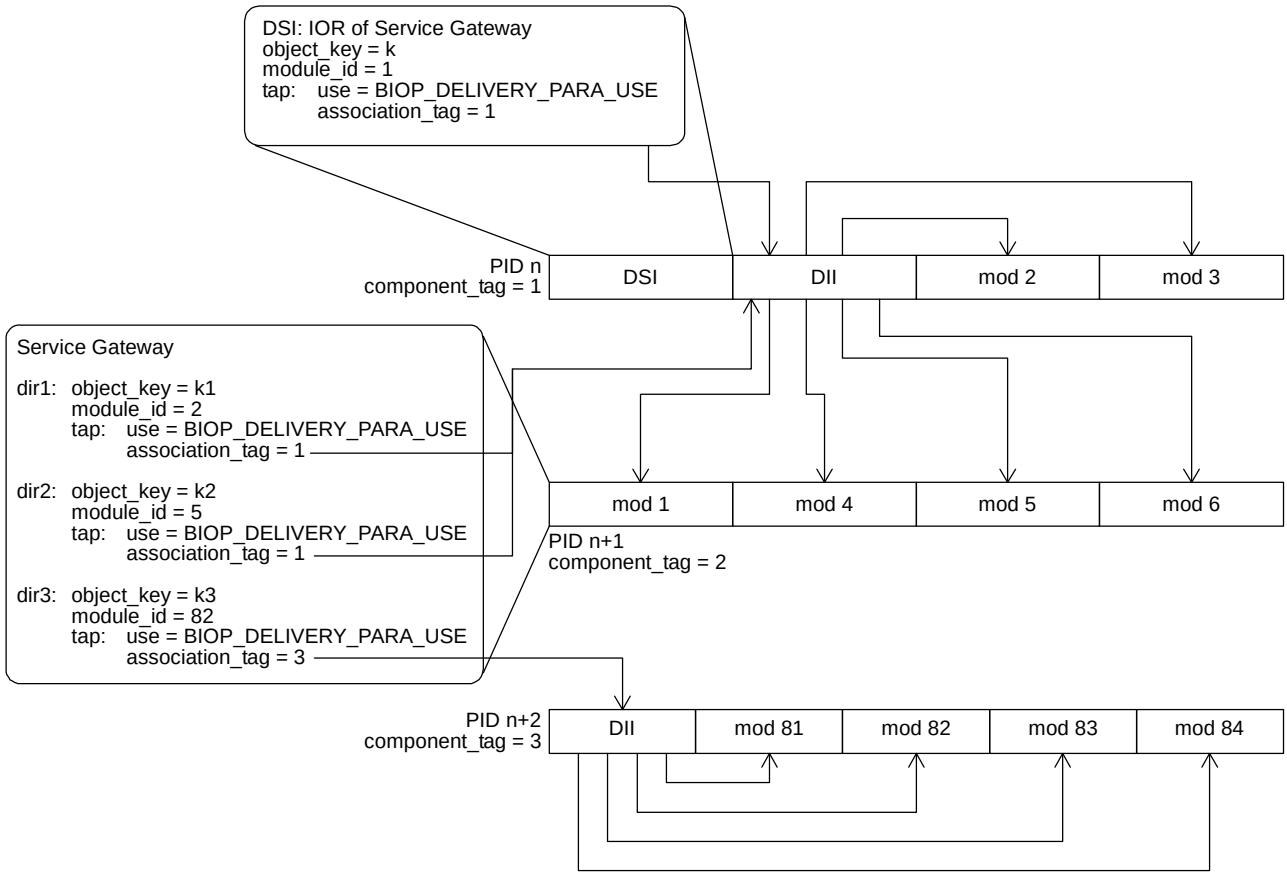

Figure B.2: Example carousel

The DownloadServerInitiate (DSI) message is carried on the first elementary stream. It contains the object reference that points to the ServiceGateway. The tap with the BIOP_DELIVERY_PARA_USE points to a DownloadInfoIndication (DII) message that provides the information about the module and the location where the module is being broadcasted. In the example, the ServiceGateway object is in the module number 1 that is carried on the second elementary stream (indicated by a BIOP_OBJECT_USE tap structure in the DII message).

The ServiceGateway object is a root directory that, in this example, references three subdirectories. Taps with BIOP_DELIVERY_PARA_USE are used in the object references of the subdirectories to provide links to the modules via the DownloadInfoIndication (DII) message. The two first subdirectories "dir1" and "dir2" are referenced in the DII message that is carried in the first elementary stream. The third subdirectory is referenced in the DII message carried in the third elementary stream.

In this example, the two first elementary streams carry the messages of one logical data carousel while the third elementary stream carries the messages of another logical data carousel. All these belong to the same object carousel. In the example, the third elementary stream contains the objects in the "dir3" subdirectory and the objects in the "dir1" and "dir2" subdirectories are distributed over the first and second elementary stream.

NOTE: It is important to note that the third elementary stream may originate from a completely separate source than the first two elementary streams. The directory hierarchy and objects contained in the third elementary stream are "mounted" in the root directory by providing the "dir3" directory entry with the appropriate location information.

This type of structure could be used, for example, in a national information service that contains some regional parts. The common national parts could be carried in this example case on the two first elementary streams that are distributed unmodified in the whole country. The regional parts are carried in the third elementary stream that is locally inserted at each region. From the application's point of view, the common national parts are in the "dir1" and "dir2" subdirectories while the regional parts are in the "dir3" subdirectory.

Another example where this type of structure could be used is if the service contains multiple independent applications. In this case, each application could be placed in its own subdirectory and these subdirectories might be carried as separate data carousels on different elementary streams.

B.5 Caching

This clause describes the constraints that a terminal compliant with the present document shall implement when caching any content from the object carousel in the memory of the terminal. Caching is optional for the terminal, but if implemented shall conform to the constraints set in this clause.

B.5.1 Determining file version

There is no version number directly related to files (or other BIOP messages), the closest association is the moduleVersion in the DII that references the module that contains the BIOP message. Therefore, to ensure that a file is up to date the terminal shall determine that the moduleVersion for the appropriate module is current and reacquire if necessary. The circumstances under which this checking is required are defined by the transparency level as specified in the following clause.

B.5.2 Transparency levels of caching

The definition of transparency levels describes the behaviour that the terminal shall implement when the content in the object carousel is changing. The transparency level determines how certain the terminal is required to be about the validity of the content when returning the content to the application. The object carousel provides a mechanism for determining version changes of the content by monitoring the DII messages.

Validity of content is specified here in terms of the version number of the module that is broadcast in the DII message. The contents of an object as cached in the memory of the terminal are defined to be valid at a certain point in time when the version number of the module in the cache matches the version number of the module as signalled in the DII message describing that module as it was last broadcast.

NOTE: The definition is based on the DII message that was last broadcast and it may be that the terminal was not filtering for this message at that time and did not receive it.

From the terminal point of view, the transparency level indicates the constraints that the terminal needs to implement for monitoring the DII messages.

The broadcaster can indicate the appropriate transparency level that shall be applied for a given piece of content by using a descriptor associated with a module in the DII message (see clause B.2.2.4.2 "Caching priority descriptor"). In the absence of this descriptor from a module, the transparent caching is the default level.

B.5.2.1 Transparent caching

The transparent caching is a caching level that ensures that the application can not practically notice a difference in the validity of the returned content between an implementation that caches content and an implementation that does not cache any content. Naturally, an implementation that caches the content will return it to the application faster.

When returning content from the cache to the application, the terminal shall ensure that the version number of the cached content matches the version number indicated in the current DII message describing that module. Once a DII has been received it can be assumed that it is current at least for 500 ms and after that period until receiving the next instance of the relevant DII. If filtering for that DII has not resumed by the end of this period, the state of that DII is to be considered unknown until it is received again.

Therefore, terminals shall not return transparently cached data if it has waited more than half a second between receiving the relevant DII and starting to filter for that DII again. If the terminal does not resume filtering within the 500ms grace period, it shall download the relevant DII again when it wishes to use that DII to check cache validity.

The choice of 500 ms is based on the normal timing uncertainty in data delivery through the broadcast chain and is independent of the repetition rate of the DII messages.

B.5.2.1.1 Active caching

There are several ways the terminal can organize its caching strategy. One possible strategy is so-called active caching. This means that the terminal has a dedicated section filter for each DII message it needs to monitor. Keeping that filter

continuously filtering for the DII guarantees that the terminal will notice the update of a module as soon as it happens and can thus be aware of the validity of all the content it has cached.

However, in some cases the DII messages might be sent with a very high repetition rate that may cause a high processing load because the terminal needs to do some processing every DII message that it receives. The 500 ms grace period is designed to help this, as it allows the terminal to stop the section filter for 500 ms after receiving the DII message. This lessens the processing burden on the terminal as it only needs to process each DII message twice a second, even if it may be repeated on the transmission much more frequently.

B.5.2.1.2 Passive caching

With active caching, the terminal may need to have a dedicated section filter reserved for each DII message that it needs to monitor. This would effectively limit the amount of content that can be cached, possibly to a very small number. Therefore, the terminal may choose a so-called passive caching strategy. This means that the terminal does not even try to monitor for the DII messages continuously, but each time an application wants to retrieve an object, it at that time retrieves the current DII and checks if the cached content is still valid. Although, this strategy imposes a delay before returning the content to the application, this delay is usually significantly smaller than having to retrieve the content from the broadcast stream.

B.5.2.1.3 DII repetition rate

It should be noted that the description of active and passive caching are only informative here and terminal implementations can use any strategy fulfilling the normative constraints set above. However, broadcasters should set the repetition rate of the DIIs so that a terminal implementing the passive caching strategy will provide the expected benefits of caching over a terminal implementing no caching.

B.5.2.2 Semi-transparent caching

The semi-transparent caching level allows the terminal to cache the data and also return slightly out-dated data to the application. The benefit of this caching level is that it allows terminals to cache larger quantities of content with a reasonable resource usage while allowing the data to be returned usually immediately to the application. The semi-transparent caching level provides less guarantees about validity of the content, but does not cause the delay implied by the passive caching strategy with the transparent caching level.

When returning content from the cache to the application, the terminal shall ensure that the version number of the cached content matches the version number indicated in a valid DII message describing that module. Once a DII has been received it can be assumed to be valid at least for 30 s and after that period until receiving the next instance of the relevant DII. If filtering for that DII has not resumed by the end of this period, the state of that DII is to be considered unknown until it is received again.

Therefore, terminals shall not return semi-transparently cached data if it has waited more than 30 s between receiving the relevant DII and starting to filter for that DII again. If the terminal does not resume filtering within the 30 s grace period, it shall download the relevant DII again when it wishes to use that DII to check cache validity.

B.5.2.2.1 Implications for the terminal (informative)

Reasons for selecting the 30 s value for the grace period in the semi-transparent caching level are different from the reasons for the 500 ms grace period in the transparent level. The 30 s grace period in this level is intended e.g. to allow terminals to keep typically a valid copy of each DII by retrieving each DII in a round robin fashion using a single section filter. Naturally, whether this goal can be achieved, depends on the repetition rate of the DIIs and the amount of content that is cached. If this is not possible, the terminal might use the passive caching strategy with this transparency level as well. These strategies are only examples and the terminal may implement any strategy as long the normative constraints defined above are fulfilled (this includes implementing no caching as it is optional, as well as treating the semi-transparent level the same as the transparent level).

B.5.2.3 Static caching

When using the static caching transparency level, the terminal shall check the validity of the cached content from the version number in the DII message when it is used for the first time during the lifetime of an application instance. After

the first usage time, the terminal does not need to check the validity of the content during the lifetime of that application instance.

B.5.2.3.1 Implications for the broadcaster (informative)

This has the implication, that content with this transparency level is appropriate for very static content that is updated only rarely and where the possible update of the content does not need to be noticed by the application during the lifetime of one application instance.

B.5.2.3.2 Implications for the terminal (informative)

The terminal, however, is allowed to update the contents of the statically cached files if it notices that they have been updated in the carousel as well as use any caching strategy as long as the normative constraint defined above are fulfilled (this includes implementing no caching as it is optional, as well as treating the static level the same as the semi-transparent and/or the transparent level).

B.5.3 Dynamic carousel structure

The Object Carousel may change structure over time, i.e. both files and directories may be added or deleted. Also, modules are not guaranteed to carry the same objects over the lifetime of the carousel. Therefore receivers shall not assume that directory structures are static or that a given path will resolve always to the same object. All cached directory information shall be cached according to the signalled cache priority. This means that before using an object that has been cached, receivers shall validate the path to it.

NOTE: Validating a path does not necessarily mean downloading all elements in the path every time. For example, simply determining that none of the objects on the path have changed since it was last fully traversed is sufficient to confirm that the path itself has not changed.

Annex C (normative): Generic Application Western European Character Set

Table C.1 defines a character set suitable for use by platform specifications addressing the Western European market.

Table C.1: Generic Western European character set

Unicode character code	Character	Unicode script name
0020 to 007E		Basic Latin
00A0 to 00FF		Latin-1 supplement
0100 to 017E		Latin Extended A (excluding 0149, 017F)
01CD	Ǎ	Latin Capital Letter A With Caron
01CE	ǎ	Latin Small Letter A With Caron
02C6	ˆ	Modifier Letter Circumflex Accent
02C7	ˇ	Caron (Mandarin Chinese third tone)
02C9	ˉ	Modifier Letter Macron (Mandarin Chinese first tone)
02D8	˘	Breve
02D9	˙	Dot Above (Mandarin Chinese light tone)
02DA	˚	Ring Above
02DB	˛	Ogonek
02DC	˜	Small Tilde
1E80	Ẁ	Latin Capital Letter W With Grave
1E81	ẁ	Latin Small Letter W With Grave
1E82	Ẃ	Latin Capital Letter W With Acute
1E83	ẃ	Latin Small Letter W With Acute
1E84	Ẅ	Latin Capital Letter W With Diaeresis
1E85	ẅ	Latin Small Letter W With Diaeresis
1EF2	Ỳ	Latin Capital Letter Y With Grave
1EF3	ỳ	Latin Small Letter Y With Grave
2007		Figure Space
2013	–	En Dash
2014	—	Em Dash
2018	‘	Left Single Quotation Mark
2019	'	Right Single Quotation Mark
201A	‚	Single Low-9 Quotation Mark
201C	"	Left Double Quotation Mark
201D	"	Right Double Quotation Mark
201E	„	Double Low-9 Quotation Mark
2022	•	Bullet
2026	…	Horizontal Ellipsis
2030	‰	Per Mille Sign
2039	‹	Single Left-Pointing Angle Quotation Mark
203A	›	Single Right-Pointing Angle Quotation Mark
2044	⁄	Fraction Slash
20AC	€	Euro Sign
2122	™	Trademark Sign
2190	←	Leftwards Arrow
2191	↑	Upwards Arrow
2192	→	Rightwards Arrow
2193	↓	Downwards Arrow
2212	−	Minus Sign
2214	∔	Dot Plus
2215	∕	Division Slash
221E	∞	Infinity
266B	♫	Beamed Eighth Notes
2713	✓	Check Mark
2717	✗	Ballot X

Annex D (informative): Bibliography

"OpenCable Application platform 1.0 Profile"; OC-SP-OCAP1.0-I16-050803.

History

Document history		
V1.1.1	January 2010	Publication
V1.2.1	July 2013	Publication